Maintaining XPath Views in Loosely Coupled Systems

Arsany Sawires †*
arsany@cs.ucsb.edu

Junichi Tatemura ‡
tatemura@sv.nec-labs.com

Oliver Po ‡
oliver@sv.nec-labs.com

Divyakant Agrawal ‡
agrawal@sv.nec-labs.com

Amr El Abbadi †
amr@cs.ucsb.edu

K. Selçuk Candan ‡
candan@sv.nec-labs.com

† Department of Computer Science
University of California Santa Barbara
Santa Barbara, CA 93106

‡ NEC Laboratories America
10080 North Wolfe Road, Suite SW3-350
Cupertino, CA 95014

ABSTRACT

We address the problem of maintaining materialized XPath views in environments where the view maintenance system and the base data system are loosely-coupled. We show that the recently proposed XPath view maintenance techniques require tight coupling, and thus are not practical for loosely-coupled systems. Our solution adapts to loose-coupling by using information that is fully available through standard XPath interfaces. This information consists of the view definition, the update statement, and the current materialized view result. Under this model, incremental maintenance is not always possible; thus, maintaining the consistency of the views requires frequent view recomputations. Our goal is to reduce the frequency of view recomputation by detecting cases where a base update is irrelevant to a view, and cases where a view is self maintainable given a base update. We develop an approach that reduces the irrelevance and self maintainability tests, respectively, to checking the intersection and containment of XPath expressions. We present experimental results showing the effectiveness of the proposed approach in reducing view recomputations.

1. INTRODUCTION

View materialization is widely used in data management systems for improving query performance and for data integration purposes. Therefore, extensive research efforts have addressed the problems introduced by view materialization; such as view maintenance [13] and answering queries using views [14]. With the emergence of XML as a universal semi-structured data model, it has become necessary to study the problems related to view materialization in the XML domain. Thus, the research community has recently shown growing interest in the problem of maintaining views defined over semi-structured data [2, 30, 31, 18, 16, 10, 26, 23, 20, 11], and the problem of answering queries using these materialized views [25, 8, 4, 29, 19, 24].

In this paper we address the problem of maintaining materialized views defined using XPath expressions over XML base data. The view maintenance problem is generally concerned with updating the materialized view result such that it remains consistent with any updates that occur at the base data. We have recently proposed an incremental maintenance approach for XPath views [26]. As we show in Section 2.2, this approach, along with other ones [23, 20, 10], assume that the view maintenance system and the base data system are tightly coupled. This assumption is valid in some settings, e.g. when the views are maintained within the same DBMS as the base data to improve query performance.

The assumption of tight coupling, however, seriously conflicts with the growing interest in seamless data dissemination and application integration over the web. The emergence of the Service Oriented Architecture (SOA) and Web Services [3] is an obvious evidence of the need for building loosely-coupled systems that can seamlessly communicate through universally-standard interfaces.

In this paper, we develop a solution for the problem of maintaining materialized XPath views in environments where the view maintenance system and the base data system are loosely-coupled. This is typical in many scenarios such as middle-tier caching applications. We adapt to loose-coupling by assuming a partial information model in which we use only information that can be communicated through standard XPath interfaces [1]. This information consists of the view definition, the update statement, and the current materialized view result. Both the view definition and the update statement are given as standard XPath expressions.

As we show later, under this partial information model, incremental maintenance is not always possible. Thus, maintaining the views consistency generally requires frequent view recomputation at the base data side, which is usually an expensive operation in terms of communication and processing time. Our goal is to reduce the frequency of view recomputation by detecting cases where a base update is irrelevant to a view (and thus, the update can be ignored), and cases where a view can be self-maintained given a base update (and thus, there is no need for any queries or recomputa-

*This work has been done during the author's internship at NEC Laboratories America.

VLDB '06, September 12-15, 2006, Seoul, Korea.

tions at the base data source).

We develop an update *Irrelevance Test (IRT)*, and a view *Self Maintainability Test (SMT)*. The IRT and the SMT are respectively reduced to checking the intersection [15] and containment [21] of XPath expressions.

If the IRT determines that an update is irrelevant to a view, we say that the *IRT is positive*; otherwise, it is *negative*. The SMT is conducted only if the IRT is negative (because the update can not be ignored); if the SMT determines that the view can be self-maintained, we say that the *SMT is positive*; otherwise it is *negative*. If both the IRT and the SMT are negative, then the view must be recomputed from the base data.

We show that, under the partial information model, the IRT and the SMT cannot be complete. In other words, false negatives are inevitable. Fortunately, false negatives do not compromise the correctness of the approach because they can not result in inconsistent views. Note however that false negatives are undesirable because they would cause unnecessary view recomputation. Therefore, we strive to tighten the derived conditions to reduce the false negatives. On the other hand, the IRT and the SMT must not have any false positives because they can cause the materialized view results to be inconsistent with the base data. In other words, the IRT and the SMT must be sound.

In previous research, the update irrelevance and view self maintainability have been studied in the context of relational data and SQL views [6, 5, 17, 12]. In the context of semi-structured data models, some research has addressed the problem considering limited forms of query languages [31, 16]. To the best of our knowledge, this is the first paper to study the IRT and the SMT for XML data and XPath views.

The rest of this paper is organized as follows: Section 2 presents the formal data and query model, and shows the need for the partial information model in loosely-coupled environments. Section 3 develops the logic for the IRT and the SMT for XPath views. Section 4 discusses the main implementation aspects. Section 5 reports our experimental results showing the effectiveness of the approach in reducing view recomputations. Finally, Section 6 concludes the paper and briefly discusses some possible extensions.

2. PROBLEM MODEL

This section presents the formal model used throughout the paper. Subsection 2.1 presents the data and query model. Subsection 2.2 shows the need for the partial information model in loosely-coupled systems.

2.1 Data and Query Model

Base Data ($\mathcal{BD}$): The base XML data is modeled as an unordered node-labeled tree. The node labels range over an infinite alphabet Σ. The root of the tree is a special node that identifies the XML document. In the examples, we use uppercase letters for the node labels; and we use numeric subscripts to differentiate the nodes that have the same label. Thus, a node label and a numeric subscript together represent a unique node id. In order to allow the application of the proposed approach in situations where schema information is unavailable, or when the schema evolves frequently, we do not assume the availability of any schema information. Figure 1 shows an example of a $\mathcal{BD}$ tree.

XPath Expressions: We consider a practical fragment of XPath, referred to as $XP^{\{[],*,//\}}$ [21]. This fragment

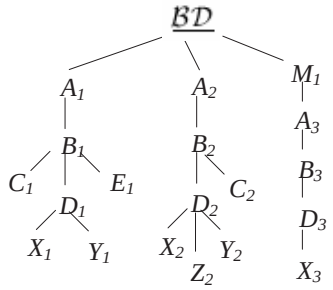


Figure 1: Base Data $\mathcal{BD}$

allows branching predicates "[]", wild cards "*", and descendent edges "//" (besides child edges "/"). An XPath expression $\mathcal{E}$ in this fragment is visually represented as a tree pattern of arity-1 [21], i.e. a tree pattern with a single node designated as the *return node*. This node, referred to as $Ret(\mathcal{E})$, identifies the final result of the expression. The root node of a tree pattern identifies the $\mathcal{BD}$ against which the expression is issued. In the figures, we underline the root node, and we use boldface for the return node. Figure 2 illustrates an expression $\mathcal{V}$ with its pattern tree representation, and its evaluation on the $\mathcal{BD}$ instance of Figure 1.

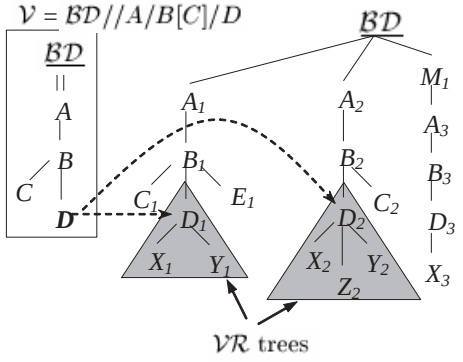


Figure 2: View $\mathcal{V}$, and View Result $\mathcal{VR}$

In the examples throughout the paper, we refer to a specific node of an expression by its label. To avoid confusion, the expressions in the examples do not have multiple nodes with the same label. Note that this is not a required assumption, we use it only for the clarity of the presentation.

For an expression $\mathcal{E}$, we use some formal terms to refer to parts of $\mathcal{E}$. Below, we define these terms and we use the expression $\mathcal{V}$ in Figure 2 as an example:

- $Spine(\mathcal{E})$ is the linear path of nodes and edges starting with the root node and ending with the return node. For example, $Spine(\mathcal{V}) = \mathcal{BD}//A/B/D$. We say that a node or an edge is *spinal* if it belongs to $Spine(\mathcal{E})$ and *non-spinal* otherwise.
- $Prefix_n(\mathcal{E})$ is the expression $\mathcal{E}$ with n being the return node and with all the nodes in the subtree of n removed. For example, $Prefix_B(\mathcal{V}) = \mathcal{BD}//A/B$.
- $Prefix'_n(\mathcal{E})$ is similar to $Prefix_n(\mathcal{E})$, but includes the branching predicates at n. For example, $Prefix'_B(\mathcal{V}) = \mathcal{BD}//A/B[C]$
- $Sub_n(\mathcal{E})$ is the expression represented by the subtree of $\mathcal{E}$ rooted at n. For example, $Sub_B(\mathcal{V}) = B[C]/D$

We define $Res(\mathcal{E})$ as the final result set of an expression $\mathcal{E}$. For example, in Figure 2, $Res(\mathcal{V}) = \{D_1, D_2\}$. We say that an expression $\mathcal{E}$ *reaches* a node m in $\mathcal{BD}$ iff $m \in Res(\mathcal{E})$.

We also use some formal definitions to refer to the intermediate result sets of an expression besides its final result set. We define $Exp_n(\mathcal{E})$ as the expression $\mathcal{E}$ with n being the return node. For example, in Figure 2, $Exp_B(\mathcal{V}) = \mathcal{BD}//A/B[C][D]$. We define the *intermediate result of an expression $\mathcal{E}$ at one of its nodes n* as the result set of the expression $Exp_n(\mathcal{E})$, i.e. $Res(Exp_n(\mathcal{E}))$. For notation simplicity, we refer to this intermediate result set as $Res_n(\mathcal{E})$. For example, $Res_B(\mathcal{V}) = \{B_1, B_2\}$.

For a node n in an XPath expression, we define $e(n)$ as the type of edge (/ or //) leading to n. We also define $n\uparrow$ as the parent node of n, and $n\downarrow$ as the spinal child node of n (defined only if n is spinal). For example, in the expression $\mathcal{V}$ of Figure 2, $e(B) = /$, $B\uparrow = A$, $A\uparrow = \mathcal{BD}$, and $B\downarrow = D$.

XPath Views: A view $\mathcal{V}$ is defined by an XPath expression. The materialized view result $\mathcal{VR}$ is the collection of trees rooted at the nodes of the set $Res(\mathcal{V})$. The shaded trees in Figure 2 form the result of the view $\mathcal{V}$ shown in the same figure.

Base Updates: A base update is defined [1] as a triplet $\langle \mathcal{U}, \mathcal{U}type, \mathcal{U}data \rangle$. $\mathcal{U}$ is an XPath expression that specifies the $\mathcal{BD}$ nodes at which the update should take place. For example, in Figure 1, if $\mathcal{U} = \mathcal{BD}/A/B[E]/D$, then the update operation takes place at D_1 since $Res(\mathcal{U}) = \{D_1\}$. $\mathcal{U}type$ is either Add, Del, or $Modify$; the semantics of each of these update types are defined below and illustrated by examples on the $\mathcal{BD}$ of Figure 1 and the update expression $\mathcal{U} = \mathcal{BD}/A/B[E]/D$.

Base Additions: If $\mathcal{U}type = Add$, then $\mathcal{U}data$ represents an XML data tree to be added as a child to each node that $\mathcal{U}$ reaches. In the example, using $\mathcal{U}data =< Z/ >$ adds a new node with label Z as a child of node D_1.

Base Deletions: If $\mathcal{U}type = Del$, then the update deletes all the nodes that $\mathcal{U}$ reaches along with their subtrees. In the example, using $\mathcal{U}$ as a deletion update, deletes the subtree rooted at D_1.

Base Modifications: If $\mathcal{U}type = Modify$, then the update modifies the labels of all the nodes that $\mathcal{U}$ reaches to some new label specified by $\mathcal{U}data$. In the example, using $\mathcal{U}data = W$ modifies the label of D_1 from D to W.

2.2 The Partial Information Model

First, we point out the technical challenges of applying the XPath incremental maintenance techniques [26, 23, 20, 10] in loosely-coupled environments. Based on that, we propose a practical partial information model which is realistic for these environments. For our first objective, we use a simple example of a typical incremental maintenance scenario.

Example: Consider Figure 2, and consider an update with $\mathcal{U} = \mathcal{BD}/M//B, \mathcal{U}type = Add$, and $\mathcal{U}data =< C/ >$. Before the update happens, $\mathcal{VR}$ is given by the shaded trees in Figure 2. When the update operation is executed, the base query processor will add a node labeled C as a child of node B_3, let the id of the new node be C_3.

In order to report this update operation to the view maintenance system, incremental view maintenance algorithms assume that what is reported is not merely the update statement, i.e. the triplet $\langle \mathcal{U}, \mathcal{U}type, \mathcal{U}data \rangle$, but the *update path*. In this example, the update path is $M_1 - A_3 - B_3 - C_3$. Obviously, this update path carries more detailed information about the update operation than the update statement does: (1) it includes node ids specifying where exactly in the $\mathcal{BD}$ the update took place, and (2) it does not include any of the symbols $*$ or //, which make the update statement less specific.

By analyzing the update path, an incremental maintenance algorithm finds out that a node labeled C is being added as a child of B_3, and thus, it suspects that the predicate of the second step of $\mathcal{V}$ is now true at node B_3. So, it needs to verify if B_3 has any children labeled D to satisfy the third step of $\mathcal{V}$. To find the answer, the algorithm has to issue the query $q = B_3/D$ against $\mathcal{BD}$. Note that q starts with a specific node id, rather than the $\mathcal{BD}$ root. This tells the base query processor to start evaluating q at B_3 rather than at the root of the $\mathcal{BD}$. Thus, the query processor is likely to access a smaller portion of $\mathcal{BD}$, giving the incremental maintenance its main advantage over recomputing $\mathcal{V}$ over the entire $\mathcal{BD}$. If B_3 already had a child labeled C even before the update happened, then the result of q must not be added to $\mathcal{VR}$ because it would be a duplicate of data that is already in $\mathcal{VR}$. To detect such a case, incremental maintenance algorithms require that $\mathcal{VR}$ includes not only node labels but also node ids, so that node ids can be compared to avoid duplication.

This example shows that a general incremental maintenance technique has to make three main assumptions: (1) The base query processor can answer queries that start with node ids rather than the $\mathcal{BD}$ root. (2) $\mathcal{VR}$ includes node ids besides the node labels, and (3) The update path information is available rather than only the update statement.

Satisfying these assumptions require a non-standard interface between the base data system and the view maintenance system. The reason is that the XPath standard [1] does not include node ids in the query statement or in the returned query answer. Also, the update path information, which includes both node ids and intermediate query results, can not be provided to the view maintenance system through standard interfaces.

In fact, none of the major DBMS vendors' XML query processors satisfy any of the three assumption mentioned above. This demonstrates that a practical partial information model is needed for scenarios where the view maintenance system is loosely coupled with the base data system.

In our partial information model, we make none of the three assumptions mentioned above. According to this model, the available information is exactly: (1) the update statement, i.e. the triplet $\langle \mathcal{U}, \mathcal{U}type, \mathcal{U}data \rangle$, (2) the view definition $\mathcal{V}$, and (3) the current view result $\mathcal{VR}$, which includes only node labels without node ids.

Obviously, incremental maintenance is not always possible under this model. For example, the maintenance scenario in the example shown above can not be carried out under this model. Nevertheless, it is possible to use the available partial information to reason about the update irrelevance (IRT) and the view self maintainability (SMT), and thus to avoid unnecessary view recomputation. The following examples give some intuitions behind this reasoning. Section 3, develops the general IRT and SMT for XPath views.

[1]This language is consistent with (and subsumed by) the current W3C working draft for an XML update facility [1].

Example: A positive IRT. Let $\mathcal{V} = \mathcal{BD}/A/B/C$, $\mathcal{U} = \mathcal{BD}/M/B/C$, and $\mathcal{U}type = Del$. Regardless of the content of $\mathcal{BD}$, which is not available to the view maintenance system, it is guaranteed that $\mathcal{U}$ is irrelevant to $\mathcal{V}$, i.e. the IRT is positive. The reason is that any node deleted by $\mathcal{U}$ must have its first ancestor labeled M, while any node in $\mathcal{VR}$ must have its first ancestor labeled A. Since there cannot be any single node labeled M and A at the same time, we are certain that $\mathcal{U}$ can not affect $\mathcal{VR}$. Thus, we can safely ignore the update.

Example: A positive SMT. Let $\mathcal{V} = \mathcal{BD}/A/B/C$, $\mathcal{U} = \mathcal{BD}/A/B/C/D$, and $\mathcal{U}type = Del$. In this case, regardless of the $\mathcal{BD}$, $\mathcal{VR}$ can be maintained using only the available partial information, i.e. the SMT is positive. To see this, recall that $\mathcal{VR}$ for $\mathcal{V}$ is a collection of trees with all the roots labeled C. It is easy to see that every node labeled D that is a child of a root of a $\mathcal{VR}$ tree would be deleted by $\mathcal{U}$. Moreover, $\mathcal{U}$ would not delete any node from $\mathcal{VR}$ other than those identified by this criterion. Therefore, we can use an expression $\mathcal{S} = \mathcal{VR}/C/D$, to reach and delete the exact set of nodes that need to be deleted from $\mathcal{VR}$ in order to incorporate the effect of the base deletion. In this way, we have self maintained $\mathcal{VR}$ using the expression $\mathcal{S}$; we call $\mathcal{S}$ a *Self Maintenance Expression (SME).*

Since the partial information model does not provide complete information about the exact effect of the update operation on a specific instance of the $\mathcal{BD}$, it may not be possible to prove the irrelevance or self maintainability under some base updates. In this case, the IRT and the SMT must act conservatively by returning a negative result. Recall that false negatives are acceptable (but undesirable), while false positives are not acceptable. The following example shows a case of a false negative.

Example: A falsely negative IRT. Consider Figure 2, given an update operation with $\mathcal{U} = \mathcal{BD}//A/B[X]/D$, and $\mathcal{U}type = Del$. This update does not affect $\mathcal{VR}$; in fact it does not even affect $\mathcal{BD}$ at all. However, there is no way of verifying this fact given the available partial information. The reason is that it is possible to have a hypothetical $\mathcal{BD}$ instance with a certain node which both $\mathcal{V}$ and $\mathcal{U}$ could reach. An example of such a document is $< A >< B >< C/ >< X/ >< D/ >< /B >< /A >$. Since the IRT does not have access to the actual $\mathcal{BD}$, it has to act conservatively by returning a negative result.

3. THE IRT AND SMT

In this section, we develop a logic-based approach for the IRT and the SMT for XPath views. For each of the three types of base updates (additions, deletions, and modifications), we identify all the different types of effects that the update could cause to $\mathcal{VR}$. For each type of effect, we derive a necessary condition that must be satisfied in order for this type to take place. If the condition is not satisfied, then it is guaranteed that the corresponding type of effect did not occur; and thus, *the update is irrelevant to the view w.r.t. this type.* On the other hand, if the condition is satisfied, then the update irrelevance can not be concluded; and thus, the IRT is negative.

If the IRT is negative, the SMT is conducted to explore the possibility of writing a correct self maintenance expression (SME) that can be used to update the current $\mathcal{VR}$. An SME is *correct* if it can be used to reflect the exact effects of the base update on $\mathcal{VR}$. For each type of effect, we derive a condition which is sufficient for writing a correct SME. If this condition is satisfied, then the view is guaranteed to be *self maintainable under the given update w.r.t. this type.* In this case, the SME is also given by the test procedure. If the condition is not satisfied, then the SMT is negative; and the view result must be recomputed from the base data.

In the following subsections, we develop the IRT and the SMT logic for each of the three types of base updates: additions, deletions, and modifications.

3.1 Base Additions

A base addition adds a tree $\mathcal{U}data$ to the nodes which $\mathcal{U}$ reaches. Note that only the node labels, without ids, of the $\mathcal{U}data$ are available to the IRT and the SMT.

A base addition can affect $\mathcal{VR}$ in two ways: (1) by adding a subtree as a child of a node that belongs to the current $\mathcal{VR}$; we call this an *internal addition* since it happens inside the current $\mathcal{VR}$, or (2) by adding an entire new $\mathcal{VR}$ tree; we call this an *external addition.* We further classify external additions into two types: *explicit* and *implicit.* An external addition is explicit if the new $\mathcal{VR}$ tree is fully included in $\mathcal{U}data$, and it is implicit otherwise. For simplicity, we will refer to explicit external additions as *explicit additions*, and to implicit external additions as *implicit additions.* The following subsections develop the IRT and SMT logic for the three types of addition effects: internal, explicit, and implicit additions.

It is possible to show that a single base addition may cause any combination of the three types of addition effects. Thus, for a given base addition, the IRT and SMT have to be conducted for all the three types of addition effects.

3.1.1 Internal Additions

For an example of an internal addition, consider the $\mathcal{BD}$, the view $\mathcal{V}$, and the base addition with $\mathcal{U}$ and $\mathcal{U}data$ as shown in Figure 3. In this example, $\mathcal{U}$ reaches Y_1 (i.e. $Y_1 \in Res(\mathcal{U})$), and adds W_1 to it. As shown on the figure, after the update occurs, W_1 becomes part of the shaded $\mathcal{VR}$ tree. This is an internal addition since the reached node, Y_1, belongs to the current $\mathcal{VR}$ (i.e. the view result before the update occurs).

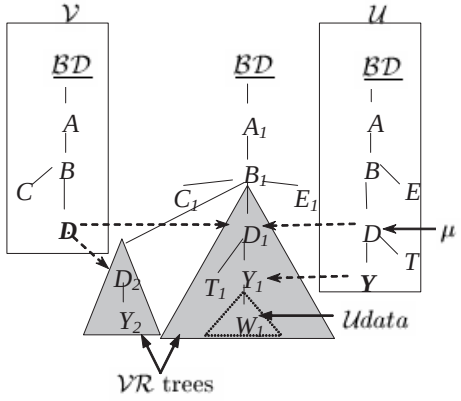


Figure 3: Internal addition via intermediate result

The IRT. As mentioned above, to develop the IRT logic for some type of effect, we need to derive a necessary condition that must be satisfied in order for that type of effect to happen. Examining the example shown above reveals that the internal addition happened because there is a

node, D_1, which belongs to both the final view result $Res(\mathcal{V})$ and the intermediate result of $\mathcal{U}$ at its spinal node D, i.e. $Res_D(\mathcal{U})$. Thus, $\mathcal{U}$ was able to reach Y_1 through D_1. In other words, a necessary condition for this scenario to take place is that the intersection of $Res(\mathcal{V})$ and $Res_D(\mathcal{U})$ (i.e. $Res(Exp_D(\mathcal{U}))$) is not empty. This condition is formally stated as: $\exists$ *node* $\mu \in Spine(\mathcal{U})$*s.t.*

$$Res(\mathcal{V}) \cap Res(Exp_\mu(\mathcal{U})) \textit{ is not empty} \quad (1)$$

Before discussing how to test Condition 1, we need to explore any other scenarios of internal additions.

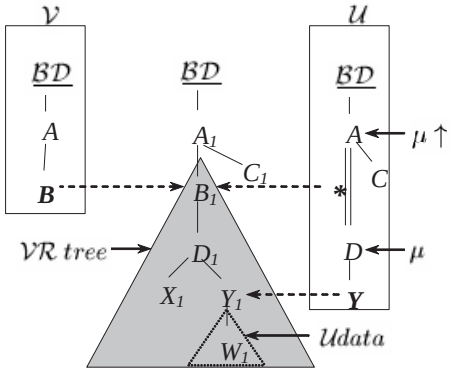


Figure 4: Internal addition via a // edge

Consider the case in Figure 4; in this example, the final result set of the view is $Res(\mathcal{V}) = \{B_1\}$. Obviously, none of the intermediate results of $\mathcal{U}$ includes B_1. Yet, $\mathcal{U}$ reaches Y_1 in $\mathcal{VR}$. In this case, $\mathcal{U}$ used a spinal descendent edge // to bypass the root B_1 and reach inside $\mathcal{VR}$.

The existence of a spinal descendent edge is necessary for this scenario. We formulate this necessary condition by referring to the spinal node μ which is the destination of the descendent edge, in this case $\mu = D$. Thus, we get the following necessary condition for this scenario: $\exists$ *node* $\mu \in Spine(\mathcal{U})$ s.t.

$$e(\mu) = // \ \wedge$$
$$Res(\mathcal{V}) \cap Res(Exp_{\mu\uparrow}(\mathcal{U})//*) \textit{ is not empty} \quad (2)$$

The notation used between the parentheses in the right operand of the intersection operator is used for concatenating expression fragments. In this case, it means: the Exp expression, followed by "//", followed by "*". We use this notation to form expressions using expression fragments. In the example of Figure 4, the right operand of the intersection operator of Condition 2 is $Res(Exp_A(\mathcal{U})//*)$ which is $Res(\mathcal{BD}/A[C][//D/Y]//*)$ which intersects $Res(\mathcal{V})$ at B_1.

There is no way for $\mathcal{U}$ to reach inside $\mathcal{VR}$ without having some spinal node μ satisfying at least one of Conditions 1 or 2. Thus, the disjunction of these two conditions forms a necessary condition for an internal addition to happen. If the IRT can test this disjunctive condition and determine that it is not satisfied, then it can safely conclude that the base addition is irrelevant to the view w.r.t. internal additions.

However, under the partial information model, the IRT does not have enough information to test neither Condition 1 nor Condition 2. The reason is that no intermediate results and no node ids are available. Thus, the actual data sets for which the intersection test should be conducted are not available. Only the expression that generates each set is available. Therefore, the intersection test must be conducted intensionally rather than extensionally [12]. In other words, the test must be conducted independently of the actual $\mathcal{BD}$ instance. We use the following definition for the $\mathcal{BD}$-independent intersection test.

Definition 1. *For two XPath expressions $\mathcal{E}_1$ and $\mathcal{E}_2$, $Intersect(\mathcal{E}_1, \mathcal{E}_2)$ is TRUE iff there exists a hypothetical $\mathcal{BD}$ instance in which $Res(\mathcal{E}_1) \cap Res(\mathcal{E}_2)$ is not empty.*

Section 4 shows the implementation of the XPath intersection test according to Definition 1.

Using the notion of XPath intersection in Definition 1, Conditions 1 and 2 are respectively reduced to:

$$Intersect(\mathcal{V}, Exp_\mu(\mathcal{U})) \quad (3)$$

$$e(\mu) = // \ \wedge \ Intersect(\mathcal{V}, Exp_{\mu\uparrow}(\mathcal{U})//*) \quad (4)$$

Since Definition 1 is independent of any specific $\mathcal{BD}$ instance, the intersection of two XPath expressions according to this definition is necessary but not sufficient for the intersection of the two sets generated by the expressions from a specific $\mathcal{BD}$ instance. Thus, Conditions 3 and 4 are necessary but not sufficient for $\mathcal{U}$ to reach a node inside $\mathcal{VR}$. This is the main source of false negatives of the IRT [2].

Having these false negatives, we are faced with the question of whether it is possible to refine (i.e. tighten) Conditions 3 and 4 in order to reduce the false negatives of the IRT. Fortunately, the available partial information can be used for such a refinement. The key idea is to use the current materialized $\mathcal{VR}$. First, we refine Condition 3, then we refine Condition 4.

In the example of Figure 3, for the effect to take place, the expression $\mathcal{VR}/D[T]/Y$ (where $D[T]/Y$ is the subtree expression of $\mathcal{U}$ at node D) must reach some node in $\mathcal{VR}$ (in the example, it reaches Y_1). Without this condition, $\mathcal{U}$ as a whole expression can not reach any node in $\mathcal{VR}$. This condition is formally stated as: for a node μ satisfying Condition 3, it is also necessary to satisfy

$$Res(\mathcal{VR}/Sub_\mu(\mathcal{U})) \textit{ is not empty} \quad (5)$$

Similarly, we refine Condition 4. In the example of Figure 4, for the effect to take place, the expression $p = \mathcal{VR}/*//D/Y$(where D/Y is the subtree expression of $\mathcal{U}$ at node D) must reach some node in $\mathcal{VR}$ (in the example, it reaches Y_1). Note the hypothetical "*" label added on the descendant edge of $\mathcal{U}$ in the figure, and used in expression p to account for the bypassed root B_1. The condition that expression p reaches a node in $\mathcal{VR}$ is formally stated as: for a node μ satisfying Condition 4, it is also necessary to satisfy

$$Res(\mathcal{VR}/*//Sub_\mu(\mathcal{U})) \textit{ is not empty} \quad (6)$$

Based on this analysis, we state the following lemma:

Lemma 1. *A base addition is irrelevant to a view w.r.t. internal additions if the following holds: $\nexists \mu \in Spine(\mathcal{U})$ s.t.*

$$(Cond\ 3 \wedge Cond\ 5) \vee (Cond\ 4 \wedge Cond\ 6) \quad (7)$$

Note that detecting irrelevance using this lemma is sound but not complete since the conditions used are independent

[2]The other source is the insufficiency of Condition 2 per se for $\mathcal{U}$ to reach inside $\mathcal{VR}$.

of the $\mathcal{BD}$. Thus, the IRT may have false negatives as mentioned earlier. The conditions used in this lemma can not be further refined under the partial information assumption.

The SMT. If the IRT is negative, then the SMT is conducted to explore the possibility of writing a correct Self Maintenance Expression (SME) $\mathcal{S}$. For internal additions, $\mathcal{S}$ is correct iff it reaches exactly the nodes that $\mathcal{U}$ reaches in $\mathcal{VR}$. We say that $\mathcal{S}$ is *complete* if it reaches all nodes that $\mathcal{U}$ reaches in $\mathcal{VR}$, and we say that it is *safe* if it does not reach any node that $\mathcal{U}$ does not reach. Thus, a view is self maintainable under a given update (i.e. the SMT is positive) if it is possible to write a complete and safe SME $\mathcal{S}$.

To guarantee completeness, we make use of the fact that $\mathcal{U}$ can not reach any node in $\mathcal{VR}$ unless at least one of the disjuncts of Condition 7 is satisfied at some node $\mu \in Spine(\mathcal{U})$. For the example in Figure 3, the left disjunct of Condition 7 holds; all the nodes that $\mathcal{U}$ reaches in $\mathcal{VR}$ are reached by the expression $\mathcal{VR}/D/Y$; thus, this expression is complete for this example. In general, if the left disjunct of Condition 7 holds, then the SME defined as:

$$\mathcal{S} = \mathcal{VR}/Sub_{\mu}(\mathcal{U}) \tag{8}$$

reaches all the nodes that $\mathcal{U}$ reaches in $\mathcal{VR}$. Similarly, using the example of Figure 4, we deduce that if the right disjunct of Condition 7 holds, then the SME defined as:

$$\mathcal{S} = \mathcal{VR}/*//Sub_{\mu}(\mathcal{U}) \tag{9}$$

reaches all the nodes that $\mathcal{U}$ reaches in $\mathcal{VR}$. Thus, Expressions 8 and 9 are complete for the scenarios illustrated in Figures 3 and 4 respectively.

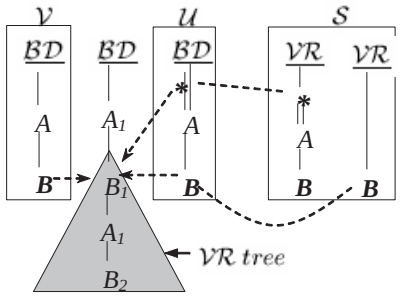


Figure 5: Multi-component SME

Note that there may be multiple nodes $\mu \in Spine(\mathcal{U})$ that satisfy Condition 7 through one, or both, of its disjuncts. For example, consider the view and update in Figure 5. In this example, $Spine(\mathcal{U})$ has node A satisfying the right disjunct, and node B satisfying the left disjunct. Thus, there are two possible scenarios by which $\mathcal{U}$ can reach nodes inside $\mathcal{VR}$; these scenarios are illustrated by the two mappings shown in the figure. One mapping maps the spinal node B of $\mathcal{U}$ to the tree root B_1, and the other maps a hypothetical "*" node on $Spine(\mathcal{U})$ to the same tree root. For the SME $\mathcal{S}$ to be complete in this case, it has to incorporate both scenarios; thus, it must have the two components shown in the figure in correspondence to the two mappings. In this example, $\mathcal{U}$ reaches both nodes B_1 and B_2 in $\mathcal{VR}$. For $\mathcal{S}$ to reach both these nodes, it must use both the components shown in the figure.

The discussion above shows that it is always possible to generate a - possibly multi-component - SME that is complete for internal additions. The question now is: is this SME always safe? Unfortunately, the answer is "No". For example, consider the case in Figure 3 and assume that node E_1 does not exist in $\mathcal{BD}$. In this case, $B_1 \notin Res_B(\mathcal{U})$, and thus, $D_1 \notin Res_D(\mathcal{U})$. Therefore, $\mathcal{U}$ does not reach any node in $\mathcal{VR}$. Using Lemma 1, the IRT is negative. Thus, the SMT is conducted and the SME $\mathcal{S} = \mathcal{VR}/D[T]/Y$ is generated based on Expression 8. This expression is unsafe because it would reach Y_1 which is not reached by $\mathcal{U}$.

The unsafe behavior in this example is due to the fact that there exists a node, D_1, such that $D_1 \in Res(\mathcal{V})$ while $D_1 \notin Res_D(\mathcal{U})$; and $\mathcal{S}$ could not discern this fact. This observation suggests that one way to guarantee safety in this example is to guarantee that $Res(\mathcal{V}) \subseteq Res_D(\mathcal{U})$. i.e., $Res(\mathcal{V}) \subseteq Res(Exp_D(\mathcal{U}))$. If this condition is true, then $\mathcal{S}$ will never unsafely reach a node in $\mathcal{VR}$ through a node in $Res(\mathcal{V})$, because all the nodes in $Res(\mathcal{V})$ are also in $Res_D(\mathcal{U})$, and thus are reached by $\mathcal{U}$.

We note that this condition is only sufficient but not necessary for safety. For example, D_2 in the example in Figure 3 does not satisfy it since $D_2 \in Res(\mathcal{V})$ while $D_2 \notin Res_D(\mathcal{U})$; and yet, $\mathcal{S}$ does not reach Y_2 through D_2.

We use this observation to refine the condition, and thus reduce the false negatives of the SMT. To do the refinement, we need to study the difference between the case of D_1/Y_1 on one hand, and the case of D_2/Y_2 on the other hand. In the former, Y_1 was unsafely reached through D_1 because $\mathcal{S}$ does not discern the reason for which Y_1 is not reached by $\mathcal{U}$. This reason is that $\mathcal{U}$ has a predicate constraint that node B must have a child E. The absence of E_1 violated this constraint; $\mathcal{S}$ can not discern this fact since (1) the predicate constraint $[E]$ is not part of $\mathcal{S}$, and (2) a node like E_1, which could have satisfied this constraint, is outside $\mathcal{VR}$, which is the only part of $\mathcal{BD}$ that is available to $\mathcal{S}$. In the case of D_2/Y_2, on the other hand, the absence of a child labeled T at D_2 is enough for $\mathcal{S}$ to discern that $\mathcal{U}$ does not reach Y_2. $\mathcal{S}$ could verify this constraint since (1) the predicate $[T]$ is part of $\mathcal{S}$, and (2) any node that could satisfy the constraint must be within $\mathcal{VR}$.

This demonstrates that all the constraints - i.e. predicates, label tests, and axis tests - of $\mathcal{U}$ that $\mathcal{S}$ can verify would not cause $\mathcal{S}$ to be unsafe. Since $\mathcal{S}$, in this case, is defined by Expression 8 as $\mathcal{VR}/Sub_D(\mathcal{U})$, the only constraints of $\mathcal{U}$ that $\mathcal{S}$ can not discern within $\mathcal{VR}$ are those that are not included in the expression $\mathcal{VR}/Sub_D(\mathcal{U})$, i.e. the constraints included in the expression $Prefix_D(\mathcal{U})$. Therefore, if $Res(Prefix_D(\mathcal{U}))$ is guaranteed to be a superset of $Res(\mathcal{V})$, then we guarantee that the SME $\mathcal{S}$ will never unsafely reach a node in $\mathcal{VR}$ through a node in $Res(\mathcal{V})$, because all the nodes in $Res(\mathcal{V})$ are also in $Res(Prefix_D(\mathcal{U}))$, and thus are either (1) reached by $\mathcal{U}$, or else (2) will not be reached by $\mathcal{S}$ because the constraints included in $\mathcal{S}$ will excluded these nodes. The general form of this safety condition is:

$$Res(\mathcal{V}) \subseteq Res(Prefix_{\mu}(\mathcal{U})) \tag{10}$$

Using similar logic, we derive a sufficient condition for safety for the other case of internal effects shown in Figure 4. In this case, the SME $\mathcal{S}$ is defined by Expression 9 as $\mathcal{VR}/*//Sub_{\mu}(\mathcal{U})$; and thus, the constraints that are not included in $\mathcal{S}$ are the constraints included in the expression

$Prefix'_{\mu\uparrow}(\mathcal{U})//*$. Hence, the general safety condition for this scenario is:

$$Res(\mathcal{V}) \subseteq Res(Prefix'_{\mu\uparrow}(\mathcal{U})//*) \quad (11)$$

As in the argument for the IRT, under the partial information model, no intermediate results, and no node ids are available. Thus, in Conditions 10 and 11, the actual data sets for which the containment test should be conducted are not available. Only the expression that generates each set is available. Thus, we need to conduct the containment test independently of the $\mathcal{BD}$. We define containment for XPath expressions as:

DEFINITION 2. *For two XPath expressions $\mathcal{E}_1$ and $\mathcal{E}_2$, $Contains(\mathcal{E}_1, \mathcal{E}_2)$ is TRUE iff $\forall\, \mathcal{BD}$, $Res(\mathcal{E}_2) \subseteq Res(\mathcal{E}_1)$.*

Section 4 discusses the implementation of the XPath containment test according to Definition 2.

Using this definition, and the analysis mentioned above, we state the following lemma:

LEMMA 2. *A view is self maintainable under a base addition w.r.t. internal additions if $\forall \mu \in Spine(\mathcal{U})$,*

$$(Cond\ 3 \wedge Cond\ 5) \Rightarrow Contains(Prefix_{\mu}(\mathcal{U}), \mathcal{V}),\ AND$$
$$(Cond\ 4 \wedge Cond\ 6) \Rightarrow Contains(Prefix'_{\mu\uparrow}(\mathcal{U})//*, \mathcal{V})$$

Since the XPath containment tested by Lemma 2 is $\mathcal{BD}$-independent, the SMT according to this lemma can result in false negatives. For example, the SMT is negative when conducted for the expressions $\mathcal{V}$ and $\mathcal{U}$ in Figure 3. For the $\mathcal{BD}$ shown in the figure; this is a false negative since the SME $\mathcal{S}$ as defined by Expression 8 reaches the set $\{Y_1\}$ which is exactly the set of nodes reached by $\mathcal{U}$.

3.1.2 Explicit Additions

Consider the example in Figure 6. In this example, $\mathcal{U}$ reaches B_1 and adds to it the tree rooted at C_1. This consequently adds the shaded tree as an entire new tree to $\mathcal{VR}$. This is an explicit addition since the added $\mathcal{VR}$ tree is fully included in $\mathcal{U}data$.

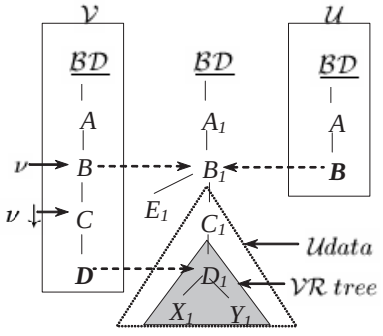


Figure 6: Explicit Addition Effect

The IRT. For an explicit addition to happen, as shown in this example, it is necessary that $\mathcal{V}$ reaches a node in the newly added tree $\mathcal{U}data$. Recall that for an internal addition to happen, as shown in Figures 3 and 4, it is necessary that $\mathcal{U}$ reaches a node in a $\mathcal{VR}$ tree. Comparing these two facts shows that there is a symmetry between internal additions and explicit additions by switching the roles of the $\mathcal{VR}$ *tree* and $\mathcal{U}data$, and the roles of $\mathcal{V}$ and $\mathcal{U}$. A logical analysis that is symmetric to the analysis for internal additions leads to symmetric necessary conditions for explicit additions. We refer to the node on $Spine(\mathcal{V})$ which is analogous to node μ on $Spine(\mathcal{U})$ as ν; in Figure 6, $\nu = B$.

Based on the symmetry, we derive the following conditions as analogous to Conditions 3, 4, 5, and 6, respectively:

$$Intersect(\mathcal{U}, Exp_{\nu}(\mathcal{V})) \quad (12)$$

$$e(\nu) = //\ \wedge\ Intersect(\mathcal{U}, Exp_{\nu\uparrow}(\mathcal{V})//*) \quad (13)$$

$$Res(\mathcal{U}data/Sub_{\nu\downarrow}(\mathcal{V}))\ is\ not\ empty \quad (14)$$

$$Res(\mathcal{U}data//Sub_{\nu}(\mathcal{V}))\ is\ not\ empty \quad (15)$$

Note that Conditions 14 and 15 are not exactly symmetric to Conditions 5 and 6. This is because internal and explicit additions are not exactly symmetric. In Figures 3 and 4 (internal additions), $Ret(\mathcal{V})$ is mapped to the root of the $\mathcal{VR}$ tree; while in Figure 6 (explicit additions), the parent of $Ret(\mathcal{U})$ is mapped to the root of $\mathcal{U}data$.

For the same reason, the node $Ret(\mathcal{V})$ is excluded from the set of spinal nodes for which the conjunction ($Cond\ 12 \wedge Cond\ 14$) is checked since $\mathcal{V}$ has to reach a node in $\mathcal{U}data$, and not the parent of $\mathcal{U}data$. The reason is that if, due to the addition, the parent of $\mathcal{U}data$, which is B_1 in the example, is added to $Res(\mathcal{V})$, then this is an implicit addition because some of the descendants of this parent, e.g. E_1, are not included in $\mathcal{U}data$. Based on this analysis, we state the following lemma:

LEMMA 3. *A base addition is irrelevant to a view w.r.t. explicit additions if the following two conditions hold:*

$$\nexists\, \nu \in (Spine(\mathcal{V}) - Ret(\mathcal{V}))\ s.t.\ (Cond\ 12 \wedge Cond\ 14) \quad (16)$$

$$\nexists\, \nu \in Spine(\mathcal{V})\ s.t.\ (Cond\ 13 \wedge Cond\ 15) \quad (17)$$

The SMT. Since the new tree added to $\mathcal{VR}$ is fully included in $\mathcal{U}data$, which is part of the update statement, it is possible in some cases to self maintain the view by extracting the added tree, or trees, from $\mathcal{U}data$. Self maintainability is possible w.r.t. explicit additions if it is possible to write an SME $\mathcal{S}$ which reaches exactly the nodes that $\mathcal{V}$ reaches in $\mathcal{U}data$. Using the symmetry mentioned above between internal and explicit additions, we state a sufficient condition for the SMT as in the following lemma:

LEMMA 4. *A view is self maintainable under a base addition w.r.t. explicit additions if the following two conditions hold:*

$$\forall \nu \in (Spine(\mathcal{V}) - Ret(\mathcal{V})),\ (Cond\ 16) \Rightarrow Contains(Prefix_{\nu}(\mathcal{V}), \mathcal{U})$$

$$\forall \nu \in Spine(\mathcal{V}),\ (Cond\ 17) \Rightarrow Contains(Prefix'_{\nu\uparrow}(\mathcal{V})//*, \mathcal{U})$$

3.1.3 Implicit Additions

Consider the example in Figure 7. The base addition adds the tree Y_1/Z_1 as a child to X_1; this in turn adds X_1 to $Res_X(\mathcal{V})$. Thus A_1 is added to $Res_A(\mathcal{V})$, and B_1 is added to $Res(\mathcal{V})$. This causes an implicit addition since it adds the shaded tree, which is not included in $\mathcal{U}data$, to $\mathcal{VR}$.

The IRT. As shown by this example, a necessary condition for an implicit addition is that the added tree $\mathcal{U}data$

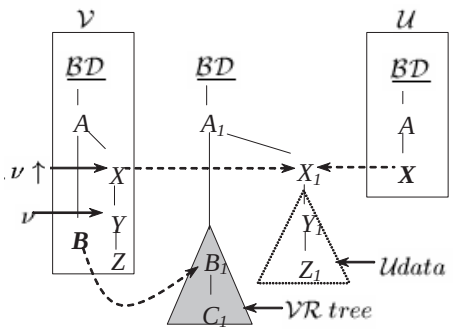


Figure 7: Implicit Addition Effect

includes a match of a non-spinal subtree of $\mathcal{V}$. In this example, $\mathcal{U}data$ is the tree Y_1/Z_1 which matches the non-spinal subtree of $\mathcal{V}$ at Y, i.e. $Sub_Y(\mathcal{V})$. This condition is formally stated as: $\exists$ non-spinal node ν in $\mathcal{V}$ s.t. $Res(\mathcal{U}data/Sub_\nu(\mathcal{V}))$ is not empty. In the case where $e(\nu)$ is "//" instead of "/", this condition becomes: $Res(\mathcal{U}data//Sub_\nu(\mathcal{V}))$ is not empty. Thus, in either case, a necessary condition for an implicit additions is: $\exists$ non-spinal node ν in $\mathcal{V}$ s.t.

$$Res(\mathcal{U}data.e(\nu).Sub_\nu(\mathcal{V})) \text{ is not empty} \quad (18)$$

In fact, we can refine this necessary condition by making it stricter to reduce the false negatives. The example in Figure 7 shows that it is necessary that $\mathcal{U}$ reaches a node which becomes in $Res_{\nu\uparrow}(\mathcal{V})$ after the update happens. In this example, $\mathcal{U}$ reaches X_1 which becomes in $Res_X(\mathcal{V})$ after the update. If this condition is not satisfied, then the addition can not lead to adding X_1 to $Res_X(\mathcal{V})$, and adding A_1 to $Res_A(\mathcal{V})$, and thus can not cause the implicit addition. Note that the node reached by $\mathcal{U}$ may be spinal or non-spinal. This condition is formally stated as: $Res(\mathcal{U}) \cap Res_{\nu\uparrow}(\mathcal{V})$ is not empty. In the partial information model, as mentioned before, we conduct a $\mathcal{BD}$-independent intersection test to check this condition. Thus, this condition is:

$$Intersect(\mathcal{U}, Exp_{\nu\uparrow}(\mathcal{V})) \quad (19)$$

This analysis leads to the following lemma:

LEMMA 5. *A base addition is irrelevant to a view w.r.t. implicit additions if:* $\nexists$ *non-spinal node* $\nu \in \mathcal{V}$ *s.t.* (*Cond* 18 $\wedge$ *Cond* 19).

The SMT. Unlike the case with explicit additions, an implicitly added $\mathcal{VR}$ tree is not available as part of $\mathcal{U}data$, thus it can not be extracted from the available partial information. Hence, self maintenance is not possible for implicit additions, i.e. if the IRT is negative w.r.t. implicit additions, then the SMT is also negative w.r.t. implicit additions.

3.2 Base Deletions

Like base additions, we classify the effects of a base deletion into two types: *internal deletions* and *external deletions*. An internal deletion happens when $\mathcal{U}$ reaches and deletes a node that is in the current $\mathcal{VR}$. An external deletion, on the other hand, happens when an entire tree is deleted from $\mathcal{VR}$. The following subsections illustrate these two types of deletion effects and develop the IRT and SMT logic for each of them.

Note that both the characterizations of internal and external deletions apply to the special case in which $\mathcal{U}$ reaches and deletes the root of some $\mathcal{VR}$ tree. For the clarity of the derived general conditions, we keep this border-line case covered by both types of effects instead of removing it from one and keeping it in the other. This does not cause any duplication problems.

It is possible to show by examples that a single base deletion can result in both internal and external deletion effects. Thus, for a given base deletion, the IRT and SMT have to be conducted for both types of effects.

3.2.1 Internal Deletions

For an example of an internal deletion, consider Figures 3 and 4, and assume the base update is a deletion. In these examples, $\mathcal{U}$ reaches and deletes Y_1 which is in the current $\mathcal{VR}$. The IRT and SMT conditions for internal deletions are the same as those for internal additions. In both cases, $\mathcal{U}$ reaches a node inside the current $\mathcal{VR}$ and affects it.

3.2.2 External Deletions

For an example of an external deletion, consider Figure 8. In this figure, $\mathcal{U}$ reaches B_1 and deletes it. Thus, the shaded tree is deleted from $\mathcal{VR}$. This is an external deletion since B_1 is not in the current $\mathcal{VR}$.

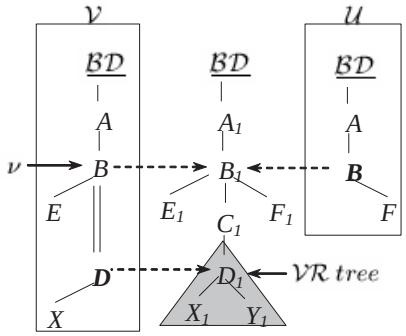


Figure 8: External Deletion Effect

This example shows an external deletion happening by deleting a node, B_1, which belongs to some spinal intermediate result of $\mathcal{V}$, namely, $Res_B(\mathcal{V})$. External deletions can also happen by deleting a node that belongs to a non-spinal intermediate result. An example of such a case is an update that reaches and deletes E_1 from $Res_E(\mathcal{V})$. Based on the type of the deleted node (spinal or non-spinal), we can classify external deletions into explicit and implicit, as we do for external additions; however, we do not make such distinction because the conditions for both cases are exactly the same.

The IRT. The examples discussed above show that external deletions could happen when $\mathcal{U}$ reaches a node in some intermediate result of $\mathcal{V}$. Formally speaking, $\exists\nu \in \mathcal{V}$ s.t. Condition 12 holds.

It is also possible that an external deletion happens without satisfying Condition 12. For example, in Figure 8, assume that $\mathcal{U} = \mathcal{BD}/A/B/C$. In this case, $\mathcal{U}$ reaches C_1 and deletes it. This causes an external deletion to the shaded tree even though C_1 does not belong to any intermediate result of $\mathcal{V}$. In this case, the update indirectly deleted D_1, which belongs to $Res_D(\mathcal{V})$, through deleting its ancestor C_1. Obviously, the "//" edge in $\mathcal{V}$ between B and D is what allowed the external deletion even though $\mathcal{U}$ does not reach

any node in any intermediate result of $\mathcal{V}$. This condition is formally stated as: $\exists \nu \in \mathcal{V}$ s.t. Condition 13 holds.

Thus, a necessary condition for an external deletion to happen is that some node is deleted by the base deletion from some intermediate result $Res_\nu(\mathcal{V})$, either directly (Cond 12) or indirectly (Cond 13). Based on this analysis, we state the following lemma:

LEMMA 6. *A base deletion is irrelevant to a view w.r.t. external deletions if $\mathcal{VR}$ is not empty (trivial case), and $\nexists \nu \in \mathcal{V}$ s.t. ($Cond$ 12 $\vee$ $Cond$ 13)*

Recall that, in the case of base additions, Conditions 12 and 13 are respectively refined by, i.e. conjuncted with, Conditions 14 and 15. These refining conditions have no counterparts in the case of base deletions because they use $\mathcal{U}data$, which is available only with base additions. Thus, we expect that processing base deletions would result in more false negatives than processing base additions. Our experimental results in Section 5 confirmed this expectation.

The SMT. If the IRT of external deletions is negative, then this implies that one or more trees may be deleted from the current $\mathcal{VR}$. Self maintainability for external deletions is possible if it is possible to write an SME $\mathcal{S}$ which exactly identifies the roots of the $\mathcal{VR}$ trees that must be deleted from the current $\mathcal{VR}$ collection. This is generally not possible because the materialized $\mathcal{VR}$ does not include node ids or intermediate results. For example, in Figure 8, if F_1 did not exist, then $\mathcal{U}$ would not reach B_1, and thus the shaded tree must not be deleted from $\mathcal{VR}$. Since, there is no information available to the IRT or SMT on whether there exists an F child of B_1 or not, there is no way to determine whether the shaded tree must be deleted or not. Thus, self maintenance is generally not possible for external deletions.

However, there is a special case in which it is possible to determine, for each tree in the current $\mathcal{VR}$, whether it must be deleted or not. If node ν that caused the IRT to be negative, based on Condition 12 or Condition 13, is one of the descendants of $Ret(\mathcal{V})$, i.e. if $\nu \in Sub_{Ret(\mathcal{V})}(\mathcal{V})$, then the information in each $\mathcal{VR}$ tree is sufficient to determine whether the tree must be deleted or not. For example, in Figure 8, assume that $\mathcal{U} = \mathcal{BD}/A//X$. In this case, $\mathcal{U}$ reaches X_1 and deletes it. This deletes X_1 from $Res_X(\mathcal{V})$, and consequently, deletes D_1 from $Res_D(\mathcal{V})$, i.e. from $Res(\mathcal{V})$. This causes an external deletion since the shaded tree would be deleted from $\mathcal{VR}$. In this case, it is possible to determine, using the available information, that the tree must be deleted from $\mathcal{VR}$. The reason is that the deletion of X_1 is reflected as an internal deletion in the current $\mathcal{VR}$. Thus, by processing internal deletions first, it is always possible to self maintain $\mathcal{VR}$ given that $\nu \in Sub_{Ret(\mathcal{V})}(\mathcal{V})$.

There is another case in which external deletions can be trivially self maintained. If it is guaranteed that the base deletion deletes all the nodes in some intermediate result of $\mathcal{V}$, either directly (Cond 12), or indirectly (Cond 13), then all the trees in $\mathcal{VR}$ would be deleted. Thus, the SMT is trivially positive. Based on this analysis, we state the following lemma:

LEMMA 7. *A view is self maintainable under a base deletion w.r.t. external deletions if $\forall \nu \in (\mathcal{V} - Sub_{Ret(\mathcal{V})}(\mathcal{V}))$,*

$$(Cond\ 12) \Rightarrow Contains(\mathcal{U}, Exp_\nu(\mathcal{V})),\ AND$$
$$(Cond\ 13) \Rightarrow Contains(\mathcal{U}, Exp_{\nu\uparrow}(\mathcal{V})//*)$$

3.3 Base Modifications

In this section, we use the following definition for base modifications.

DEFINITION 3. *For a base modification update with expression $\mathcal{U}$, $\mathcal{U}^{new}$ is the expression $\mathcal{U}$ after modifying the label of its return node to $\mathcal{U}data$, i.e. to the new label imposed by the update operation.*

For example, if $\mathcal{U} = A/B[E]/C$, and $\mathcal{U}data = K$, then $\mathcal{U}^{new} = A/B[E]/K$.

The effects of a base modification can be internal (modifying the label of a node in $\mathcal{VR}$), or external. External effects of a base modification can be either additions or deletions of entire $\mathcal{VR}$ trees. The following subsections develop the IRT and SMT logic for the three types of effects that a base modification can cause: internal modifications, external additions, and external deletions. Like the case with additions and deletions, for a given base modification, the IRT and SMT have to be conducted for all the three types of effects.

3.3.1 Internal Modifications

For an example of an internal modification, consider Figures 3 and 4, and assume that the base update is a modification with $\mathcal{U}data = K$. In this case, $\mathcal{U}$ reaches Y_1 and changes its label to K, this is an internal modification since the modified node is in the current $\mathcal{VR}$. The IRT and SMT conditions for internal modifications are the same as those for internal additions and deletions.

3.3.2 External Additions

For an example of an external addition, consider Figure 6, and assume that $\mathcal{U} = A/K$, and that B_1 is originally labeled K, and changed by the update to B. This effect adds B_1 to $Res_B(\mathcal{V})$, and consequently adds the shaded tree to $\mathcal{VR}$.

The IRT. A necessary condition for this type of effect to happen is that $Res(\mathcal{U}^{new})$ intersects with $Res_\nu(\mathcal{V})$ for some $\nu \in \mathcal{V}$. The following lemma uses this condition to detect irrelevant modifications w.r.t. external additions.

LEMMA 8. *A base modification is irrelevant to a view w.r.t. external additions if: $\nexists \nu \in \mathcal{V}$ s.t. $Intersect(Exp_\nu(\mathcal{V}), \mathcal{U}^{new})$*

The SMT. Self maintainability is not possible for this type of effect because the tree that should be added to $\mathcal{VR}$ is not available as part of the update statement.

3.3.3 External Deletions

For an example of an external deletion, consider Figure 6, and assume that the update changes the label of B_1 to K. This deletes B_1 from $Res_B(\mathcal{V})$, and consequently deletes the shaded tree from $\mathcal{VR}$.

The IRT. This case is similar to the case of external deletions explained in Section 3.2.2. The difference is that the modified node has to be in some intermediate result and not just an ancestor of some node in an intermediate result. It is also necessary that the new label imposed by the update does not match the label test of this intermediate result. Thus, we state the following lemma:

LEMMA 9. *A base modification is irrelevant to a view w.r.t. external deletions if: $\nexists \nu \in \mathcal{V}$ s.t.*

$$(Cond\ 12) \wedge (\mathcal{U}data\ does\ not\ match\ \nu) \qquad (20)$$

The SMT. By the similarity to the external deletion effects as mentioned above:

LEMMA 10. *A view is self maintainable under a base modification w.r.t. external deletions if:* $\forall \nu \in (\mathcal{V} - Sub_{Ret(\mathcal{V})}(\mathcal{V}))$, $Cond\ 20 \Rightarrow Contains(\mathcal{U}, Exp_\nu(\mathcal{V}))$

4. IMPLEMENTATION

This section discusses the implementation of the XPath intersection test (for the IRT) and the XPath containment test (for the SMT).

XPath Intersection. As shown in Section 3, the IRT involves checking that two expressions in $XP^{\{[],*,//\}}$ do not intersect according to Definition 1. This requires verifying that there is no hypothetical $\mathcal{BD}$ instance in which the final result sets of the two expressions intersect. We used an approach that is similar to [15]. First, we reduce the problem from the domain of $XP^{\{[],*,//\}}$ to the simpler domain of $XP^{\{*,//\}}$; then we solve the problem in the simpler domain.

LEMMA 11. *For two XPath expressions* $\mathcal{E}_1$ *and* $\mathcal{E}_2$, $Intersect(\mathcal{E}_1, \mathcal{E}_2) \Rightarrow Intersect(Spine(\mathcal{E}_1), Spine(\mathcal{E}_2))$.
Proof. $Intersect(\mathcal{E}_1, \mathcal{E}_2) \Rightarrow \exists \mathcal{BD}$ *instance s.t.* $Res(\mathcal{E}_1) \cap Res(\mathcal{E}_2)$ *is not empty. Let* $m \in Res(\mathcal{E}_1) \cap Res(\mathcal{E}_2)$. *For every expression* $\mathcal{E}$, $Res(\mathcal{E}) \subseteq Res(Spine(\mathcal{E}))$ *(because* $Spine(\mathcal{E})$ *is just a relaxation of all the branching predicates of* $\mathcal{E}$*). Therefore,* $m \in Res(Spine(\mathcal{E}_1))$ *and* $m \in Res(Spine(\mathcal{E}_2))$. *Thus,* $Intersect(Spine(\mathcal{E}_1), Spine(\mathcal{E}_2))$. □

Hence, to verify that two expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ do not intersect, we extract the linear expressions $Spine(\mathcal{E}_1)$ and $Spine(\mathcal{E}_2)$, and verify that there is no hypothetical linear string (over the infinite label alphabet Σ) that matches both of these expressions. For this purpose, we note that any expression in $XP^{\{*,//\}}$ can be represented by a finite state automaton. For example, Figure 9 shows two expressions and their respective representations.

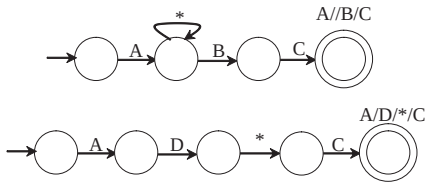


Figure 9: Automata representation of expressions

Our problem now becomes: given two automata as in Figure 9, is there any string that is accepted by both automata? For example, the two expressions in Figure 9 intersect because both automata accept the string $A/D/B/C$. To check if there exists such a common string, we simply extended the classical algorithm of automata intersection (cross product of states) by adding the logical rule that the symbol "*" matches any symbol, and we used the classical emptiness test (search for a path between initial and final states). Thus, we verify the emptiness of the intersection in $O(n*m)$, where n and m are the sizes of the two automata. Some straightforward optimizations have been applied to this basic implementation.

Obviously, if this automata test determines that the linear expressions $Spine(\mathcal{E}_1)$ and $Spine(\mathcal{E}_2)$ do not intersect, then we are sure that the expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ do not intersect either. Thus, this test maintains the IRT correctness because it can not result in any false positives. One may wonder, however, whether this test could result in any false negatives, which is acceptable but undesirable as explained before. A false negative may result if it is possible that $Spine(\mathcal{E}_1)$ and $Spine(\mathcal{E}_2)$ intersect while $\mathcal{E}_1$ and $\mathcal{E}_2$ do not. Fortunately, this can not happen; the following lemma states this fact [3].

LEMMA 12. *For two XPath expressions* $\mathcal{E}_1$ *and* $\mathcal{E}_2$, $Intersect(Spine(\mathcal{E}_1), Spine(\mathcal{E}_2)) \Rightarrow Intersect(\mathcal{E}_1, \mathcal{E}_2)$.
Proof. $Intersect(Spine(\mathcal{E}_1), Spine(\mathcal{E}_2)) \Rightarrow \exists \mathcal{BD}$ *instance s.t.* $Res(Spine(\mathcal{E}_1)) \cap Res(Spine(\mathcal{E}_2))$ *is not empty. Let* $m \in Res(Spine(\mathcal{E}_1)) \cap Res(Spine(\mathcal{E}_2))$. *Construct a new instance* $\mathcal{BD}'$ *from* $\mathcal{BD}$ *as follows: (1) add branches to the ancestors of* m *such that all the predicates (non-spinal nodes) of* $\mathcal{E}_1$ *are satisfied. (2) add branches to the ancestors of* m *such that all the predicates (non-spinal nodes) of* $\mathcal{E}_2$ *are satisfied. Since the predicates in* $XP^{\{[],*,//\}}$ *are not mutually-exclusive, it is always possible to realize these two steps together. In* $\mathcal{BD}'$, $m \in Res(\mathcal{E}_1) \cap Res(\mathcal{E}_2)$. *Thus,* $Intersect(\mathcal{E}_1, \mathcal{E}_2)$. □

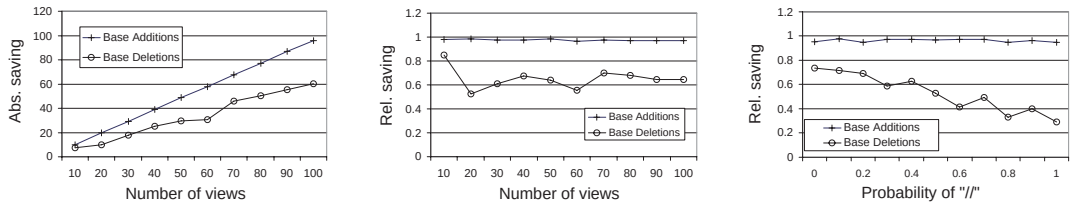
The proof of this lemma depends on the fact that the branching predicates can not be mutually exclusive, and thus we can satisfy any combination of predicates together at any node. Although this is true for the abstract data and query model of $XP^{\{[],*,//\}}$, it is not true in practice: the actual specification of the XML standard requires that the attributes of any element node have unique names. Thus, the two predicates [@*price* < 100] and [@*price* > 150] are actually mutually exclusive. Such an implicit uniqueness constraint is not captured in the abstract data and query model. Fortunately, this issue can be fixed by a simple encoding of the XPath expressions: we include any attribute predicates of a node as part of the node label, and we define the logic that compares two labels such that it takes into account the attribute predicates rather than only the equality of the element names. For example, the labels *Book*[@*price* < 100] and *Book*[@*price* > 150] are not compatible; while *Book*[@*price* < 100] and *Book*[@*price* < 150] are compatible.

XPath Containment. To check the containment between two expressions in $XP^{\{[],*,//\}}$, we implemented a practical algorithm presented in [21]. This algorithm is based on finding a homomorphism between two tree patterns, and it runs in $O(n*m)$, where n and m are the sizes of the two tree patterns. The algorithm is sound but incomplete; as mentioned before, soundness is sufficient for the correctness of the SMT. In this work, the algorithm is given for boolean (i.e. arity-0) tree patterns; we used a technique, described in the same work, to apply the algorithm to expressions of arity-1.

Since we include the attribute predicates of a node with the node label as explained above, we define the logic that maps between the nodes of the tree patterns in the homomorphism algorithm such that it takes into account the predicate containment relationships. For example, the label *Book*[@*price* < 100] subsumes the label *Book*[@*price* < 50], and thus it can be mapped to it in the homomorphism.

Note that even when an attribute predicate of a node is included as part of the node label as described above, we still need to create a separate node in the tree pattern to represent the attribute predicate. This is needed to enable

[3]This holds for the schemaless model that we are using.



(a) Abs. saving vs. Num of Views (b) Rel. saving vs. Num of Views (c) Rel. Saving vs. Prob. of "//"

Figure 10: The approach benefit in avoiding view recomputation (savings)

the homomorphism algorithm to discover the containment in some cases. For example, consider the two expressions $A[//@a]/B$ and $A/B[@a]$, the fact that the former expression contains the latter one can not be discovered by the homomorphism algorithm unless the attribute predicates are represented by separate nodes in the tree patterns.

5. EXPERIMENTS

We conducted experiments to evaluate the effectiveness of the proposed approach in avoiding unnecessary view recomputation. We used the XMARK benchmark [27] to generate synthetic $\mathcal{BD}$ instances. We also used an Xpath expression generator [9] to generate view definitions and update statements over the base XML document. This XPath expression generator takes as input some numeric parameters that control some properties of the generated expressions, such as the number of steps, the number of predicates, and the probabilities of using the symbols "*" and "//".

The performance results for base modifications are similar to those for base deletions, while they are both different from base additions. Thus, we show and compare the results for base additions and deletions.

We used two metrics to evaluate the benefits of the approach: *Absolute saving*, and *Relative saving*. The former is the number of views for which recomputation is avoided because either the IRT or the SMT was positive. The latter is defined as: *(Absolute saving/(Absolute saving + false negatives))*; this metric measures how close the amount of saving is to the optimal possible saving. The optimal possible saving is the actual number of views that were not affected by a base update, which is equal to *(Absolute saving + false negatives)*. Thus, this metric shows the impact of the false negatives on the savings achieved by the approach. To suppress the effect of outliers, for each experimental setup, we generate 10 base updates and take the average.

We conducted two main experiments. In the first experiment, we fixed the parameters used for the XPath generator and varied the number of views in the system. Figure 10(a), shows that the *Absolute saving* scales well with the number of views in the system. Both Figures 10(a) and 10(b) show that the savings for base additions are larger than the savings for base deletions. This is consistent with the expectation made in Section 3.2.2 due to the unavailability of $\mathcal{U}data$ in the update statement of base deletions. To further confirm this, we traced back the conditions which result in the majority of the false negatives for base deletions: it turned out that the vast majority of these false negatives result from the IRT of external deletions, which would have used $\mathcal{U}data$ if it were available as in the case of base additions.

In the second experiment, we fixed the number of views and the parameters used to generate them; and we varied one of the parameters used for generating the update expressions. We conducted several experiments of this type by changing the parameter to be varied. We have observed that, for most of the parameters, varying the parameter value resulted in no change in the savings performance. However, this was not the case for the parameters that control the probabilities of using the symbols "*" and "//" in the update expression. Figure 10(c) illustrates the *Relative saving* performance results of an experiment in which we varied the probability of an edge in $\mathcal{U}$ being "//" from 0.0 (no // edges) to 1.0 (all the edges are //).

This figure shows that increasing the probability of the symbol "//" reduces the *Relative saving* performance in the case of base deletions. However, the performance is almost not affected in the case of base additions. The *Absolute saving* performance also follows the same trend. A similar performance trend was observed with increasing the probability of the symbol "*" in the update statement.

This reduction in savings is due to the fact that the symbols "//" and "*" make the update statement less specific as mentioned in Section 2.2, and thus reduce the amount of information that is known about the actual effect of the update operation on $\mathcal{BD}$. The IRT and the SMT for base additions, unlike those for base deletions, are able to make up for the missing information because $\mathcal{U}data$ is available as part of the update statement of base additions.

6. CONCLUSION AND DISCUSSION

We have presented a practical solution for maintaining materialized XPath views in environments where the view maintenance system and the base data system are loosely-coupled. Our approach reduces the frequency of view recomputation by conducting an update Irrelevance Test (IRT) and a view Self Maintainability Test (SMT). We have implemented the approach using XPath intersection and containment checking techniques. The experimental results show that the proposed approach is very effective in avoiding unnecessary view recomputation, especially for base addition updates.

We have assumed that the base update expressions are available to the view maintenance system; this is obviously a minimum requirement for any view maintenance solution. In reality, however, true loose coupling implies that the base data server does not even know about the "client" systems that are maintaining views computed from its base data. So, how would it report the update expressions to these unknown clients? This can be done by allowing clients to register with the server, or by providing a service (by the

server) that returns the update expressions that took place since a given time-stamp.

The query language which is supported in this paper is $XP^{\{[],*,//\}}$. Although practical for many purposes, this language lacks some desirable XML querying features; most importantly, it lacks the power of binding variables. A query language with this power, such as XPath 2.0/XQuery 1.0 [1], can specify queries of arbitrary arity, and can include general value-based join conditions in addition to the structure-based join conditions. We plan to extend the results of this paper to allow such powerful query languages.

In this paper we have assumed a non-ordered tree model to represent XML documents. While this model is valid for most data-centric applications, it is not generally sufficient for document-centric applications. Considering the document order, and order-based predicates in the queries, raise some non-trivial issues in the problem of maintaining XPath views. Under the tight-coupling model, the problem can be solved by using node ids that capture the document order [10, 11]. Finding solutions (or partial solutions) under the loose-coupling model requires further investigation.

In Section 4, we have exploited the uniqueness constraint on attribute names to avoid some false negatives of the IRT. This idea can be generalized to exploiting schema information, if available, to reduce the false negatives of the IRT and/or the SMT. This research direction can leverage the works that have used schema information in the XPath intersection [7], and containment [28, 22] tests.

7. REFERENCES

[1] http://www.w3c.org/.

[2] Serge Abiteboul, Jason McHugh, Michael Rys, Vasilis Vassalos, and Janet L. Wiener. Incremental maintenance for materialized views over semistructured data. In *VLDB*, pages 38–49, 1998.

[3] Gustavo Alonso and Fabio Casati. Web services and service-oriented architectures. In *ICDE*, page 1147, 2005.

[4] Andrey Balmin, Fatma Ozcan, Kevin S. Beyer, Roberta Cochrane, and Hamid Pirahesh. A framework for using materialized xpath views in xml query processing. In *VLDB*, 2004.

[5] José A. Blakeley, Neil Coburn, and Per-Ake Larson. Updating derived relations: detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst.*, 14(3):369–400, 1989.

[6] José A. Blakeley, Per-Ake Larson, and Frank Wm. Tompa. Efficiently updating materialized views. In *SIGMOD Conference*, pages 61–71, 1986.

[7] Stefan Böttcher. Testing intersection of xpath expressions under dtds. In *IDEAS*, pages 401–406, 2004.

[8] Li Chen, Elke A. Rundensteiner, and Song Wang. Xcache: a semantic caching system for xml queries. In *SIGMOD Conference*, page 618, 2002.

[9] Yanlei Diao and Michael J. Franklin. Query processing for high-volume xml message brokering. In *VLDB*, pages 261–272, 2003.

[10] Katica Dimitrova, Maged El-Sayed, and Elke A. Rundensteiner. Order-sensitive view maintenance of materialized xquery views. In *ER*, pages 144–157, 2003.

[11] Maged El-Sayed, Elke A. Rundensteiner, and Murali Mani. Incremental maintenance of materialized xquery views. In *ICDE*, page 129, 2006.

[12] Ashish Gupta and José A. Blakeley. Using partial information to update materialized views. *Inf. Syst.*, 20(8):641–662, 1995.

[13] Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems and techniques and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[14] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[15] Beda Christoph Hammerschmidt, Martin Kempa, and Volker Linnemann. On the intersection of xpath expressions. In *IDEAS*, pages 49–57, 2005.

[16] Cheng Hua, Ji Gao, Yi Chen, and Jian Su. Self-maintainability of deletions of materialized views over xml data. In *International Conference on Machine Learning and Cybernetics*, volume 3, pages 1883 – 1888, 2003.

[17] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *VLDB*, pages 171–181. Morgan Kaufmann, 1993.

[18] Hartmut Liefke and Susan B. Davidson. View maintenance for hierarchical semistructured data. In *DaWaK*, pages 114–125, 2000.

[19] Bhushan Mandhani and Dan Suciu. Query caching and view selection for xml databases. In *VLDB*, pages 469–480, 2005.

[20] Hidetaka Matsumura and Keishi Tajima. Incremental evaluation of a monotone xpath fragment. In *CIKM*, 2005.

[21] Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of xpath. *J. ACM*, 51(1):2–45, 2004.

[22] Frank Neven and Thomas Schwentick. Xpath containment in the presence of disjunction, dtds, and variables. In *ICDT*, pages 315–329, 2003.

[23] Makoto Onizuka, Fong Yee Chan, Ryusuke Michigami, and Takashi Honishi. Incremental maintenance for materialized xpath/xslt views. In *WWW*, pages 671–681, 2005.

[24] Nicola Onose, Alin Deutsch, Yannis Papakonstantinou, and Emiran Curtmola. Rewriting nested xml queries using nested views. In *SIGMOD*, 2006.

[25] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semistructured data. In *SIGMOD Conference*, pages 455–466, 1999.

[26] Arsany Sawires, Junichi Tatemura, Oliver Po, Divyakant Agrawal, and K. Selçuk Candan. Incremental maintenance of path expression views. In *SIGMOD Conference*, 2005.

[27] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. Xmark: A benchmark for xml data management. In *VLDB*, pages 974–985, 2002.

[28] Peter T. Wood. Containment for xpath fragments under dtd constraints. In *ICDT*, pages 300–314, 2003.

[29] Wanhong Xu and Z. Meral Özsoyoglu. Rewriting xpath queries using materialized views. In *VLDB*, pages 121–132, 2005.

[30] Yue Zhuge and Hector Garcia-Molina. Graph structured views and their incremental maintenance. In *ICDE*, pages 116–125, 1998.

[31] Yue Zhuge and Hector Garcia-Molina. Self-maintainability of graph structured views. Technical report, Computer Science Department - Stanford University, 1998.