Integration and Efficient Lookup of Compressed XML Accessibility Maps

Fianny Mingfei Jiang, Ada Wai-Chee Fu

April 26, 2005

Abstract

XML is emerging as a useful platform-independent data representation language. As more and more XML data is shared across data sources, it becomes important to consider the issue of XML access control. One promising approach to store the accessibility information is based on the CAM (Compressed Accessibility Map). We make two advancements in this direction. 1)Previous work suggests that for each user group and each operation type, a different CAM is built. We observe that the performance and storage requirements can be further improved by combining multiple CAMs into an ICAM (Integrated CAM). We explore this possibility and propose an integration mechanism. 2)If the change in structure of the XML data is not frequent, we suggest an efficient lookup method, which can be applied to CAMs or ICAMs, with a much lower time complexity compared to the previous approach. We show by experiments the effectiveness of our approach.

Contents

1 Introduction

With more and more data represented in XML, the issue of controlled access for XML data has become extremely important. Encryption and digital signature methods, mainly used in electronic transactions, do protect the XML data in some aspects [4]. However, the design of a sophisticated access control mechanism for XML data still remains an open issue [15]. The models proposed by [3, 7] permit access control at different granularity levels and provide both positive and negative authorizations to better handle exceptions.

The study of access control for relational and object-oriented database models has a long history [5, 21, 10, 13, 14, 23, 19]. The access control for XML data is more complicated due to its semistructured feature. [8, 3, 15, 18] are important work concerning the definition of various access control models for XML data. Fundulaki and Marx[12] survey some of these models, and proposes a simple and unambiguous language to specify the access authorizations for XML data with XPath. While these efforts focus on constructing a semantically complete model to specify authorizations but discuss little about the efficiency of space and evaluation time of a query in the presence of a set of authorizations. Fan et al.[11] is one possible complementary work of these efforts. It inherits the access control mechanism from relational database systems. For each user or group of users, a virtual security view is generated, which is embedded into the query before evaluating the query against the document. The method in [11] requires that the XML document conform to a DTD, and every query be rewritten before being evaluated. Yet, another possible solution is to evaluate the authorizations in advance and generate an accessibility map for each user or a group of users, then to look up this map to discover the accessible nodes for an incoming query. But, since the accessibility of every node is explicitly defined, accessibility lookup involves scanning the whole map, which can be highly inefficient in both space and time.

This paper studies the problem of compressing the accessibility map while still achieving acceptable lookup time. We assume that the authorizations have been clearly defined by one of the models, such as [12], and we evaluate them in advance to generate an accessibility map for each user group. Then, we try to compress these maps. The idea of the compressed accessibility map (CAM) is proposed by Ting Yu *et al*. [24]. Efficient lookup on the compressed map is studied in [17]. However, in [24], one CAM is constructed for each possible operation type and for each user group. We notice that if we merge the CAMs for different types of operations into one integrated CAM (ICAM), we can further reduce the space needed for the accessibility maps. In addition, with the integrated CAM, if a user needs to look up the accessibilities for multiple types of operations for a data object, he/she needs to refer to only one ICAM, instead of multiple CAMs. We also study the lookup mechanism and propose a new method whose time complexity is much lower than that of the previous approach. The new method can also be applied to CAMs.

Our key contributions are as follows:

- We propose an algorithm to merge the optimal compressed accessibility maps for different types of operations into one ICAM. The result is a more compressed form of the accessibility maps which consolidate the information for each user/user group.

- We propose an efficient lookup method to determine the accessibility of a node with a time complexity of *O(depth of the requested node in the XML tree + log (the maximum fanout of the ICAM or CAM))*. This method applies to both CAM and ICAM, and its time complexity compares favorably with the previously known time complexity of *O(depth of the requested node in XML tree* $\times$ *log(CAM size))*.
- We verify the effectiveness of the ICAM and the lookup method for both space and time requirements by experiments.

2 CAM

An XML document is one kind of semistructured data, which can be represented as an XML tree (or XML database tree in [24]) with a node for each element, attribute, and value in the document, and with an arc between each element and each of its sub-elements or attributes or values and between each attribute and its value(s) [1]. [19] and [20] are the early works that describe a blueprint of the authorization mechanism for object-oriented and semistructured database. They denote an authorization by a 3-tuple, consisting of an authorization type (we refer to it as an operation type in this paper), an authorization object and a subject. The idea of implicit authorization is also introduced. That is, given a set of explicitly specified authorizations (authorization base), other authorizations, such as the operation type, object, and subject domains can be inferred. Authorization that is not stated explicitly and can be implied by authorization base is called implicit authorization. In this way, space can be saved by storing only the authorization base. Utilizing such an idea, the space for the accessibility map of an XML document can also be reduced.

CAM is one of the works that utilize the idea of implicit authorization along the dimension of authorization object. CAM stands for Compressed Accessibility Map. It is proposed by Ting Yu *et al* [24]. An accessibility map is generated by evaluating the authorizations against an XML document. It states the accessibility of each node in an XML document for a user. Storing the whole accessibility map takes up lots of space. In [24], the authors observe the structural locality of the accessibility of XML documents. That is, if a node is accessible (inaccessible) for some operation for a user, it is very likely that the same is true for its ancestors, its descendents, and its siblings. It becomes possible to record or label a node such that by default all the descendants are accessible (inaccessible), so that the labels of such descendants can be omitted to greatly reduce the number of labels. In other words the accessibilities of some nodes can be induced by the labels of the others. As a result, instead of storing the whole accessibility map, it is possible to store only a small portion of the accessibility information. They refer to the resulting map as the compressed accessibility map, i.e., CAM.

It is convenient to first assume that when a node is accessible, then all of its ancestors are also accessible. In other words, the descendants of an inaccessible node must also be inaccessible. But this assumption may not hold in reality. Ting Yu *et al.* solve this problem by decomposing the whole tree into regions, each of which satisfies the assumption. They called such a region a **unit region**. The construction and lookup algorithms for unit regions are modified to handle the general case.

First Step: The first step of constructing a CAM in a unit region is to assign labels to the nodes in an

XML tree of the form $(s*, d*)$, accordingly. Here '$*$' can be either '$+$' or '$-$'. $s+/s-$ means that the current node is accessible/inaccessible in terms of an operation type.[1] $d+/d-$ means that the descendants of the current node are also accessible/inaccessible unless they are overruled by the label of their closest ancestors. The notions of positive/negative/neutral nodes are proposed in [24] to determine the d part of the internal nodes during the labeling step. An **internal terminal node** refers to a non-leaf accessible node with no accessible descendants. (An **internal non-terminal node** is a non-leaf node that is either inaccessible or that has some accessible descendants.)

Definition 1 *[24] Given an operation, and an XML tree, a node e is* **positive/negative** *provided that e is an accessible/inaccessible leaf, or e is an internal non-terminal node with more positive/negative than negative/positive children. A node e is* **neutral** *if it is an internal non-terminal node with the same number of positive and negative children.* [2] □

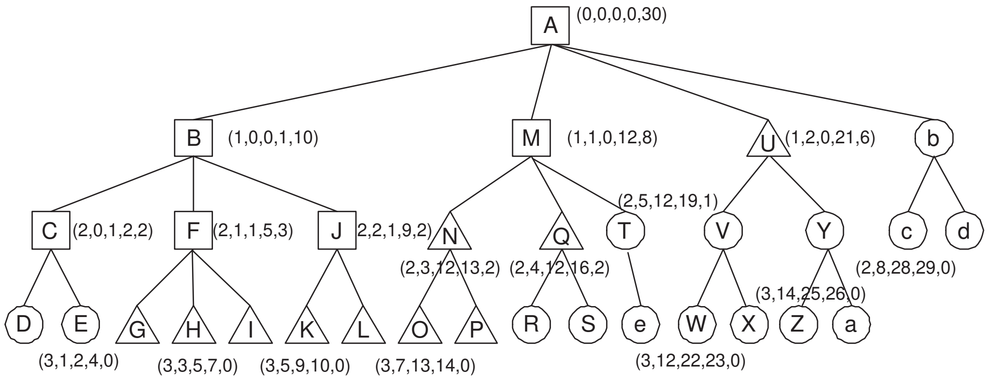


Figure 1: A marked XML tree

Here we give an example to illustrate the first step in the construction of a CAM in a unit region. The details of the algorithm can be found in [24].

Example 1 *Figure 1 shows an XML document in the form of a tree [1]. (Let us ignore the numerical labels in the tree for now.) The square nodes can be written and read, the triangular nodes can only be read, and the circular nodes are inaccessible. In order to construct a CAM for the read operation, we first assign labels to the nodes in the XML tree of the form $(s*, d*)$ according to the accessibilities for the read operation. The labeling is done bottom up.*
1. $L_r(D) = L_r(E) = (s-, d-)$ (inaccessible and negative).
2. $L_r(C) = (s+, d-)$ (internal terminal).
3. All the nodes in the subtrees rooted at F and J are labeled with $(s+, d+)$ (accessible and positive).

[1] When we say that a node is accessible for an operation, it means that the operation can be executed on the node, or the operation is permitted at the node.

[2] Note that an internal terminal node is neither positive, nor negative, nor neutral.

4. $L_r(B) = (s+, d+)$ (accessible and positive).
5. All nodes in the subtree rooted at N are labeled $(s+, d+)$ (accessible and positive).
6. $L_r(R) = L_r(S) = (s-, d-)$ (inaccessible and negative).
7. $L_r(Q) = (s+, d-)$ (internal terminal).
8. $L_r(T) = L_r(e) = (s-, d-)$ (inaccessible and negative).
9. $L_r(M)$ is deferred, for it is neutral.
10. All nodes in the subtree rooted at U (except U) are labeled $(s-, d-)$ (inaccessible and negative).
11. $L_r(U) = (s+, d-)$ (internal terminal).
12. All nodes in the subtree rooted at b is labeled with $(s-, d-)$ (inaccessible and negative).
13. A is neutral. As it is the root, we arbitrarily label it with $(s+, d-)$. It leads M to be labeled as $(s+, d-)$, too.

□

Second Step: The second step of the construction is to delete redundant labels in a certain order. There are two kinds of redundant labels in a labeled XML tree [24]. (1) The **subsumed** labels. A subsumed label is a label that is not in the CAM but that can be induced by other labels in the CAM. (2) The **upward redundant** labels. For such a label, the ‘s’ can be induced, and ‘d’ is immaterial because all the children of that node are either labeled or upward redundant in the CAM. (Please refer to [24] for more details.)

Definition 2 (Upward Redundant Label [24]) *A label of a node e in a CAM C is said to be $upward\ redundant$ if e has a accessible proper descendant[3], and for every child c of e, either c is labeled in C or c is upward redundant.*

Example 2 *We illustrate Step 2 of the construction by continuing with Example 1. In this step we delete the redundant labels (subsumed labels are deleted before upward redundant labels, while the order of subsumed and upward redundant label deletion is immaterial), we obtain the CAM for the read operation, which is shown in Figure 2a. For example, $L_r(D)$ and $L_r(E)$ are subsumed by $d-$ of $L_r(C)$. Thus they are deleted. There is no upward redundant label in this example. The CAM for the write operation is shown in Figure 2b.* □

Later on, we refer to a CAM for an operation x as CAM_x. The deleted labels for the XML tree can be induced from the labels in the CAM by Definition 3. For example, given CAM_r, the label of node M can be induced from the label of A and that of either N or Q.

Definition 3 *[24] Given a CAM for an operation z, CAM_z, the* induced label *at a node e in the corresponding labeled database tree (we use the term XML tree in this paper), written $L_z(e)$, is its label in CAM_z if one exists. Else, let y be the nearest labeled ancestor of e if any (note that e is also considered its own ancestor and its own descendent),*

[3] we consider e to be its own descendent, a descendant of e is proper if it is not equal to e

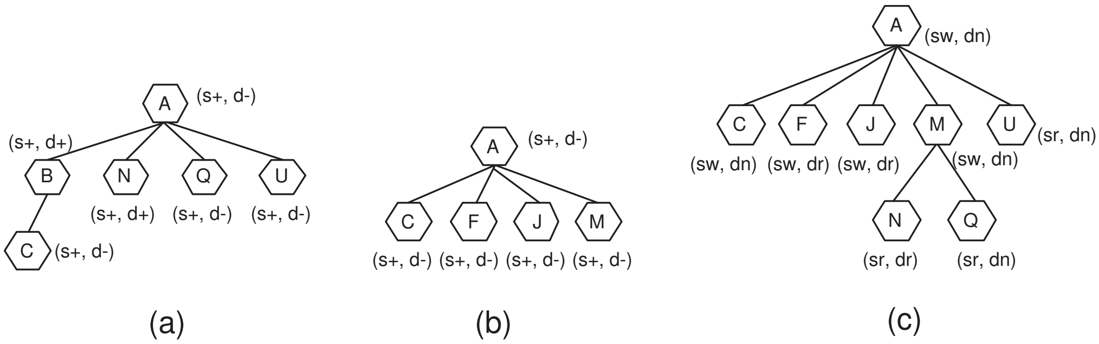


Figure 2: CAMs and ICAM for the example XML tree

- *If y has a label $(s+, d+)$, $L_z(e) = (s+, d+)$.*
- *If y has a label $(s-, d-)$, $L_z(e) = (s-, d-)$.*
- *If y has a label $(s+, d-)$,*
  - *If e has a descendant labeled either $(s+, d-)$ or $(s+, d+)$, $L_z(e) = (s+, d-)$*
  - *Else $L_z(e) = (s-, d-)$.*
- *If e has no labeled ancestor* [4], *$L_z(e) = (s+, d-)$.* □

3 Integrated CAM (ICAM)

3.1 Motivation of ICAM

In [24], if there are multiple operation types defined in the system, a different CAM is constructed for each type of operation and for each user group. We observe that the set of accessible nodes for different types of operations[5] may overlap, e.g. as shown in Figures 2a and 2b, node C is labeled in both CAMs for read and write operations. Assume that if the write operation is allowed, the read operation is automatically allowed. We say that write *covers* read. The "cover" relation of operations can be represented by an *operation hierarchy* in which a node *"covers"* its descendent nodes. With the assumption of such a hierarchy, it is possible to combine all the CAMs for different operations to form an **integrated CAM (ICAM)**. Given an XML tree, an ICAM is a compressed accessibility map where labels of a subset of the XML nodes

[4]In this case $L_z(e)$ is undefined in [24]. From the assumption that if a node is accessible, all its ancestors should be accessible, e has no labeled ancestor only when it and all its ancestors are upward redundant. Then, they should have at least one accessible labeled descendant by the definition of the upward redundant label. Hence, e should be accessible, implying that the 's' part of $L_z(e)$ must be '$+$'. While the 'd' part of its label is not essential for all of its children are labeled or upward redundant. We assign '$-$' to it in order to simplify our proof.

[5]In the remaining discussions we shall refer to "type of operation" simply as "operation", if no ambiguity may arise.

are recorded, the accessibility information of all operations have been integrated at each labeled node, and from this map, the accessibility of each node in the XML tree for each operation can be determined. If we use a label having a form of $(s*, d*)$ to represent the accessibility information of each node in an ICAM, the label can be obtained from merging the labels in the CAMs for each operation. Roughly speaking, in this example, we label a node with $(s*, d*)$ where $*$ can be r, w, n. The label sn means that the node is not accessible, sr means it is read accessible, sw means it is write accessible, which implies it is also read accessible. The d part can be understood similarly for the descendants of the current node. For example, Figure 2c shows an ICAM for the read and write operations of the XML tree in Figure 1. We will further explain the meanings of the labels in the following sections.

In practice, many authorization mechanisms, for example [19, 3, 6], assume an operation hierarchy, in which a *partial ordering* of operations exists, so that if a user is allowed to do one operation, some other operations are automatically permitted. If no explicit hierarchy exists, there is always an implicit hierarchy for the multiple operations, which is composed of **atomic** and **composite** operations. For example, the Unix file system supports three atomic operations: $read$, $write$ and $execute$ (r, w, x). An implicit operation hierarchy exists, which contains four composite operations: $ReadWrite$ (rw), $ReadExecute$ (rx), $WriteExecute$ (wx), $ReadWriteExecute$ (rwx), as illustrated in Figure 3a.

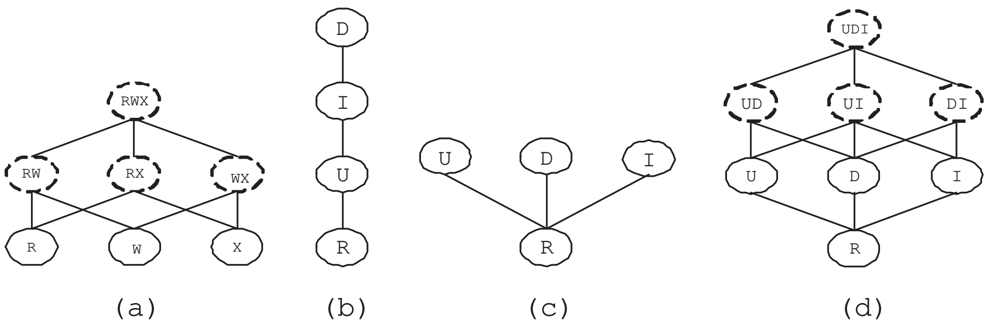


Figure 3: Operation hierarchies (The composite operations are represented in dash circle.)

Thus we claim that assuming an operation hierarchy is reasonable and beneficial. Under such an assumption, when the CAMs for different operations overlap in the node sets, the size of ICAM is smaller than the total size of the CAMs. Thus we can further reduce the space needed to store the accessibility map. In the following sections, unless specified, the term "operation" refers to both atomic and composite operations.

3.2 Operation Hierarchy

Here, we introduce the operation hierarchy more formally, where every element in the hierarchy stands for an allowed operation. And, an operation may "cover" other operations.

Definition 4 *Operation x* **covers** *operation y if whenever operation x is permitted at a node, operation y is also permitted at that node. We denote this by $x \geq y$. It defines a partial ordering on the operations.* □

We say that x **immediately covers** y if $x \geq y$ and there is no distinct z with $x \geq z \geq y$. If $x \geq y$ and $x \neq y$ we can write $x > y$. If x does not cover y and y does not cover x, we write $x \sim y$. A graph of the cover relation is the digraph whose vertices are the operations and which has an edge from x to y if and only if x immediately covers y. If we always draw x above y when $x > y$, we have an **operation hierarchy**. We make the following assumption.

Assumption 1: Given any two operations, x and y, in an operation hierarchy, where x and y have no "cover" relation, i.e. $x \sim y$, if it is possible that x and y are both allowed at a node in the XML tree, there should be a composite operation, o, s.t. o immediately covers both x and y. And, when x and y are both permitted at a node, o is permitted at the same time. □

Then, the label of o can be inferred from those of x and y. That is, if both the s part of x and y are '+', then the s part of o is '+'; Otherwise, it is '-'. d part of o's label can be inferred in the same way.

Figure 3 shows three possible operation hierarchies for four atomic operations, $Delete(D)$, $Update(U)$, $Insert(I)$, $Read(R)$. In Figure 3b, D covers all the other three operations; I covers U and R; and U covers only R. In Figure 3c, D, U, I cover R, but they do not cover each other, and at most one of them is permitted at a time. For example, once a node can be deleted, it can be read because D covers R. But it cannot be updated or inserted. If more than one operations are allowed at a time, the operation hierarchy becomes Figure 3d, in which one composite operation is created for each possible combination of atomic operations. And, these composite operations are presented in dash circles. An operation hierarchy has the following properties:

Property 1 *Every operation hierarchy implies a virtual null operation, n, denoting no operation is permitted. n is covered by all the other operations in the operation hierarchy. And, for every node, e, in the XML tree, $L_n(e) = (s+, d+)$.*

Property 2 *The operation hierarchy is a directed acyclic graph (DAG) and can be topologically sorted.*

An operation hierarchy is constructed by the "cover" relation, which is asymmetric. Hence, the graph on this relation cannot have a cycle. Thus the operation hierarchy is a DAG. A **topological sort** [2] of a digraph G is a sequence of all the nodes of G so that if x appears before y in the sequence, there is no path from y to x in G. It is proved by construction in [22] (Page 549) that every DAG can be topologically sorted. Hence, Property 2 holds. For example, one of the topological sorting of the operation hierarchy in Figure 3d is $UDI \gg UD \gg UI \gg DI \gg U \gg D \gg I \gg R$. And, it is easy to see that given two operations, x and y, in an operation hierarchy, $x \geq y$ implies $x \gg y$, but not vice versa. And, if $x \gg y$, we say that x is **larger** than y, or y is **smaller** than x.

It is obvious that the slimmer the operation hierarchy, the higher the probability that the CAMs will overlap, which leads to more space being saved. Even if there are no relationships among the atomic operations (see Figure 3a), it is still possible for the CAMs to overlap. That is, space can still be saved by the ICAM.

4 Construction and Lookup of ICAM

In this section, we will discuss the algorithm to construct and look up an ICAM efficiently.

4.1 Node Labels

The node label of the ICAM is an extension of that of the CAM. Recall that the label of a node, e, in CAM_x is of the form $(s*, d*)$, where $*$ can be '$+$' or '$-$'. Let us refer to such a label as $L_x(e)$. We use $L(e)$ to denote the label of node e in the integrated CAM (ICAM). It is of the form (sx, dy), where x and y are operations in the given operation hierarchy. The first part, sx, determines that the current node is accessible for operation x by the user. And, the second part, dy, denotes that the descendents of the current node are accessible for operation y by the user unless overruled by the label of a closer labeled ancestor.

For example, in Figure 2c, the label of node F, (sw, dr), means that node F is writable but its descendants can only be read unless overruled by a closer labeled ancestor.

It is possible that operation x is denied at node e, but permitted at a descendant of node e. In order to cope with this general case, we extend the idea of the unit region from [24] to the multi-operation case. That is, we decompose the tree into regions, such that within each region, for any operation x, if x is permitted at node e in the XML tree, it is also permitted at all the ancestors of e. We also call such a region a **unit region**. We first define the label merging rule and inducing rule based on one unit region. Then, we make some modifications to cope with the general case.

4.2 ICAM Construction for a Unit Region

In this subsection we will describe the algorithm of constructing an ICAM in a unit region. There are many possible ICAMs, but we aim at building a minimal one. The algorithm to construct an ICAM consists of the following main steps

1. For each atomic operation, label the XML tree according to the algorithm in [24], except that when encountering a neutral root, we choose a favorite label by Rule 1 (see below) instead, instead of assigning a label arbitrarily. Construct CAMs for the atomic operations. However, in this step, we do not remove the redundant labels in the CAMs.

2. From the CAMs construct an ICAM.

Rule 1 (Favorite Label for the Neutral Root) *Consider an XML tree and operations $x_1, x_2, \ldots, x_n$, and* *a topological sort $x_1 \gg x_2 \gg \ldots \gg x_n$, based on the cover relation. In the process of constructing CAM_{x_i}, if the root is neutral in terms of x_i we label the root as follows: We construct CAMs in the topological order of the operations, from x_1 to x_n, a)if $\exists x_j$, s.t. x_j immediately covers x_i, let $L_{x_i}(root) = L_{x_j}(root)$; b)else $L_{x_i}(root) = (s+, d+)$.* □

We follow the procedure to construct CAM_x for each operation x except that we do not really delete the redundant labels for x, instead they are marked with 'D' for "deleted". The CAM's are actually maintained by storing at each XML node a list of $(label, delete\_mark)$ pair, one pair for each operation, where $label$ is the label for an operation and $delete\_mark$ is 'D' or 'Null'. Then, all of the XML nodes whose labels are not all marked as 'D' will form the nodes in the ICAM. We merge the labels of such nodes using the **Label Merging Rule**.

Rule 2 (Label Merging Rule (LMR)) *Consider n different operations, $x_1, x_2, \ldots, x_n$, and a topological sort of $x_1 \gg x_2 \gg \ldots \gg x_n$ based on the cover relation. The integrated label of a node e in the ICAM is (sx, dy), where x is the smallest operation covering all x_i's such that $L_{x_i}(e) = (s+, d*)$; and y is the smallest operation covering all x_j's such that $L_{x_j}(e) = (s+, d+)$.* □

Note that if $X = x_1, x_2, ...x_n$ includes all operations in the corresponding operation hierarchy, then the minimal operation covering all x_i's such that $L_{x_i}(e) = (s+, d*)$ must also be in set X and it must have a label of $(s+, d*)$ for node e. Similarly, the minimal operation covering all x_j's such that $L_{x_j}(e) = (s+, d+)$ must be in X and have a label of $(s+, d+)$ for e. This is because if two or more operations are accessible at the node, then an operation must exist in the hierarchy which is the minimal operation covering these operations and it must also be accessible. Also due to this assumption, given a label (sx, dy) for node e in the ICAM, x will not cover any operation, x_i, such that $L_{x_i}(e) = (s-, d-)$. And, y will not cover any operation x_j, such that $L_{x_j}(e) = (s*, d-)$.

The label merging step may introduce new redundant labels. Since the labels in the ICAM are non-redundant in at least one CAM, the only possibility of redundant labeling in the ICAM is upward redundant labeling, whose definition is given in Definition 5 below. It was first introduced by [24], but multiple operations are considered here.

Definition 5 (Upward Redundant Label in an ICAM) *A label at node e in an ICAM I is said to be **upward redundant** if e has an accessible proper descendant for each operation, and for every child c of e, either c is labeled in I or c is upward redundant.*

We remove the upward redundant labels while we merge the labels by traversing the XML tree in post-order. Note that for a CAM, only the upward redundant labels without any labeled proper ancestor are removed. That is because any node with upward redundant label and with labeled proper ancestors are subsumed and deleted when the subsumed labels are removed. However, for an ICAM, a node is labeled after the merging step by LMR if it is not subsumed in at least one CAM, thus when removing the upward redundant labels, those with labeled ancestors should also be considered. Figure 4 describes the construction of a minimal ICAM. By a *minimal ICAM*, we refer to an ICAM with no redundant labeling due to subsumption or upward redundancy. Here, we given an example to show how a minimal ICAM is generated from CAMs by Algorithm 1.

Example 3 *Suppose there is an XML tree as shown in Figure 1 and the system supports two atomic operations, $r(read)$ and $w(write)$, where w covers r. We also assume another virtual operation, n, which denotes that no operation is permitted at the node. The topological order of these three operations is: $w \gg r \gg n$. Then, there are two steps to construct an ICAM.*

Step 1: For each atomic operation, we first label the XML tree by $(s, d*)$, where '*' can be '+' or '-'. Then, we mark the redundant labels. While CAM_r is being constructed, a neutral root (A) is encountered. According to Rule 1, as CAM_w is constructed first and $L_w(A) = (s+, d-)$ (A is accessible and negative for w. Note that M is an internal terminal node for w and is neither positive nor negative. Hence, A has one neutral child (B) and two negative children (U and b)), we choose $(s+, d-)$ for $L_r(A)$. As a result, a labeled XML tree is obtained, and each node in it maintains a label list. For operation w, all the labels except those of of A, C, F, J and M are marked 'D'. For operation r, all the labels except those of A, B, C, N, Q and U are marked 'D'.*

Step 2: For each node in the XML tree, if not all of its labels in the CAMs are marked 'D', we merge its labels for every operation. In this example, for node A, $L_w(A) = (s+, d-)$ and $L_r(A) = (s+, d-)$, thus $L(A) = (sw, dn)$. For node F, $L_w(F) = (s+, d-)$ and $L_r(F) = (s+, d+)$. Although $L_r(F)$ is marked 'D' for operation r, F has to be labeled in the ICAM, for it is not marked 'D' for w. Then, according to LMR, $L(F) = (sw, dr)$. Labels of the other nodes can be obtained in the same way. During the label merging, we also find that node B is upward redundant because B has an accessible labeled descendant for both w and r, and all its children are labeled. Thus we delete the label of B. The resulting ICAM is shown in Figure 2c. □

In the construction, the time needed for Step 1 is the product of the number of atomic operations and the time for constructing a CAM for one operation. As the time for constructing a CAM is linear to the number of nodes in the XML tree, and the number of the atomic operations can be considered as a constant, the time complexity for Step 1 is *O(number of nodes in the XML tree)*. The time complexity of Step 2 is also the same. Hence, the overall time for constructing an ICAM is proportional to the number of nodes in the XML tree.

If we refer to the number of nodes in a CAM/ICAM as the size of the CAM/ICAM, then the following theorem holds.

Theorem 1 *The size of ICAM is equal to or smaller than the total size of CAMs for different operations.*

Algorithm 1: Constructing a Minimal ICAM in a Unit Region

Step 1: Create CAMs for each operation
  For each atomic operation in the operation hierarchy
    label each node and add the label to the label list;
    Mark the redundant labels with 'D';
Step 2: Merge the labels of each node
  Traverse the XML tree in postorder, and for each node
    If it is upward redundant
      unlabel the node; (the node is not labeled in the ICAM)
    Else if not all of the labels in its label list are marked 'D'
      Infer the labels of the composite operations.
      Merge the labels by LMR

Figure 4: Constructing a minimal ICAM in a unit region

Proof
As shown in the algorithm, in the process of the label merging, we only consider any node which is not marked as redundant in at least one CAM. In other words, we only merge the labels of the nodes in the CAMs. Thus if the labeled nodes in all CAMs are different, then the size of the ICAM is equal to the total size of the CAMs, otherwise the size of the ICAM is strictly smaller.

Theorem 2 *Given an XML tree and atomic operations $x_1, x_2, \ldots, x_n$, $C_{x_1}, C_{x_2}, \ldots C_{x_n}$ are CAMs that label the root by Rule 1. $C'_{x_1}, C'_{x_2}, \ldots, C'_{x_n}$ are CAMs constructed in the same way except that they may not follow Rule 1. Then, the size of the ICAM generated from $C_{x_1}, C_{x_2}, \ldots, C_{x_n}$ is no larger than the one generated from $C'_{x_1}, C'_{x_2}, \ldots, C'_{x_n}$.* □

Theorem 3 *Given n optimal CAMs, $CAM_{x_1}, CAM_{x_2}, \ldots, CAM_{x_n}$, the ICAM generated by Algorithm 1 is minimal (in the number of nodes) in that no redundant node remains* □

The proofs of the above theorems are given in Section 9.

4.3 ICAM Lookup in a Unit Region

In this subsection, we describe the algorithm of looking up an ICAM in a unit region. The following lemma is essential for this purpose.

Lemma 1 *In a unit region, if the integrated label of a node e is (sx, dy), then $x \geq y$.*

Proof
Note that x and y are both permitted at e (in a unit region, if y is permitted at descendants of e, y is

also permitted at e). According to LMR, $x \geq y$, since x is the minimal operation covering all operations permitted at e in the CAMs.

Given a label in the ICAM, the label of a node in terms of an operation can be induced by the Label Inducing Rule.

Rule 3 (Label Inducing Rule (LIR)) *Given an ICAM and a node e in the corresponding XML tree, the induced label $IL_z(e)$ for operation z is determined according to the following rule:*

1. *If e is labeled in the ICAM, and $L(e) = (sx, dy)$*
   - *if $y \geq z$, then $IL_z(e) = (s+, d+)$;*
   - *else if $x \geq z$, then $IL_z(e) = (s+, d-)$;*
   - *else $IL_z(e) = (s-, d-)$;*
2. *Else, suppose f is the nearest labeled ancestor of e in the ICAM if any, and $L(f) = (sx, dy)$*
   - *if $y \geq z$, then $IL_z(e) = (s+, d+)$;*
   - *else if $(x \geq z)$ then*
     - *if e has a labeled descendant, p, in the ICAM, labeled (sm, dn), and $(m \geq z)$, then $IL_z(e) = (s+, d-)$;*
     - *else $IL_z(e) = (s-, d-)$;*
   - *else $IL_z(e) = (s-, d-)$;*
3. *Else if e has no labeled ancestor, $IL_z(e) = (s+, d-)$.* □

Example 4 *Consider the ICAM in Figure 2c as an example. Suppose $IL_r(F)$ is requested. Because F is labeled in the ICAM, $L(F) = (sw, dr)$, and $r \geq r$, $IL_r(F) = (s+, d+)$ according to the LIR. If $IL_r(B)$ is requested, we should locate its nearest labeled ancestor first. It is A and $L(A) = (sw, dn)$. Because $w \geq r$, but $\neg(n \geq r)$, we need to check if B has any labeled descendant. We find C, and $L(C) = (sw, dn)$. So, we get $IL_r(B) = (s+, d-)$ by LIR.* □

Note that the d part of $IL_r(B)$ and $L_r(B)$ are not equal. After the ICAM is constructed from LMR, and before the upward redundant labels are deleted, B should be labeled in the ICAM. In this ICAM, $L(B) = (sw, dr)$. Then, $IL_r(B) = (s+, d+)$ by LIR, which equals $L_r(B)$, as induced from CAM_r. When upward redundant labels are removed, $IL_r(B)$ becomes $(s+, d-)$. However, the d part of such an induced label does not affect the correct accessibility deduction for the other nodes. This is because the children of an upward redundant node are either labeled or upward redundant in the ICAM. The following theorem, which is proven in Section 9, makes sure that LIR is correct.

Theorem 4 *Suppose an ICAM, C, is generated by LMR for the operations $x_1, x_2, \ldots, x_n$, where upward redundant labels are not removed. Given a node e in the corresponding XML tree, its induced label for operation x_i in C is the same as its induced label in CAM_{x_i}.* □

After an induced label is obtained, the accessibility of the node is easily deduced, as in [24]. In particular, given a node e and $IL_z(e)$, the accessibility of e in terms of the operation z can be determined as follows: 1)If $IL_z(e) = (s+, d*)$, '*' can be '+' or '−', then e is accessible. 2)Else e is inaccessible. The following lemma is proved in Section 9.

Lemma 2 *The accessibilities of the nodes in an XML tree can be correctly induced from the ICAM generated by Algorithm 1.*

The lookup algorithm for the ICAM in a unit region aims at inducing the label for a requested operation for a requested node, which can be derived by following the steps outlined in the LIR rule. If we store the ICAM as a trie as suggested in [24], i.e. each node is identified by a string which is a prefix of the identifier of its descendants, then the time complexity of this lookup algorithm is *O(depth of the requested node in the XML tree × log(size of the ICAM))* [24]. In Section 5 we propose a mechanism to enhance the performance of the lookup algorithm.

4.4 Construction and Lookup for Multi-Unit Regions

In the CAM for a single operation with multiple unit regions, there exists some accessible node with an inaccessible parent. This node is called a **marker node**, and the inaccessible parent is called an **inter–region terminal node**, IRT for short [24]. In an ICAM, this situation also exists. We extend the definition of marker nodes. A marker node for an operation x refers to a node at which x is permitted but denied at its parent. We call an XML tree with at least one marker node a **Multi-Unit Region**. In this case, the construction and lookup algorithm need some modification.

According to the construction algorithm of multi-unit regions in [24], the parent of a marker node is never labeled in the CAM. That means the label $(s-, d+)$ is not allowed in the CAM. We make similar restriction by enforcing Lemma 1 on the labels of the ICAM.

The basic idea of constructing a CAM in multi-unit regions is to decompose the tree into several unit regions and reduce the tree in each individual region. Then, we combine the labels in each region to form a CAM, in which the marker nodes are marked. We adopt a similar idea, with some augmentation:

1. Label the IRT as $(s-, d-)$. As a result, the Label Merging Rule (LMR) can still be applied in multi-unit region case.

2. The marker nodes should be identified in the ICAM with the proper operation identifier. Even if its label is upward redundant, it will not be removed.

As for the lookup algorithm, there are details that are worth noting. Suppose we look for the accessibility of node e for operation z, and e is not labeled in the ICAM, nor is it a marker node. Suppose f is the nearest labeled ancestor, if any, of e in the ICAM, and $L(f) = (sx, dy)$.

1. If f is an IRT for operation z, $L_z(f) = (s-, d-)$ as we have mentioned above. Then, after merging the labels, we have $y < z$ and $x < z$. Thus $IL_z(e) = (s-, d-)$. On the other hand, $L_z(e) = (s-, d-)$ as it is a non-marker descendant of an IRT for operation z. Hence, the accessibility of e can be correctly induced in this special case.

2. If e has no labeled ancestor, or $L(f)$ satisfies $x \geq z$ and $\neg(y \geq z)$, we have to check whether e has a non-marker nearest labeled descendant with label (sm, dn) and $m \geq z$. If it does, e is accessible, otherwise, e is inaccessible in terms of operation z.

3. If f is the nearest labeled ancestor of e, and $y \geq z$, we have to further check whether e is an descendant of an IRT for operation z. That is, among the children of f in the ICAM, we look for d, which is a marker for operation z, and whose parent is the ancestor-or-self of e. If there is such a d, meaning e is a descendant of an IRT for z, then e is inaccessible, otherwise it is accessible in terms of operation z.

Items 2 and 3 are mentioned in [24], while item 1 is special for ICAM because an IRT is not labeled and cannot be the nearest labeled ancestor of any node in a CAM.

Using similar data representation as in [24], the time complexity of this algorithm is *O(depth of the requested node in XML tree* $\times$ *logarithm(size of ICAM))*.

5 Efficient Lookup

In this section, we propose an efficient way of lookup, which can be applied to both CAMs and ICAMs. By adopting a proper numbering scheme for the XML tree and creating indexes for both the XML tree and the CAM/ICAM, our lookup algorithm is more efficient than that of [24].

5.1 Numbering Scheme

One of the keys of the accessibility lookup is the determination of ancestor-descendant relationship, which can be assisted by a proper node identifier scheme. There are several methods to identify nodes in an XML tree. [24] adopts a scheme similar to the Dewey notation. Hence, the ancestor-descendant relation determination involves comparing two strings, which is time consuming. We apply a numbering scheme to represent nodes in an XML tree and the ICAM. To every node in the XML tree we assign a 5-tuple (*node-info*) and store it as an attribute of that node in the XML tree. The 5-tuple is ($level\_num$, $level\_order$, $parent\_order$, pre_order, $range$), where:

1. $level\_num$: the level of the node. The root belongs to level 0.
2. $level\_order$: the left to right order of the node on its level. Also begins with 0.

3. $parent\_order$: the preorder number of the direct parent of the node.

4. pre_order: the preorder number of the node.

5. $range$: the number of descendants of the node.

Figure 1 shows an example XML tree with *nodeinfo* (we omit some *nodeinfo*'s due to space limitations). Among these five numbers, $level\_num$, pre_order, $range$ are similar to those of the numbering scheme proposed by [16][6]. Using these three numbers, the structural relation between two nodes can be determined in constant time according to a lemma in [16]. In our case, given two nodes, x and y, with $xinfo$ and $yinfo$ as their 5-tuple *node info* respectively, then x is y's ancestor iff $xinfo.pre\_order < yinfo.pre\_order \leq xinfo.pre\_order + xinfo.range$. Also, if $xinfo.level\_number = yinfo.level\_number - 1$, then x is the parent of y.

5.2 Storage Mechanism of ICAM

Suppose an ICAM is created for a user or user-group for the original XML tree. It is stored as a table. Here, we use an example to explain the mechanism. Suppose we want to create an accessibility map for user A for an XML tree T. The XML tree as well as the accessibility information is shown in Figure 1, and the ICAM of that tree is shown in Figure 2c. Figure 5 shows a table for the logical structure of the ICAM and also the example data. Each row in the table corresponds to a node in the ICAM. It has three elements: $Node\_Info$ is the assigned 5-tuple of the node, $Access\_Label$ records the accessibility information, i.e. the label in the ICAM, while $ptrChildren$ contains the pointers to its direct children in the ICAM. (In practice, in order to avoid variable length arrays, two pointers can be used instead of a list of child pointers, one of which points to the first child and the other to the next sibling.)

	Node_Info	Access_Label	ptrChildren
0	(0,0,0,0,30)	(sw,dn)	(1,2,3,4,7)
1	(2,0,1,2,2)	(sw,dn)	NULL
2	(2,1,1,5,3)	(sw,dr)	NULL
3	(2,2,1,9,2)	(sw,dr)	NULL
4	(1,1,0,12,8)	(sw,dn)	(5,6)
5	(2,3,12,13,2)	(sr,dr)	NULL
6	(2,4,12,16,2)	(sr,dn)	NULL
7	(1,2,0,21,6)	(sr,dn)	NULL

Figure 5: The logical table structure of an ICAM

5.3 Indexes on the XML Tree and the ICAM

Indexes are created to quickly locate nodes in an XML tree and the ICAM.

[6] It is easy to extend to a duration numbering scheme [16] with an $extended\ preorder$ to accommodate future insertions. Since we do not discuss the update of CAM and ICAM, using $preorder$ is convenient for illustrating our idea.

1. We create an index array on the source XML tree. The method flattens the XML tree by traversing it in preorder, and an array ($Index\_Arr$) is built. The ith element of the array has a pointer pointing to the ith node in the XML tree in preorder. Thus, as long as the preorder of a node is known, we can locate the node and get its *nodeinfo* in constant time. Using the $parent\_order$, combined with the index array of the XML tree, we can traverse from the node to the root of the XML tree in a time proportional to the depth of the node.

2. A 2-layer hash index[7] is created on the two elements of *nodeinfo*, $level\_num$ and $level\_order$, to quickly locate entries in the ICAM. We first hash the nodes on the same level of the XML tree into one bucket. The level of a node can be obtained by the first element of its 5-tuple *nodeinfo*. If h_1 is the hashing function, we have:

   $FirstLayerBucketNumber(x) = h_1(x.nodeinfo[1])$

   Let x be a node in the ICAM with the highest $level\_num$. Then, the total number of buckets of the first layer of hashing is $x.nodeinfo[1] + 1$. which is equal to $Maximum\_ICAM\_Level$. For example, for the ICAM stored in Figure 5 , it is 3.

   Then, in each bucket, we further hash the node according to its order among its siblings. The second layer hash function h_2 gives

   $SecondLayerBucketNumber(x) = h_2(x.nodeinfo[2])$

The hash table allows a constant average lookup time given sufficient storage space. There is a trade off between speed and space. We can adjust the hash function to reduce the number of empty buckets, while maintaining efficient lookup time. Assisted by the above two indexes, the nearest labeled ancestor of a requested node can be located in time proportional to the depth of the requested node. Figure 6 describes the procedure.

Algorithm 2: Locate the nearest labeled ancestor of a requested node

```
let e be the requested node, let f be its nearest labeled ancestor;
f = e;
while (f ≠ NULL )
    B1 = h1(f.nodeinfo[1]) ;      % hash on level number
    if B1 is not empty
        B2 = h2(f.nodeinfo[2]);      % hash on level order
        if B2 is not empty, return f;
    f = Index_Arr[f.nodeinfo[3]];      %get parent order
```

Figure 6: Locate the nearest labeled ancestor of a requested node

[7] Note that we could have hashed on one layer only, e.g. if we only hash on the pre-order, then there is no need of the level-order. But in the case when ICAM is much smaller than the XML tree, there maybe a lot of empty buckets using a one layer hash index. By using a 2-layer hash index, we can adjust the number of entries of the second layer, which will save some space.

5.4 Storing the "Cover" relation in an Operation Hierarchy

A "cover" matrix, M, is created based on the operation hierarchy in order to determine the "cover" relation between two operations. It is an $n \times n$ matrix, where n is the number of operations, including atomic and composite operations. $M[i, j] = 1$ iff operation i covers j. In this way, the "cover" relation between two operations can be determined in constant time. Next, we will discuss how to determine a node's accessibility efficiently.

5.5 Lookup of the Accessibility Information

In this section, we explain our lookup algorithm. The key of determining a given node's accessibility is to compute its induced label. Here we use ICAMs as an example, but note that our algorithm and the following discussion are also applicable to CAMs. The first step of computing the induced label is to locate the nearest labeled ancestor, which is described in Figure 6. Then, LIR is followed. We give an example to illustrate the procedure.

Example 5 *Suppose a request arrives, say an operation w at node T in the XML tree (Figure 1). We determine its accessibility in the following way.*

1. If the nodeinfo of the node is stored with the node in the XML tree, then we can get the nodeinfo of the requested node (i.e. T) directly. Let $T.nodeinfo$ = (2,5,12,19,1).

2. Calculate the first layer hash function to get to the first layer bucket. We find that 2 $\leq Maximum\_ICAM\_Level$. That means that T may be labeled in ICAM. Further hash it by its $level\_order$, that is 5. We find out that it is not labeled in the ICAM, for the number 5 bucket in the second layer is empty.

3. Get the parent of node T by $T.nodeinfo[3]$, the preorder number of T's parent. Given this number, we can make use of the index array ($Index\_Arr$) to locate the parent in constant time and obtain the nodeinfo of T's parent, i.e M. And, $M.nodeinfo$ = (1,1,0,12,8).

4. By calculating the value of the first layer hash function for M, we find that it should appear in the first bucket of the hash table. Further calculate the second layer hash function to locate M in the first bucket. Thus we get the entry of M in the ICAM table. $L(M) = (sw, dn)$. As $\neg(n \geq w)$, we need to find out whether T has a labeled descendant.

5. From the ptrChildren field of the M entry, we know that M has two direct children in the ICAM, that is, N and Q. $N.nodeinfo$= (2,3,12,13,2) and $Q.nodeinfo$ = (2,4,12,16,2). Both of them are not the descendants of T, for 13 and 16 do not fall in the range of (19, 20]. We know that if T were an ancestor of N, it should satisfy the inequality: $T.nodeinfo.pre\_order < N.nodeinfo.pre\_order \leq T.nodeinfo.pre\_order + T.nodeinfo.range$. So does Q. Therefore, we deduce that T has no accessible descendants. Hence, by LIR, $IL_w(T) = (s-, d-)$. So, we know that T is inaccessible in terms of operation w. □

5.6 Time Complexity

We state in this subsection the results concerning the time complexity of the algorithm for looking up a node in the ICAM.

1. The time needed to assign the *nodeinfo* to nodes. This is done only once for every XML tree. The time is proportional to the number of nodes in the XML tree.

2. The time to construct the ICAM. The time is proportional to the XML tree size (number of nodes in the XML tree [24]).

3. The lookup time. Since the requests may be frequent and the users may not want to wait a long time for the results, this is the time we try to reduce. There are two components: 1)The time needed for looking up a requested node's nearest labeled ancestor in the ICAM. It is proportional to the number of levels in the XML tree between the requested node and its nearest labeled ancestor in the ICAM. 2)For a CAM, the label for an operation is obtained directly. While for an ICAM, the label for an operation is induced from the ICAM as well as from the cover relation between the two operations. But based on the cover matrix for the operation hierarchy, the cover relation between two operations can be determined in constant time, implying the label for an operation can be induced in constant time given the label in the ICAM. If the label of the nearest labeled ancestor is (sx, dy), and $x \geq z$ but $\neg(y \geq z)$, or the requested node has no labeled ancestor, we need to determine if the requested node has any accessible descendants. So we have to search in the children of this labeled ancestor in the ICAM, or the roots of the ICAM if no labeled ancestor exists, to see if there exist any descendants of the requested node with label (sm, dn), and $m \geq z$. It will take time proportional to the fanout of the requested node's nearest labeled ancestor in the ICAM.

Lemma 3 *The overall time of lookup is O(depth of the requested node in the XML tree + the maximum fanout of CAM/ICAM).*

But if we sort the children of each node in the ICAM by their pre-order numbers, the time complexity will be reduced to be the logarithm of the fanout. This is because we can determine the range of pre-order numbers for descendants of the requested node and we can search the sorted list by binary search in logarithmic time.

Lemma 4 *If the child-pointers of each ICAM node are sorted, the overall time of lookup is O(depth of the requested node in the XML tree + log(the maximum fanout of CAM/ICAM)).*

Compared with the time complexity of [24], which is *O(depth of the requested node in the XML tree* $\times$ *log(CAM/ICAM size))*, the proposed algorithm has a much lower complexity.

5.7 Space Requirement

We now discuss how we measure the space for CAM/ICAM. It includes the space for the CAM/ICAM, its hash index and the index on the original XML tree.

The space needed for CAM/ICAM is the number of nodes in CAM/ICAM times the number of bits needed for each node, which includes the node identifier, the label, and the necessary pointers. For the node identifier, if we adopt the 5-tuple *nodeinfo* as the node identifier, 4 bytes for each number, then totally 20 bytes are needed.[8]

Pointers are needed to link the nodes in the CAM/ICAM. In practice for each node two pointers are maintained, one linking to its first child, the other to its right sibling. Assuming that each pointer requires 4 bytes, then 8 bytes are needed for storing the pointers of each node in the CAM/ICAM.

As for the label, in a CAM, three bits are enough for a label, one for the 's' part, one for the 'd' part, and the third one for a marker. While in an ICAM, the number of the bits needed for each part of a label depends on the number of operations. E.g., in the hierarchy in Figure 3d, there are eight operations, R, U, D, I, UD, UI, DI, UDI. Thus three bits are needed to distinguish them. The 's' and 'd' parts require 3 bits each. (In general, for n operations, we need $\log n$ bits each.) We also need to indicate whether a node is a marker for an operation or not. One marker bit has to be assigned to each atomic operation, i.e. 4 bits are needed in total.

In other words, if we adopt 5-tuple *nodeinfo* for both CAM and ICAM. The number of the bits needed for each node in CAM/ICAM is given by the following equations:

$$CAM: \qquad (20 \times 8) + (8 \times 8) + 3 = 227(\text{bit}) \tag{1}$$

$$ICAM: \qquad (20 \times 8) + (8 \times 8) + (3 \times 2) + 4 = 234(\text{bit}) \tag{2}$$

The size of the hash index is affected by two factors: the size of the CAM/ICAM, and the distribution of nodes in the CAM/ICAM. The number of non-empty buckets is bounded by the level of CAM/ICAM times the maximum number of the nodes on one level. Note that for the CAM, each hash index has be to created for each operation. But the ICAM requires only one hash index. The index array of the original XML tree, which is stored outside the XML tree, takes the space proportional to the number of nodes in the XML tree. These indexes are used to speed up the lookup. The same algorithm can also be applied to the CAM.

[8] If we use the method of [24], the length of a node identifier depends on the depth of the node. Take a full binary XML tree as an example, the average length of the identifier can be computed by $\frac{1}{2^k-1}\sum_{i=1}^{k} i \times 2^{i-1}$, where k denotes the level of the tree beginning from 1. For a 18-level binary tree, the average length of the node identifier is 17. Using 4 bytes for each symbol, then on average 68 bytes are needed to identify a node, which is much longer.

6 Extension of ICAM

To further reduce lookup time, we can add nodes into the ICAM. After constructing the ICAM, do one more traversal of the ICAM. If we observe that the XML level between the parent and the child in the ICAM is bigger than a certain threshold, we can add more nodes on this path simply by re-inserting the deleted nodes. We call the resulting ICAM as Extended ICAM (EICAM). The size of EICAM is controlled by the value of the threshold.

Lemma 5 *EICAM represents the same accessibility as the original ICAM.*

Proof
Here, we prove that for any CAM_z, labels of the other nodes can still be correctly induced after adding back an induced label l (the resulting CAM is denoted by CAM'_z). For we have proven in Theorem 3 that after the label merging, the labels of the nodes for an operation can be correctly induced.

Thus, given an unlabeled node e, we prove that the induced label $L_z(e)$ in CAM_z equals to the induced label $L'_z(e)$ in CAM'_z. Let f and g denote the nearest labeled ancestor and descendant of e in CAM_z, resp., if any. Adding l to CAM_z can affect the induced label of e only if l become e's nearest labeled ancestor or descendant in CAM'_z.

CASE 1: l is the nearest labeled ancestor of e in CAM'_z.

If $L_z(f) = (s+, d+)$ (or $(s-, d-)$, resp.), according to Definition 3, $L_z(l) = L_z(e) = (s+, d+)$ (or $(s-, d-)$, resp.). On the other hand, in CAM'_z, since $L'_z(l) = (s+, d+)$ (or $(s-, d-)$, resp.), $L'_z(e) = (s+, d+)$ (or $(s-, d-)$, resp.), equals to $L_z(e)$.

If $L_z(f) = (s+, d-)$ or no f exists, $L_z(e)$ and $L_z(l)$ can be $(s+, d-)$ or $(s-, d-)$, depending on whether e and l have an accessible descendant. If there is such g with label $(s+, d*)$, $L_z(e) = L_z(l) = (s+, d-) = L'_z(l)$. And, it is obvious that $L'_z(e)$ also equals $(s+, d-)$ in this case. If there is no such g. $L_z(e) = L_z(l) = (s-, d-) = L'_z(l)$. Then, in CAM'_z, since $L'_z(l) = (s-, d-)$, $L'_z(e) = (s-, d-) = L_z(e)$.

CASE 2: l is the nearest labeled descendant of e in CAM'_z.

If $L_z(f) = (s+, d+)$ (or $(s-, d-)$, resp.), $L_z(e) = L'_z(e) = (s+, d+)$ (or $(s-, d-)$, resp.) irrespective of its labeled descendants.

If $L_z(f) = (s+, d-)$ or no such f. If e has a nearest labeled descendant, say g, in CAM_z. It should also be the nearest labeled descendant of l, otherwise, it should be the label of g that we have re-inserted, instead of l. Thus if $L_z(g) = (s+, d*)$, $L_z(e) = L_z(l) = (s+, d-) = L'_z(l)$. It is obvious that in this case, $L'_z(e) = (s+, d-)$, too. It e has no such g in CAM_z, l does not either. Thus $L_z(e) = L_z(l) = (s-, d-)$. Then, in CAM'_z, $L'_z(e) = (s-, d-)$.

Therefore, the label of e can still be correctly induced in CAM'_z.

In this way, we can further reduce the time for looking up the nearest labeled ancestor to the value of threshold. It is the tradeoff between the space and the lookup time.

Lemma 6 *The lookup time of the EICAM is independent of the depth of the node in the XML tree*

Proof
As discussed in Section 5.6, looking up a node in an ICAM involves locating a nearest labeled ancestor and descendants. In the ICAM, locating a nearest labeled ancestor is linear to the depth of the requested node. While in the EICAM, by adding back an induced label, the number of the nodes on the path from the nearest labeled ancestor of the requested node to the requested node in the XML tree is upper bounded by the threshold, i.e. irrespective of the depth of the node. While the time complexity of looking for a nearest label descendant is still *O(log(maximum fanout of EICAM))*. In other words, looking up a node in the EICAM is independent of the depth of the node in the XML tree.

7 Experimental Results

We have conducted experiments to show how the use of ICAM can help reduce the storage size of the accessibility maps and verify the efficiency of our lookup algorithm. All the experiments are conducted on a SUN Enterprise E4500, running Solaris 7 and having 8G RAM. The synthetic XML data is generated by the IBM XMLGenerator [9]. There are several parameters in the XMLGenerator to control the size, the depth and the breadth (i.e. the average number of nodes on one level) of the input XML tree. As a result, we can verify our algorithms on the XML trees of different sizes and shapes. The real data is the accessibility control information of the Unix file system of a big university. We report here the experiments we conducted to test space efficiency as well as the construction and lookup time complexities of CAM, ICAM and the full materialized map (**FMM**). The full materialized map is the accessibility map containing all the accessible nodes and no inaccessible node while maintaining the structural relationship among the nodes.

We adopt two mechanisms for node identity, one is the trie mechanism, as is used for CAM in [24]; the other is the numbering scheme we described in Section 5.1. We refer to these two kinds of CAM's as **CAM_T** (T stands for TRIE), and **CAM_N** (N stands for numbering scheme), resp.. Similarly, we distinguish between **FMM_T**, and **FMM_N**. We adopt the operation hierarchy in Figure 3d in the experiments, and use the encoding method to represent the operations. Note that only the CAMs for the atomic operations are built for **CAM_N**, **CAM_T** or ICAM.

7.1 Experiments on Space Efficiency

In the experiments on space efficiency, we use the same set of synthetic data as in [24]. The parameters used in the experiments are as follows: the total number of nodes in the XML tree is 16811; fanout is varied from 1 to 60, and has an average value of 2 (not including the leaves); the average depth of the tree is 8. We also adopt parameters from [24] to control the accessibility ratio and access locality. They call a node *friendly* if the nodes in the subtree rooted at that node have a high probability of being accessible. The subtree rooted at this root is called a *friendly area*. Otherwise, the node is called *non-friendly* and the subtree rooted at the node is a *non-friendly area*. The related parameters are:

af/anf : The access probability of a node in a *friendly/non-friendly* area. In the experiments, we set these two parameters to be **98%** and **2%**, respectively, as in [24], to obtain high structural accessibility locality.

fr/rr : The *friendly ratio/reverse ratio*. It is the probability that a node is a *friendly/non-friendly* node given that its parent is a *non-friendly/friendly* node. We have to keep *fr* small in order to achieve high access locality. In the experiments, we set it to be **5%**. While *rr* is varied to control the accessibility ratio. In the experiment, we vary *rr* from **0%** to **100%**. These settings are also followed in [24].

ar : The accessibility ratio (ar) is given by $1 - rr$. This factor is an indication of the amount of nodes that are accessible in the XML tree.

Besides the above parameters, we adopt another parameter to simulate the accessibility distribution among the nodes. We called it **Accessibility Inheritance Probability (aip)**. It is defined as follows. Given an XML tree and a node e in the tree, for two different operations, x and y, where $x > y$, suppose y is permitted at e, then aip is a conditional probability that x is also permitted at e. In the experiment, we vary aip from **0%** to **100%** to get various different accessibility similarities among the operations.

We generate the accessibility information of the data as follows. Considering an operation hierarchy in Figure 3b, 3c or 3d, af/anf, fr/rr and ar are to determine whether a node is accessible for R. We call the nodes that are accessible at least for R as *accessible nodes*. Then, among all the accessible nodes, aip is used to control the accessibility probability of the other operations in the operation hierarchy. Hence, we keep *fr* low to achieve high access locality for R. For the other operations, U, I and D, their access probabilities are controlled by aip, and the access locality is not guaranteed.

The hierarchy in Figure 3a is for the experiments on the Unix file system data only. As the accessibility information is already there, we do not need the above parameters to generate the accessibility information.

In the experiments we use the same definition of Compress Ratio as that in [24].

Definition 6 (Compress Ratio) *The Compress Ratio (CR) of an ICAM is defined by the following equation:*

$$\text{CR} = \frac{\text{Sizeof}(ICAM)}{\text{Total number of accessible nodes}} \tag{3}$$

The size of a CAM or an ICAM is given by the number of nodes in the CAM and ICAM, respectively. We also define the term Gain Ratio, which is used to measure the space saved by combining all CAMs into one ICAM.

Definition 7 (Gain Ratio) *The Gain Ratio (GR) of an ICAM integrated from $CAM_1, ...CAM_n$ is defined by the following equation:*

$$\mathrm{GR} = \frac{\sum_{k=1}^{n} \mathrm{BitsOf}(CAM_k) - \mathrm{BitsOf}(ICAM)}{\sum_{k=1}^{n} \mathrm{BitsOf}(CAM_k)} \tag{4}$$

The BitsOf() function computes the product of the number of nodes in CAM/ICAM and the number of bits of each node. The latter part is determined by Equation 1 and 2, respectively (see Section 5.7). Since the numbering scheme has obvious advantages, from both complexity analysis and empirical studies in construction time and accessibility lookup (see results in this section), we assume that both CAM and ICAM adopt the 5-tuple *nodeinfo* as identifiers when calculating the Gain Ratio, i.e. CAM_N is considered here.

Compress ratio and gain ratio are two target measurements in our experiments. We study the influence of various parameters on these two ratios. There are several conditions that may affect these two targets: access locality, accessibility ratio, similarities among the CAMs for different operations and the size and shape of the operation hierarchy.

7.1.1 Compress Ratio

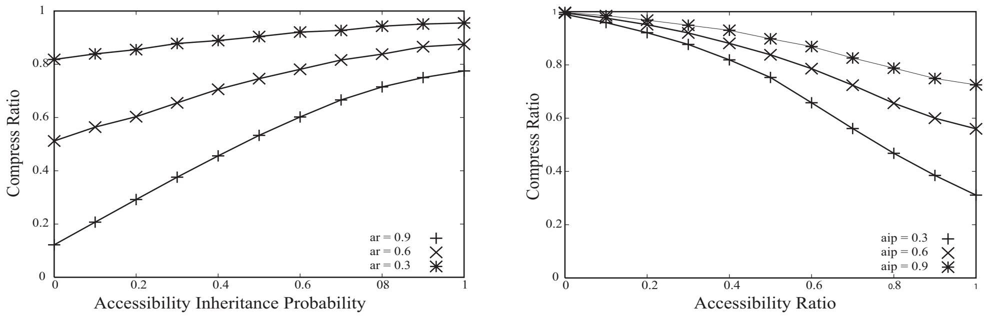
In this subsection, we will discuss how the above parameters affect the compress ratio.

[Compress Ratio Versus Accessibility Inheritance Probability]
Accessibility Ratio (*ar*) is fixed at 30%, 60%, and 90% , representing small, medium and high accessibility ratio, respectively. *aip* is varied from 0% to 100%. The result is shown in Figure 7a. From the figure we observe that CR increases when aip increases. That is because when aip increases, the number of nodes that are permitted to do the operations on the second layer or above, i.e. operation U, I and D, increases, too. That leads to much increase in the number of nodes labeled in the ICAM. Meanwhile, the number of accessible nodes remains unchanged. Hence, the compress ratio increases.

[Compress Ratio Versus Accessibility Ratio]
aip is fixed at 30%, 60% and 90%, representing small, medium and high accessibility inheritance probability, respectively. And, *ar* is varied from 0% to 100%. The result is shown in Figure 7b. The result shows that CR decreases with the increasing of *ar*.

7.1.2 Gain Ratio

This subsection lists the result of the experiments on gain ratio.



(a) (b)

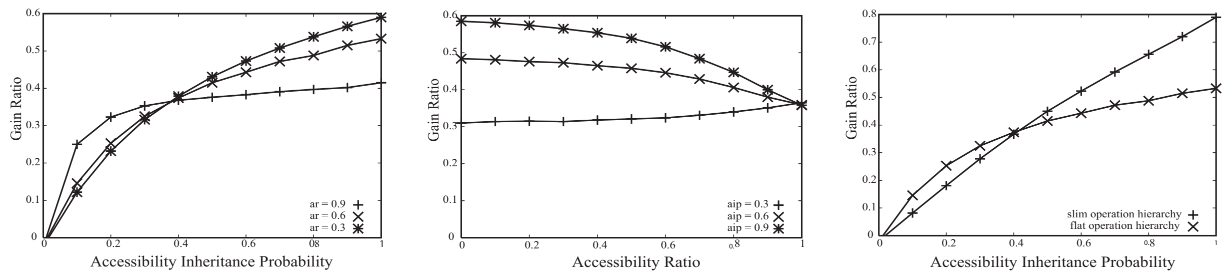
Figure 7: Results about Compress Ratio

[Gain Ratio Versus Accessibility Inheritance Probability]

ar is fixed at 30%, 60%, and 90%, respectively, representing small, medium and high accessibility ratio. The value of *aip* is varied from 0% to 100%. The result is shown in Figure 8a. From the figure we observe that GR increases as *aip* increases. That means the higher the accessibility inheritance probability, the more space is saved by ICAM. The highest gain ratio achieved is near 0.6, meaning that close to 60% of the space is saved.

[Gain Ratio Versus Accessibility Ratio]

The value of *aip* is fixed at 30%, 60% and 90%, representing small, medium and high accessibility inheritance probability, respectively. ar is varied from 0% to 100%. The result is shown in Figure 8b. The result shows that the gain ratio changes little with the accessibility ratio. That means the accessibility ratio has little effect on the space saved by ICAM.



(a) (b) (c)

Figure 8: Results about Gain Ratio

[Gain Ratio Versus the Shape of the Operation Hierarchy]

This experiment shows the influence of the shape of operation hierarchy to the gain ratio. Here we compare the gain ratio when the system supports the operation hierarchy of Figure 3b and 3d, resp.. Due to page limitation, we only list the result in the case of the medium accessibility ratio (ar = 60%). As plotted in Figure 8c, the gain ratio of the slim hierarchy increases faster with aip than that of the flat hierarchy.

Because the slimmer the hierarchy, the higher the probability of the overlapping of the CAMs, which leads to greater gain ratio.

7.2 Construction Time

Experiments are conducted to compare the construction time of CAM_N, CAM_T, and ICAM. The construction of a CAM or an ICAM includes two processes, constructing and storing. When creating an ICAM for several operations, the merging and storing of labels is done only once for all operations. While to construct several CAMs, the storing process has to be done once for each operation.

In order to maintain the structural relationship among the nodes in the CAM and ICAM, the descendant-ancestor relationship of two nodes has to be determined when linking the nodes, and the efficiency of this step greatly affects the construction time. If the trie scheme is used to encode node identifier, the determination takes much longer time because of string comparison. While the time complexity becomes constant if the numbering scheme is used. That is why the construction time of CAM_T is longer than those of CAM_N and ICAM, as the results of the experiments show.

We conduct the experiments under several situations. But due to space limitation, we only show the results with the following settings: ar=60%, fr=5%, af=98%, anf=2%, aip=60%. In other words, the following results are for the case of medium accessibility ratio, access locality and accessibility inheritance probability.

XML trees of different sizes are generated by IBM XMLGenerator. We compare the average construction time of CAM_N, CAM_T, and ICAM for four operations. The results are given in Figure 9. The results show that the ICAM and CAM_N are constructed more than two times faster than CAM_T. The construction time is linear to the size of the XML tree. Label merging and storing do not cost extra overhead when constructing an ICAM. The experimental results confirm the earlier performance claims.

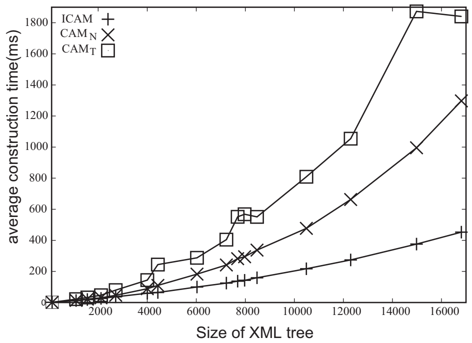


Figure 9: Construction time for 4 operations

7.3 Experiments on Accessibility Lookup

In this section, we present the experiments and their results to verify the performance claims on the lookup. The same data set as in Section 7.1 is used for these experiments.

The parameters for controlling accessibility assignment are set as in the previous experiments on construction time. First we apply our numbering scheme and indexes to the CAM (CAM_N) and compare the lookup time of our algorithm with that of the original CAM (CAM_T). The result, as shown in Figure 10a, verifies that our lookup algorithm is much more efficient. As a result, we only consider the case of using numbering scheme in the following experiments.

Then, we examine the effects of integrating multiple operations. We first test the average lookup time when the user asks for only one operation. Figure 10b shows the average lookup time of each level of an XML tree. The result shows that merging labels in the ICAM does not cost extra time when a user wants to look up only one operation. And, the lookup in both CAM and ICAM are faster than that in FMM_N. Because when looking up in the ICAM/CAM, most searching stops once a labeled ancestor is found. But the lookup in a FMM usually needs to travel the whole path from the root to the requested node if accessible or to one of its descendants if inaccessible. The average lookup time for the nodes at level one in the ICAM/CAM is larger because there are many more labeled nodes at that level.

The third experiment shows the average lookup time of CAM_N and ICAM for two operations. As illustrated in Figure 10c, the ICAM is optimal when the user wants to lookup multiple operations, for the nearest labeled ancestor has to be located only once.

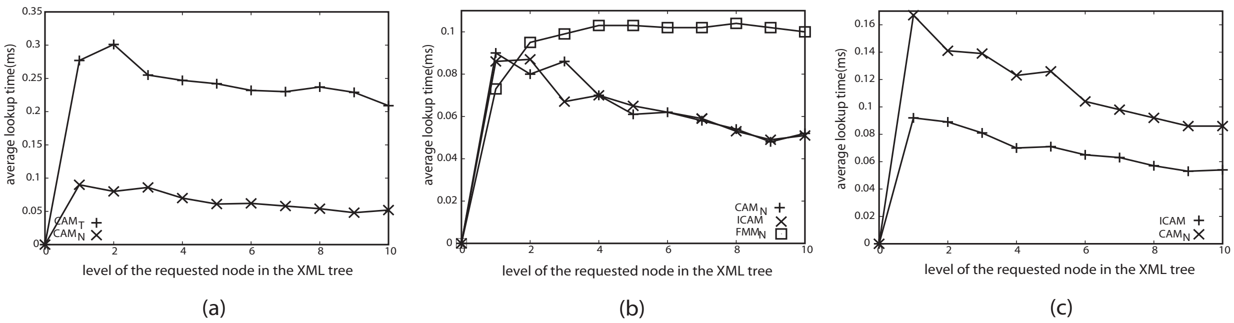


Figure 10: The average lookup time

7.4 Experiments on Real Data Set

We also conduct the experiments on a real data set. The data set is the UNIX file system data, which has the similar structural and accessibility characteristic with the XML data. Our data is obtained from an existing system of the computer science department of the Chinese University of Hong Kong. It has 408,561 nodes and 271 users. The maximum fanout of the file structure is 3,033, and the average is 15. The maximum depth is 15, and the average is 7. We assume that the three operations, $read(r)$, $write(w)$, and $execute(x)$, form an operation hierarchy shown in Figure 3a. We construct the ICAM and CAM_Ns on the data. The average compress ratio of the ICAM is 0.3, and the average gain ratio of ICAM over CAM_N is 20%. The gain ratio is not very big. As the file system has high accessibility locality, the compress ratios of the CAMs are good. Hence, the gain of the ICAM over the CAMs is not that much. But it still saves about 20% space. We also compare their construction time and lookup time. The average construction time of the CAM_N for the three operations is about 135.5 seconds, and that for the ICAM is only about 81.7 seconds. Figure 11 plots the lookup time of two operations in the ICAM and CAMs, resp.. It shows that the lookup in the ICAM is faster than in the CAMs in all the tested cases.

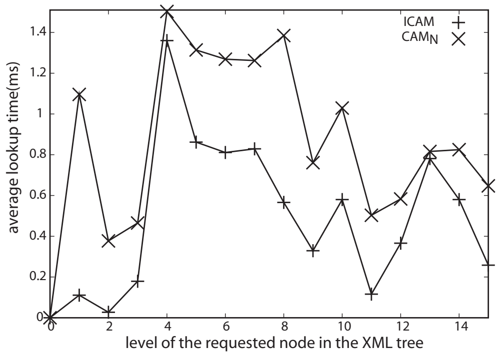


Figure 11: The average lookup time of the real data

7.5 Discussion

The experimental results can be explained as follows:

1. If the identifier of each node in XML tree is a string extension relative to its parent, that is the identifier for any node is a prefix of the identifiers for each of its descendants, searching for the nearest labeled descendants or ancestors involves string comparison, which is time consuming. While in our algorithm, with the 5-tuple *nodeinfo*, the determination of the descendant-ancestor relationship can be done in constant time. This also explains why the construction and the lookup of the CAM_N are faster than those of the CAM_T.

2. The CAMs for different operations are combined into one ICAM. Thus even when the lookup involves several operations, the nearest labeled ancestor and descendant for the requested node are located only once. While for CAM, this locating process has to be executed multiple times. Hence, ICAM has a better overall lookup performance comparing to CAM.

8 Conclusion

In this paper, we study the problem of accessibility control for XML data. We consider the approach of compressed accessibility maps (CAM) and propose an algorithm to integrate the optimal compressed accessibility maps for different operation types into an integrated compressed map, ICAM, with a time complexity linear to the data size. The result is a more compressed form of the accessibility map, which also consolidates the information for each user group. With the resulting ICAM, we propose an efficient lookup method to determine the accessibility of a node. This method can greatly improve the complexity of the lookup time provided that the structure of the XML data is not frequently updated. We also propose the possibility of extending the ICAM to EICAM, further reducing the lookup time complexity. The results of the experiments verify the effectiveness of the integration method and the lookup mechanism. From experiments, compared to the method in [24], up to 60% of space can be saved by the integration approach and our lookup algorithm can be more than two times faster than that of [24].

One of the future works is to combine the accessibility evaluation with the query evaluation to increase the overall performance of the XML query processing.

9 Proof of Theorem 2, 3 & 4

The following lemma about negative/positive/neutral nodes is essential to the proofs of Theorem 2, 3 and 4.

Lemma 7 *Given a node e in the XML tree, and an operation x, then the following statements hold:*

(1) If e is a negative node for x, then for any operation y, $y \geq x$, e is also negative.

(2) If e is a positive node for x, then for any operation y, $x \geq y$, e is also positive.

(3) If e is neutral for x, then for any operation y, if $y \geq x$, e is negative or neutral. If $x \geq y$, then e is positive or neutral for y.

Proof
Here we show the proof of (1). (2) and (3) can be proven similarly.

Prove by induction. If e is a leaf, then (1) is true by the definition of "cover" relation among the operations. Now suppose e is not a leaf and the lemma is true for all the children of e. Then, if e is negative for the operation x, according to Definition 1, e has more negative children than positive children for the operation x. By induction, as the lemma is true for the children of e, all the negative children should also be negative nodes for the operation y, if $y \geq x$. Then, for such an operation y, e has more negative children. By Definition 1, e is negative for the operation y. Hence, e is a negative node for all the operation y, if $y \geq x$.

Lemma 8 *Given an XML tree and an operation x, The root r is neutral for operation x. Let C_r^p be the set of nodes of CAM_x generated by labeling r as $(s+, d+)$; let C_r^n be the set of nodes of CAM_x generated* *by labeling r as $(s+, d-)$. Then, the following equations hold. (1) $C_r^p - C_r^n = N_r$; (2) $C_r^n - C_r^p = P_r$. (3) $|N_r| = |P_r|$. Here N_r is the set of all negative children of any descendant node X where the path from r to X contains only neutral nodes and the negative children of r itself, and P_r is the set of all positive children of any descendant node Y where the path from r to Y contains only neutral nodes and the positive children of r itself.*

We also call N_r the **set of negative children of neutral paths from** r. *We call P_r the* **set of positive children of neutral paths from** r.

Proof

Prove by induction. If the children of r are only positive or negative leaves, then P_r is the set of positive children of r, and N_r is the set of negative children of r. Since r is neutral, $|P_r| = |N_r|$. If $L_x(r) = (s+, d+)$, $C_r^p = N_r \cup \{root\}$. Otherwise, $C_r^n = P_r \cup \{root\}$. (1) and (2) hold obviously.

Now suppose all the neutral children of r satisfy the lemma. We denote the positive, negative, neutral children of r as P, N, and Ne, respectively, and the terminal internal child nodes of r as T. By induction, $\forall r_i \in Ne, C_{r_i}^p - C_{r_i}^n = N_{r_i}, C_{r_i}^n - C_{r_i}^p = P_{r_i}$, and $|N_{r_i}| = |P_{r_i}|$. And we also have that $P_r = (\cup_i P_{r_i}) \cup P$, and $N_r = (\cup_i N_{r_i}) \cup N$. It is obvious that $|P_r| = |N_r|$.

Then, we show that the labeling of the neutral root does not affect the subsumption of the descendants of its positive, negative children and the terminal internal nodes. In the labeling of CAMs, a positive node is labeled $(s+, d+)$, a non-leaf negative node is labeled $(s+, d-)$, while a neutral node inherits a label from a labeled ancestor. Therefore, if the root is labeled $(s+, d+)$, the positive children P will be subsumed, the negative children N will remained labeled in the CAM as $(s+, d-)$. *The positive children of P are also subsumed* and *the negative children of N are subsumed.* If the neutral root is labeled $(s+, d-)$, the positive children P will remain being labeled as $(s+, d+)$, the negative children N will be subsumed. *The positive children of P are also subsumed* and *the negative children of N are subsumed.* Since the subsumptions of the descendants depends on the labeling of the ancestors, we can see that the choice of root labeling has no impact on the subsumptions on the descendants of P and N. Note that the terminal internal nodes will be labeled in a CAM as $(s+, d-)$, no matter how their ancestors are labeled.

Among the descendants of the nodes in N, let P_n be those that are labeled in the CAM. Among the descendants of the nodes in P, let N_p be the ones labeled in the CAM. If $L_x(r) = (s+, d+)$, $C_r^p = \{r\} \cup N \cup N_p \cup P_n \cup T \cup (\cup_i C_{r_i}^p - Ne)$. If $L_x(r) = (s+, d-)$, $C_r^n = \{r\} \cup P \cup N_p \cup P_n \cup T \cup (\cup_i C_{r_i}^n - Ne)$. Then, $C_r^p - C_r^n = N \cup (\cup_i C_{r_i}^p - \cup_i C_{r_i}^n)$. By induction, $\cup_i C_{r_i}^p - \cup_i C_{r_i}^n = \cup_i N_{r_i}$. Thus we have $C_r^p - C_r^n = N \cup (\cup_i N_{r_i}) = N_r$. So (1) is true. (2) can be proved similarly.

Figures 12a and 12b illustrate the ideas above. P, N, Ne and T stand for positive, negative, neutral children and terminal internal nodes of the root, respectively (the notations beside the arrows can be ignored for now). The triangles below these nodes represent the subtrees. The shadow in a triangle or rectangle denotes the labeled nodes in the CAM. We assume that positive nodes are on the left of the negative nodes. Figures 12a and 12b show the cases when the root is labeled $(s+, d+)$ and $(s+, d-)$, respectively.

Theorem 2: Given an XML tree and atomic operations $x_1, x_2, \ldots, x_n$, $C_{x_1}, C_{x_2}, \ldots C_{x_n}$ are CAMs that label the root by Rule 1. $C'_{x_1}, C'_{x_2}, \ldots, C'_{x_n}$ are CAMs constructed in the same way except that they may not follow Rule 1. Then, the size of the ICAM generated from $C_{x_1}, C_{x_2}, \ldots, C_{x_n}$ is no larger than that from $C'_{x_1}, C'_{x_2}, \ldots, C'_{x_n}$.

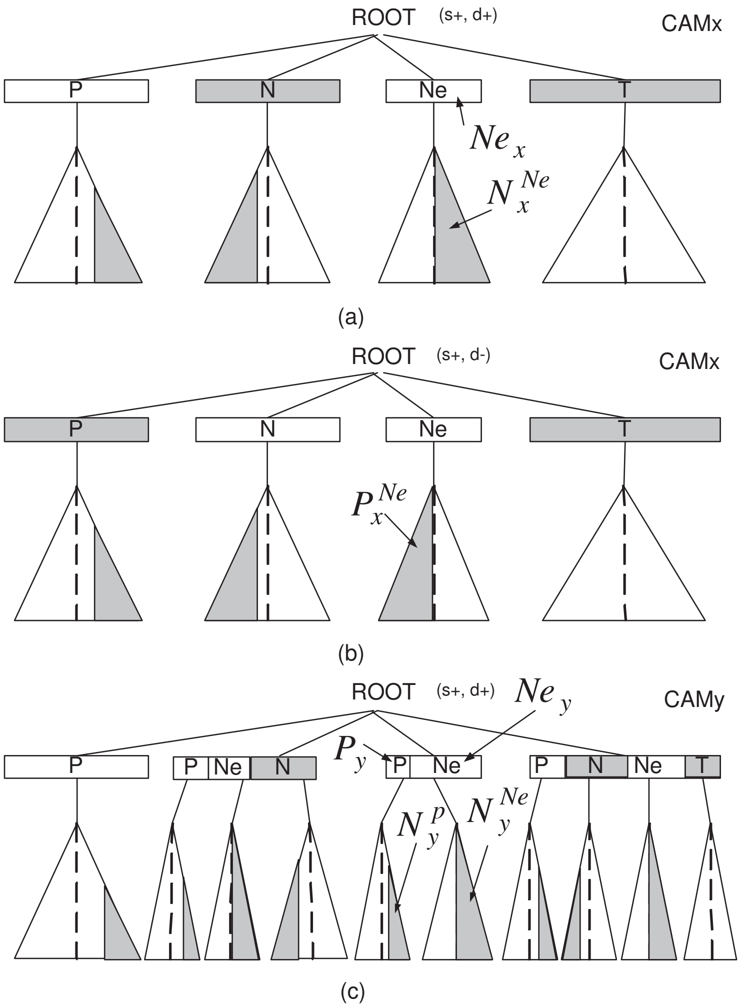


Figure 12: Choosing a favorite label for a neutral root

Proof

If Rule 1 is not followed, a neutral node of a CAM will be assigned a label arbitrarily. Suppose there is an operation x, and in the process of the labeling stage when constructing CAM_x, the root is neutral. Then, we prove here that labeling the root r by Rule 1 will not increase the size of the final ICAM. Let P_x, N_x, and Ne_x denote the positive, negative and neutral children of the root for operation x, respectively. In the proof of Lemma 8, we have shown that the labeling of the neutral root does not affect the subsumption of the descendants of P_x and N_x. but will affect the label of Ne_x, which as a result affect the subsumption state of their descendants.

From Lemma 8, the CAMs for x for $L_x(r) = (s+, d-)$ or $L_x(r) = (s+, d+)$ have the same number of nodes. Next we examine the effects of these labeling on the CAMs for other operations, which will then have effect on the ICAM. Let we denote the set of positive children of neutral paths from nodes in Ne_x as P_x^{Ne}, and the set of negative children of the neutral paths from Ne_x as N_x^{Ne}. From Lemma 8, if $L_x(r) = (s+, d-)$, P_x^{Ne} are labeled and N_x^{Ne} are not. If $L_x(r) = (s+, d+)$, N_x^{Ne} are labeled and P_x^{Ne} are not.

There are two main cases to consider.

CASE 1: There is no operation z such that $z \geq x$.

We consider any operation y, where $x \geq y$. By Lemma 7, since the root r is neutral for x, r will be neutral or positive for y. Let P_y be the set of nodes in Ne_x that are positive for y, and Ne_y are the remaining nodes in Ne_x. $Ne_x = P_y + Ne_y$. We denote the set of negative children of neutral paths from nodes in Ne_y as N_y^{Ne}.

1. If there is an operation y, where $x \geq y$, such that the root r is positive for y, then $L_y(r) = (s+, d+)$. For the descendants of P_y, let those that are labeled in CAM_y be the set N_y^p. Then, the nodes in N_y^{Ne} and N_y^p will be labeled. From Lemma 7, $N_y^{Ne} \cup N_y^p \subseteq N_x^{Ne}$.

   - (a) Suppose $L_x(r) = (s+, d+)$. The nodes in N_x^{Ne} will be labeled in CAM_x. When an ICAM is built from merging CAM_x and CAM_y, $ICAM - CAM_x$ contains no addition nodes from CAM_y that are descendants of Ne_x.
   - (b) Suppose $L_x(r) = (s+, d-)$. The nodes in P_x^{Ne} will be labeled in CAM_x, but N_x^{Ne} will be subsumed in CAM_x. When an ICAM is built from merging CAM_x and CAM_y, $ICAM - CAM_x$ contains some nodes in CAM_y which are descendants of Ne_x, these are the nodes in $N_y^{Ne} \cup N_y^p$.

   Note that the subsume states of the descendants of P_x and N_x are the same in the CAM_x for $L_x(r) = (s+, d+)$ and $L_x(r) = (s+, d-)$. Hence, the effects of additional nodes from CAM_y for descendants of P_x and N_x are identical in both cases. Thus labeling the neutral root for x as $(s+, d+)$ will ensure that the final ICAM is no larger than that from labeling x as $(s+, d-)$.

   Figure 12a, 12b and 12c help to illustrate the above arguments.

2. If for an operation y, $x \geq y$, the root remains neutral for y, we claim that assigning $L_y(r) = L_x(r)$ by Rule 1, will make the final ICAM no larger. The proof is the same as that in CASE 2.

CASE 2: There exists an operation z, such that $z \geq x$. As the CAM's are constructed in the topological order of the operations, $L_z(r)$ is determined before $L_x(r)$. We claim that assigning the same label to the root r while constructing CAM_x and CAM_z, i.e. let $L_x(r) = L_z(r)$ by Rule 1, will make the final ICAM no larger. It includes two subcases. Figure 13 illustrates these two subcases.

- (a) $L_z(r) = (s+, d+)$, $(s+, d+)$ is a better label for the root for operation x.

  If $L_z(r) = (s+, d+)$, N_z^{Ne} will be labeled in CAM_z.

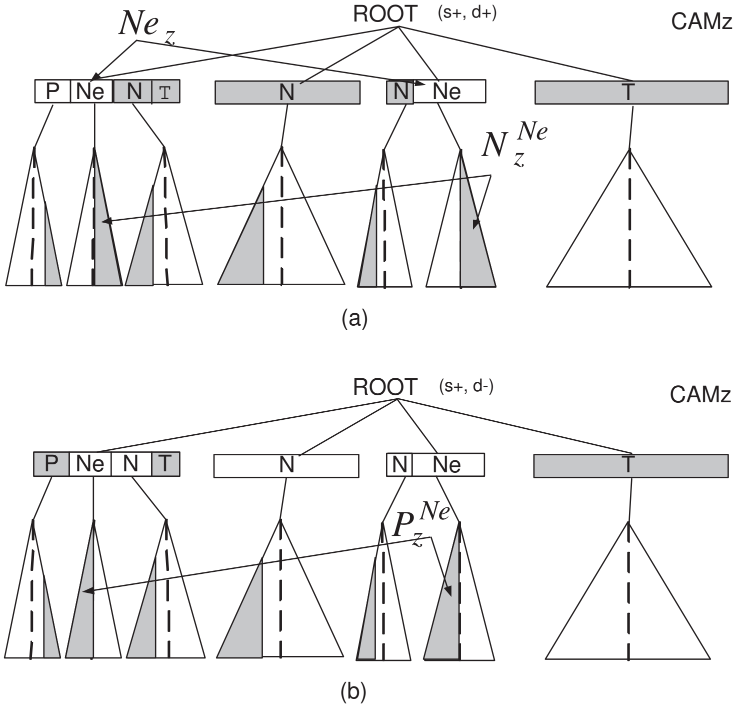


Figure 13: The labeling status of CAM_z when $L_z(r) = (s+, d+)$ and $L_z(r) = (s+, d-)$

- If $L_x(r) = (s+, d+)$, this case is similar to CASE 1 (1)(a) in the above. When an ICAM is built from merging CAM_z and CAM_x, $ICAM - CAM_z$ will contain no extra nodes from the descendants of Ne_z.
- If $L_x(r) = (s+, d-)$, then the positive children of Ne_z will be labeled in CAM_x, and the set of positive children of neutral paths from Ne_z will be labeled, these nodes are not labeled in CAM_z, and therefore when an ICAM is built from merging CAM_z and CAM_x, $ICAM - CAM_z$ contains these nodes which are descendants of Ne_z.

Thus labeling the root of x as $(s+, d+)$ results in a final ICAM no larger than a label of $(s+, d-)$. The idea of this proof can be illustrated by comparing the shadow area of Figure 13a with those in Figure 12a and 12b.

- (b) $L_z(r) = (s+, d-)$, $(s+, d-)$ is a better label for $L_x(r)$.

  (b) is the symmetric case of (a) and can be proven similarly. The idea is illustrated by Figure 13b, Figure 12a and 12b.

Theorem 3: Given n optimal CAMs, $CAM_{x_1}, CAM_{x_2}, \ldots, CAM_{x_n}$, the ICAM generated by Algorithm 1 is minimal (in the number of nodes) in that no redundant node remains.

Proof

Theorem 2 proves that the ICAM generated from the CAMs that following Rule 1 is no larger than other ICAMs. As all the labels in the ICAM are un-redundant (un-subsumed or not upward redundant) for at least one operation, no label in the ICAM can be subsumed. Thus the only possible for a label to be redundant in an ICAM is the upward redundant label. Algorithm 1 will remove such labels while merging labels by traversing the XML tree in post order. Therefore, no label is redundant for subsumption or upward redundant. Thus we claim that the ICAM generated by Algorithm 1 is a minimal one.

Theorem 4: Suppose an ICAM, C, is generated by LMR for the operations $x_1, x_2, \ldots, x_n$. Given a node e in the corresponding XML tree, its induced label for operation x_i in C is the same as its induced label in the CAM_{x_i}.

Proof

Let C be the ICAM generated from the CAMs. Suppose C_z is one of the CAMs, which is for operation z. $IL_z(e)$ is the induced label of e in C for operation z, and $L_z(e)$ is the induced label of e in C_z. Now we will prove that in the Unit-Region $IL_z(e)$ = $L_z(e)$

CASE 1: e is labeled in C, the label is $L(e) = (sx, dy)$.

1. e is labeled in C_z.
   (a) $L_z(e) = (s+, d+)$. According to LMR, for $L(e) = (sx, dy)$, $y \geq z$. Thus by LIR, $IL_z(e) = (s+, d+)$.
   (b) $L_z(e) = (s+, d-)$. There are three cases to label $L_z(e) = (s+, d-)$: e is terminal, or negative, or neutral for z. A neutral node is labeled only when it is the root. Then, Rule 1 is applied to determine the label. Please refer to CASE 1.2(b) for detail discussion.
      i. If e is a terminal node for z, then for any operation o, where $o \geq z$, there are two only possible cases. If o is permitted on e, then e is also a terminal node for o, it would be labeled as $(s+, d-)$. Otherwise, it is labeled $(s-, d-)$ for o. It is because of the obvious fact that o is permitted on none of its descendants.
      ii. If e is negative, for any operation $o \geq z$, by Lemma 7, e is also negative. Hence, label $L_o(e) = (s*, d-)$.

      Hence, for any operation $o \geq z$, $L_o(e)$ can only be $(s*, d-)$. Hence by LMR, with (sx, dy), $x \geq z$, but $\neg(y \geq z)$. Then by LIR, $IL_z(e) = (s+, d-) = L_z(e)$.
   (c) $L_z(e) = (s-, d-)$. The label means that z is not permitted at e and all its descendants, so neither is operation o with $o \geq z$. Hence by LMR, $\neg(x \geq z)$ and $\neg(y \geq z)$. Thus by LIR, we have $IL_z(e) = (s-, d-) = L_z(e)$.
2. e is not labeled in C_z, but in other CAMs.
   (a) If e obtains the label $(s+, d+)$, $(s+, d-)$ or $(s-, d-)$ because it is positive or negative for operation z. These cases have been discussed in the corresponding subcases in CASE 1.1.

(b) e is neutral and inherits the label from its labeled ancestor. By Lemma 7, for any operation $o \geq z$, any ancestor, including the root, of e can only be negative or neutral. There is no chance for the non-root ancestor to pass $d+$ to $L_z(e)$, except that when the root is also neutral for an operation. In this case, the root is possible to be labeled as $(s+, d+)$, and pass the label to its neutral descendants. Suppose operation o' is the biggest operation that covers z. (i) If the root is neutral for o', it is neutral for any operation o, $o' \geq o$ (including z). According to Rule 1, $L_o(root) = (s+, d+)$. In this case, when constructing C_o, e will at last inherit the label $(s+, d+)$. By LMR, since $L(e) = (sx, dy)$, $y \geq z$. Thus by LIR, $IL_z(e) = (s+, d+) = L_z(e)$.

(ii) If the root is negative for o', $L_{o'}(root) = (s+, d-)$. Then, according to Rule 1, all the neutral root for operation o, $o' \geq o$ (including z), will be labeled as $(s+, d-)$. Then, $L_z(e) = (s+, d-)$ in this case. And for any operation $o \geq z$, $L_o(e) = (s+, d-)$. Hence, $x \geq z$ and $\neg(y \geq z)$ Then by LIR, $IL_z(e) = L_z(e) = (s+, d-)$.

CASE 2: e is not labeled in C. Then, e should be redundant in C_z. Let f_z and f be the nearest labeled ancestor of e in C_z and C, respectively, if any. Suppose $L(f) = (sx, dy)$. There are two possible cases we should consider.

1. $f_z = f$, i.e. after labeling merging, f_z is still the nearest labeled ancestor of e in C.

   (a) If $L_z(f) = (s+, d+)$, $L_z(e) = (s+, d+)$ by Definition 3. By LMR, given $L(f) = (sx, dy)$, $y \geq z$. Thus by ILR, $IL_z(e) = L_z(e) = (s+, d+)$.

   (b) If $L_z(f) = (s+, d-)$, we consider two cases, e has a labeled descendant with $(s+, d*)$ in C_z; or e has no labeled descendants or all the descendants are labeled with $(s-, d-)$ in C_z. Now we consider the first case, and let d_z be such a nearest descendant. According to Definition 3, we have $L_z(e) = (s+, d-)$. By LMR, $L_z(f) = (s+, d-)$ implies $x \geq z$ and $\neg(y \geq z)$.

   There are two cases. (i) After label merging, d_z is still the nearest labeled descendant of e, except that its label is now (sm, dn). (see Figure 14a). By LMR, $L_z(d_z) = (s+, d*)$ implies $(m \geq z)$, Hence by LIR, $IL_z(e) = (s+, d-) = L_z(e)$. (ii) After label merging, d, an ancestor of d_z, becomes e's nearest labeled descendant with a label (sm, dn) (See Figure 14b). It follows that $L_z(d) = (s+, d*)$ from $L_z(d_z) = (s+, d*)$ and the assumption of the unit region (i.e. once an operation is permitted at a node it will also be permitted at its ancestors). Hence, $m \geq z$, implying $IL_z(e) = (s+, d-) = L_z(e)$.

   If e has no labeled descendants or all its descendants are labeled $(s-, d-)$ in C_z. Then, all the descendants of e should have the label $(s-, d-)$ in the labeling step. Hence, after the label merging, no descendant will have a label (sm, dn) with $m \geq z$. Thus by LIR, we have $IL_z(e) = (s-, d-) = L_z(e)$.

   (c) If $L_z(f) = (s-, d-)$, $L_z(e) = (s-, d-)$. On the other hand, $L_z(f) = (s-, d-)$ implies $\neg(x \geq z)$ and $\neg(y \geq z)$. Hence by LIR, $IL_z(e) = (s-, d-) = L_z(e)$.

2. f is an descendant of f_z (See Figure 15). f is labeled in C, and let $L(f) = (sx, dy)$. We have proven in CASE 1.2, $IL_z(f) = L_z(f)$. We consider the following three cases.

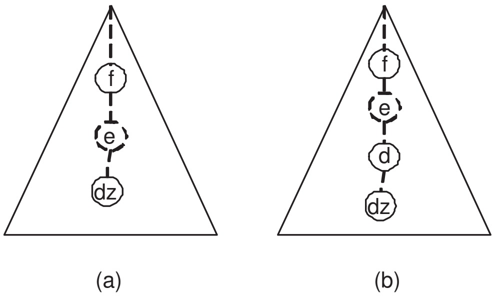


Figure 14: Two cases about the labeled descendant of e

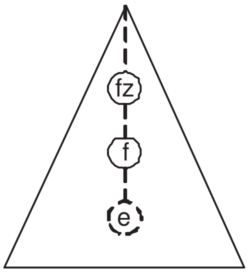


Figure 15: Descendant of f_z becomes e's nearest labeled ancestor in the ICAM

(a) If $L_z(f_z) = (s+, d+)$, $L_z(e) = (s+, d+)$ and $L_z(f) = (s+, d+)$ by Definition 3. By LMR, $y \geq z$. Hence by LIR, $IL_z(e) = (s+, d+)$.

(b) If $L_z(f_z) = (s+, d-)$, $L_z(e)$ and $L_z(f)$ can only be $(s+, d-)$ or $(s-, d-)$.

- If e has an nearest labeled descendant, d_z, in C_z with $(s+, d*)$, d_z should also be one of the nearest labeled descendant of f. Thus $L_z(e) = L_z(f) = (s+, d-)$ by Definition 3. And by LMR, if $L_z(f) = (s+, d-)$, $x \geq z$ and $\neg(y \geq z)$. Based on the similar arguments in CASE 2.1.b, after label merging, e must have some labeled descendant, d, which can be d_z or one of its ancestors, labeled (sm, dn) in C, with $m \geq z$. Then by LIR, $IL_z(e) = (s+, d-) = L_z(e)$.
- If e has no labeled descendants or all the descendants are labeled $(s-, d-)$, $L_z(e) = (s-, d-)$. Then, no descendant of e will have a label (sm, dn) with $m \geq z$. Then, if f has a labeled descendant with $(s+, d*)$, which leads to $x \geq z$ and $\neg(y \geq z)$, by LIR, $IL_z(e) = (s-, d-) = L_z(e)$. Else if $L_z(f) = (s-, d-)$, by LMR, $\neg(y \geq z)$ and $\neg(x \geq z)$. Hence by LIR, $IL_z(e) = (s-, d-) = L_z(e)$.

(c) If $L_z(f_z) = (s-, d-)$, $L_z(e) = L_z(f) = (s-, d-)$ by Definition 3. By LMR, $\neg(x \geq z)$ and $\neg(y \geq z)$. Thus by LIR, $IL_z(e) = (s-, d-) = L_z(e)$.

3. e has no labeled ancestor in C_z, but f is its nearest labeled ancestor in C. The only possible for this case is that e and all its ancestors are upward redundant in C_z, and their induced label is $(s+, d-)$ by Definition 3. Thus $L_z(f) = (s+, d-)$, implying $x \geq z$ and $\neg(y \geq z)$ by LMR. As e is upward redundant for z, e should have an accessible proper descendant in C_z. As proven in

CASE 2.1.b, e should have an descendant labeled with (sm, dn), $m \geq z$, in C. Hence by LIR, $IL_z(e) = (s+, d-) = L_z(e)$.

4. e has no labeled ancestor in C. That means e and all its ancestors are upward redundant for every operation. Thus by Definition 3, $L_z(e) = (s+, d-)$. And by LIR, $IL_z(e) = (s+, d-)$. Hence, $IL_z(e) = L_z(e)$.

Lemma 2: The accessibilities of the nodes in an XML tree can be correctly induced from the ICAM generated by Algorithm 1.

Proof
We have proven in Theorem 4 that the induced label of a node for an operation can be correctly induced from the ICAM generated by LMR. We only need to prove here that after removing the upward redundant labels, the accessibility of a node, i.e. the 's' part of a label, can still be correctly induced. Let C denotes the ICAM by LMR, and C' be the final ICAM by removing all the upward redundant labels from C. And let C_z be the CAM for operation z.

For a node to have an upward redundant label, it should have some accessible proper descendants for every operation, and at least one of these descendants must be labeled in C. Thus the 's' part of the induced label of this node is '+' in C_z and C', implying the correct accessibility.

Removing an upward redundant label, say $L(f)$, will affect the induced label of other nodes only when those nodes are not labeled in C, and f is their nearest labeled ancestor in C, or f is their only accessible nearest labeled descendant in terms of an operation in C.

1. Let e be the node whose nearest labeled ancestor is f in C. e is upward redundant itself, otherwise e should be labeled. Hence, e must have an proper accessible descendant, implying $L_z(e) = (s+, d*)$ for any operation z. After deleting $L(f)$, an ancestor of f, say f', becomes the nearest labeled ancestor of e in C'. Let $L(f') = (sx, dy)$. We know that $L_z(f') = (s+, d*)$ for any operation z. By LMR, $x \geq z$. Hence, $IL_z(e) = (s+, d*)$ in C'. Thus the accessibility of e can still be correctly induced.

2. For an operation z, let e' be the node whose only accessible nearest labeled descendant is f. That is $L_z(f) = (sm, dn)$, $m \geq z$ by LMR. For f is upward redundant, f should have an proper accessible descendant for every operation. Thus one of such descendants could be the new nearest labeled descendant of e'. Thus the induced label of e' remains the same, implying that its accessibility can still be correctly induced.

For other nodes, their induced labels are the same as those from C, which have been proven to be correct in Theorem 4. Hence, their accessibilities can be induced correctly.

Acknowledgment

The authors thank Ting Yu for providing valuable assistance in the experimental work and advice. They thank the reviewers for very constructive suggestions and helpful comments. They thank Zhixiang Chen for a thorough review of the final draft. This research is supported by the Hong Kong RGC Earmarked Grant UGC REF.CUHK 4179/01E and Hong Kong ITF Grant REF. ITS/069/03.

References

[1] Document object model(DOM) level 1 specification, version 1.0, W3C recommendation 1, http://www.w3.org/tr/rec-dom-level-1, Octorber 1998.

[2] B. Kolman and R.C. Busby. *Discrete Mathematical Structures for Computer Science*. Prentice-Hall International Editions, 2nd edition, 1987.

[3] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Controlled access and dissemination of XML documents. In *Proceedings of the 2nd International Workshop on Web Information and Data Management*, November 1999.

[4] E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5:3:290–331, August 2002.

[5] E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 9:85–101, January 1997.

[6] P. Bird. Implementing low level access control with DB2 UDB. *The IDUG Solutions Journal*, 7:3, 2000.

[7] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing XML documents. In *Proceedings of International Conference on Extending Database Technology*, pages 121–135, March 2000.

[8] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 5:169–202, May 2002.

[9] A.L. Diaz and D. Lovell. XML generator, www.alphaworks.ibm.com/tech/xmlgenerator/, September 1999.

[10] R. Fagin. On an authorization mechanism. *ACM Transactions on Database Systems*, 3:310–319, September 1978.

[11] W. Fan, C. Chan, and M. Garofalakis. Secure XML querying with security views. In *Proceedings of ACM International Conference on Management of Data*, pages 587–598, 2004.

[12] I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. In *Proceedings of ACM Symposium on Access Control Models and Technologies*, pages 61–69, 2004.

[13] N. Gal-Oz, E. Gudes, and E. B. Fernandez. A model of methods access authorization in object-oriented databases. In *Proceedings of VLDB Conference*, 1993.

[14] P. Griffiths and B. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1:242–255, September 1976.

[15] Michiharu Kudo and Satoshi Hada. XML document security based on provisional authorization. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 87–96, November 2000.

[16] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *Proceedings of the VLDB Conference*, 2001.

[17] M. Jiang and A. Fu. Efficient accessibility lookup for XML. In *International Conference on Databases and Applications (DBA2003)*, February 2003.

[18] M. Murata, A. Tozawa, M. Kudo, and S. Hada. XML access control using static analysis. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 73–84, 2003.

[19] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Transactions on Database Systems*, 16:88–131, March 1991.

[20] F. Rabitti, D. Woelk, and W. Kim. A model of authorization for object-oriented and semantic databases. In *Proceedings of International Conference on Extending Database Technology*, March 1988.

[21] D. Raphaely and et al. Chapter 12, establishing security policies. In *Oracle8i Application Developer's Guide - Fundamentals Release 8.1.5*, February 1999.

[22] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2002.

[23] M. Winslett, K. Smith, and X. Qian. Formal query languages for secure relational databases. In *ACM Transactions on Database Systems*, volume 19, pages 626–662, December 1994.

[24] T. Yu, D. Srivastava, L.V.S. Lakshmanan, and H.V. Jagadish. Compressed accessibility map: Efficient access control for XML. In *Proceedings of VLDB Conference*, pages 478–489, 2002.