Authorization-Based Access Control for XML

Yaron Kanza, Alberto O. Mendelzon, Renée J. Miller and Zheng Zhang

Department of Computer Science
University of Toronto,
Toronto, Canada
{yaron, mendel, miller, zhzhang}@cs.toronto.edu

Abstract. Fine-grained access control for XML supports the specification and enforcement of security policies on elements rather than at the level of the entire document. Parameterized access control allows modifying the security according to the user and the environment. Traditionally, views are used in database system to support fine-grained and parameterized access control. But over XML, views are complicated, lack an efficient maintenance mechanism and suffer from the absence of a comprehensive query-rewriting set of rules. This work presents an authorization mechanism in which a security policy for XML is specified as parameterized rules formulated using XPath. The rules specify relationships, between elements, that should be concealed from users. The access control mechanism authorizes only *valid queries*, i.e., queries that do not disclose information on concealed relationships. Algorithms that check the validity of queries are presented. The problem of how to create a *coherent* set of rules is addressed and solved. It is shown that the proposed access control mechanism can either serve as a substitute for views or as a layer that verifies the integrity of views.

1 Introduction

Access control is a fundamental part of databases systems. The purpose of access control is to protect secured data and personal information, from unauthorized users. It has been recognized many years ago that access control should be an integral part of database systems, and nowadays, when computer systems are often connected to the Internet, the need of access access control has further increased. Recently, database systems have started supporting XML. Apparently, access control for XML should be applied in these systems.

Access control protects secured data by putting limitations on access to data items. The granularity of an access control determines the specificity of limitations that can be enforced. In a *fined-grained* access control, enforcement of security and privacy policies are on the level of the smallest possible items, i.e., tuples rather than relations in relational systems, and elements rather than documents in XML.

A *parameterized* access control enables a security policy in which access to data entities is granted based on parameters of the user and the environment. In a parameterized access control, different users may receive different access permissions. For example, in a grade system of a university, a student may be authorized to see only her grades, while a course teacher is authorized to see all the grades in the courses that she teaches.

The need, for fine-grained and parameterized access control, has been recognized in relational database systems. Thus, in recent years the major database vendors have begun to incorporate in their systems mechanisms to support fine-grained access control. The Virtual Private Database of Oracle and the Mandatory Access Control implementation of IBM are two examples [22]. Nonetheless, these mechanisms are defined for the relational model and are not applicable for XML.

There are two basic approaches to access control, referred as the *Truman model* and the *Non-Truman model* in [21]. In the Truman security model, each user is provided with a personal view over the complete database. User queries are either executed over the views or modify by the system so that the user does not get to see data other than those exposed by the view. One of the drawbacks of the Truman model is that a user cannot tell if an answer to a query is correct with respect to (w.r.t.) the entire database or that it is only correct w.r.t. the data she is authorized to see, i.e., w.r.t. the view.

In the Non-Truman model, a query is formulated against the database relations without referring to views or to any other authorization specification. The system puts each query through a validity test. If the query passes the test, it is authorized and executed over the whole database, without any modifications. Else, the query is rejected and execution is denied.

In this paper, we present a rule-based access-control mechanism for systems that support XML and XPath queries [4]. Our mechanism is fine grained, parameterized and complies with the Non-Truman model. In our mechanism, access-control is specified by a set of security rules. Each rule consists of a pair of parameterized XPath expressions. For a given XML document, a pair of expressions in a rule defines pairs of elements, in the document, such that the relationship between these elements should be *concealed* from unauthorized users, as explained next.

XML documents are modeled as labeled graphs and two elements are considered related if they are connected by a path. Concealing from a user U a relationship between a pair of elements e_1 and e_2, means that the information provided to U, as answer to queries for example, is such that U cannot infer if e_1 and e_2 are related, i.e., connected by a path. The next example further explains the notion of concealing a relationship.

Consider an XML document D that contains information on departments and employees in a company. We assume that in D there is a path from each department element d to an employee element e whenever e works in d. Suppose that the security policy of the company does not allow a user U revealing a pair of employee e and department d such that e works in d. In this case, we say that the access-control system conceals the relationship between the elements e and d if the following hold. There are two documents D_1 and D_2, that contain the same elements as D but may have different edges than D. In D_1 there is a path from

d to e, and in D_2 there is no path from d to e. Furthermore, for every authorized query Q holds $Q(D_1) = Q(D_2)$. That is, from the answers to authorized queries, U cannot infer whether the employee e belongs to the department d.

A query Q discloses secrete information if it is possible to infer from the answer to the query the existence or non-existence of a concealed relationship. A stronger notation of information disclosure, based on probability, has been suggested in [17], for the relational model. In their work, an information disclosure means exposing information that increases the probability to guess that secret data. There are cases, however, where rejecting a query just because its answer increases the probability for guessing the concealed data is too restrictive. For example, consider a database of students, courses and grades. A query that returns all the grades that were given in a certain course C increases the chance that one will be able to correctly guess the grade of a certain student S in C. Yet, it is reasonable to assume that revealing the grades in C does not reveal the grade of S in C, unless all the students of C received the same grade.

In our security mechanism, protecting secured data is according to the Non-Truman approach. Intuitively, a query is considered *locally valid* w.r.t. a document and a set of rules, if the answer to the query does not depend on edges (or paths) in the document that are concealed by the rules. Queries are allowed to be executed only if they are locally valid.

In many cases, several XML documents in a repository conform to a single schema, either a DTD or an XSchema [25]. In such cases, being able to test the validity of queries, w.r.t. a set of rules and the schema, is useful. A query is considered *globally valid* w.r.t. a set of rules and a schema if the query is locally valid w.r.t. the rules and any document that conforms to the schema.

Traditionally, views are being used for access control, either in the Truman model or in the Non-Truman model. There are three ways of using views in the Truman model. First, keeping instances of the views, maintaining the instances whenever an update of a base relation occurs and computing queries over these instances. This option is expensive when updates are frequent. Furthermore, it is not suitable for a parameterized access control. A second way is of computing the relevant views whenever a query is posed. This option is expensive when queries frequently posed, since it limits the possibility to optimize queries. The third option is of query rewriting; however, query rewriting is limited, has high complexity and cannot always be applied, especially for XML query languages.

For the Non-Truman model, a parameterized and fined-grained mechanism, of access control in relational system, as been proposed in [21]. Their mechanism is based on using *authorization views.* The authorization views are instantiated over a database D by fixing the parameters of the views according to D. A query Q is considered valid, over a database D, if there is a query Q' that is a rewriting of Q, using only the instantiated authorization views, and is equivalent to Q. Execution is authorized for merely valid queries. The approach of using authorization views has the benefit of *authorization transparency* where queries can be formulated against the database without having to refer to the authorization views. Other authorization transparent models have been proposed in [18, 23, 24]. All these models were proposed for relational systems and are not applicable to XML.

Our work follows the work of [21], trying to adapt their approach to XML. But for XML query languages, query rewriting is limited and has high time complexity. Using rules instead of views relief from the need of query rewriting and thus, allows an access-control mechanism that is efficient and comprehensive, i.e., capable of dealing with a variety of queries.

Several papers were dealing with access control for XML. In [7], it has been proposed to use security views and query rewriting of XPath expressions. Their security views are specified by extending a DTD with annotations. For each production $A \rightarrow \alpha$ in the DTD, the annotation specifies to which elements of α access is permitted. Annotations can be parameterized and thus the proposed access-control mechanism is fine-grained, parameterized and complies with the Truman model. Our approach different from their approach in complying to the Non-Truman model and in using rules instead of views.

Many papers proposed access control for XML that is based on restricting access to nodes [1, 5, 9, 19]. This approach has also been used in XACML [10] and XACL [14], two proposed industrial standards for XML access control. See a comparison of these proposals in [8].

Our approach of concealing relationships provides a finer granularity than restricting access to nodes. On one hand, restricting access to a node is possible in our approach. This is done by concealing all the relationships (edges and paths) to that node. On the other hand, in our approach it is possible to conceal a relationship with out restricting access to any of the nodes in the relationship. For example, consider the document in Figure 1 that contains information on courses, student and grades. Using rules, it is possible to conceal relationships between students and grades without concealing relationships between courses and grades. This suits a policy where users are not allowed to see student grades but seeing the grades given in each course is permitted.

Our main contribution in this paper is by providing a simple mechanism for parameterized fine grained access control, for XML, in the Non-Truman model. We show how to test validity of queries, how to construct *coherent* sets of rules, i.e., sets where rules indeed conceal the relationships that they suppose to conceal. Finally, we discuss conditions that when hold it is provided that unauthorized users will not be able to view or infer concealed relationships.

The outline of the paper is as follows. In Section 2, document, schema and query are formally defined. In Section 3, access-control rules are presented and their semantics is defined. Section 4 presents tests for checking local and global validity of queries. In Section 5, we define the notion of a coherent set of rules. We provide conditions for testing that a set of rules is coherent. These conditions can also help users in creating coherent rule sets. Section 6, deals with concealment of relationships. In particular it is shown that when using a coherent set of rules (with some additional technical conditions), concealment of the secret relationships is guaranteed. Finally, in Section 7, we discuss two possible usages of security rules and conclude.

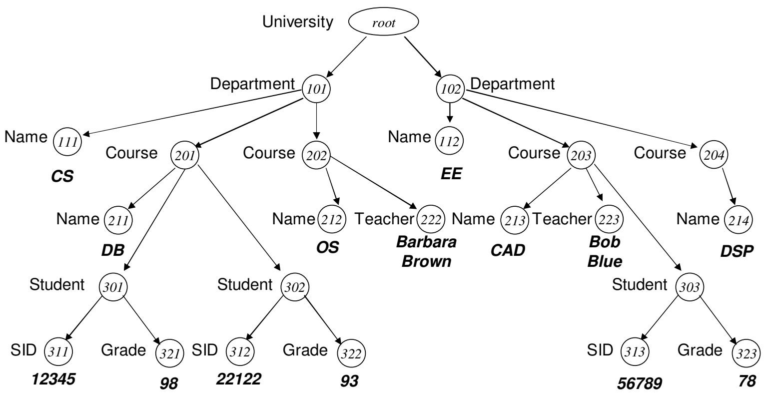

Fig. 1. A document that contains information on courses, students and grades in a university.

2 Preliminaries

In this section we provide some preliminary definitions. We formally define a document, a schema and a query. In our model, all three are represented using a rooted labeled directed graph.

2.1 Document

We represent an XML document as a rooted labeled directed graph. Each node, in the graph, is an *XML element* and has an identifier (*eid*). Values are attached to *atomic nodes*[1] (i.e., nodes without outgoing edges). For simplicity, we assume that all the atomic nodes are of type PCDATA (i.e., *String*). Nodes that have outgoing edges are *complex elements*. A document has a *root* and every node in the document is reachable from the root. Note that for XML documents that do not contain references the graph is a tree. When references are allowed, the graph may include cycles.

Formally, let L be a finite set of labels and A be a finite set of atomic values. A *document* D is a 5-tuple $(X, E_D, root_D, \textit{label-of}_D, \textit{value-of}_D)$, where X is a finite set of elements, E_D is a set of edges over X, $root_D$ is the document root, $\textit{label-of}_D$ is a function that maps each element of X to a label of L and $\textit{value-of}_D$ is a function that maps each atomic node to a value of A. Given an element e

[1] In XML, there is a distinction between elements and attributes. We ignore this distinction in order to simplify the model.

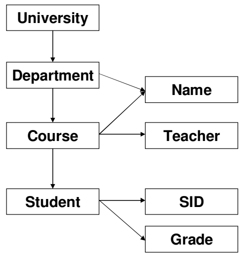

Fig. 2. A schema for documents that contain information on courses, student and grades.

in X, the label $label\text{-}of_D(e)$ is called the *element name* of e. The *skeleton* of D is the 4-tuple $(X, E_D, root_D, label\text{-}of_D)$.

Example 1. Figure 1 depicts a document that contains information on courses, students and grades. Elements are presented by circles. The eid's are written inside the circles. Atomic values appear below the atomic nodes and written with a bold font.

2.2 Schema

A *schema* is a rooted labeled directed graph that can be used to constrain the structure of a document or to provide a succinct description of the document structure. Each node of a schema represents an *element name* and has a label. The element name l that the node represents is also the label of this node. In a schema, nodes are pairwise disjoint by their labels. That is, each pair of nodes is mapped to two different labels.

Let L be a finite set of labels. Formally, a schema S over L is a 4-tuple $(Names_S, E_S, root_S, label\text{-}of_S)$ where $Names_S$ is a set of nodes called *element names*, E_S is a set edges over $Names_S$, $root_S$ is the root of the graph and $label\text{-}of_S$ is a function that maps the element names of $Names_S$ to labels of L. Schema nodes are uniquely labeled. That is, for each pair of nodes $e_1 \neq e_2$ in $Names_S$, holds $label\text{-}of_S(e_1) \neq label\text{-}of_S(e_2)$.

A document *conforms* to a schema if there exists a *homomorphism* from the skeleton of the document to the schema. Graph homomorphism is defined as follows. Let $G_1 = (V_1, E_1, root_{G_1}, label\text{-}of_{G_1})$ and $G_2 = (V_2, E_2, root_{G_2}, label\text{-}of_{G_2})$ be two rooted labeled directed graphs. A homomorphism from G_1 to G_2 is a mapping $h : V_1 \rightarrow V_2$ such that the following three conditions hold. *(1)* The root is mapped to the root: $h(root_{G_1}) = root_{G_2}$. *(2)* Labels are preserved: if $h(v_1) = v_2$ then $label\text{-}of_{G_1}(v_1) = label\text{-}of_{G_2}(v_2)$. *(3)* Edges are preserved: if G_1 has an edge from v_1 to u_1 then G_2 has an edge from $h(v_1)$ to $h(v_2)$.

An example of a schema is given in Figure 2. The document that is presented in Figure 1 conforms to this schema.

Note that testing whether a given document D conforms to a given schema S does not require a check of all the possible mappings from X to $Names_S$. This is because the nodes of S are uniquely labeled and in a homomorphism labels are preserved. Indeed, only a single mapping should be considered when checking if a document conforms to a schema.

2.3 Queries

We use XPath [4] for formulating queries. XPath is a simple language for navigating in an XML document. XPath queries are omnipresent in XML applications. In particular, XPath is part of both XSLT [3] and XQuery [2], the WWW Consortium standards for querying and transforming XML.

Syntax We consider, in this work, a class $\mathcal{C}_{basic}$ of *basic* XPath queries. The class $\mathcal{C}_{basic}$ contains the expressions that are defined by the following grammar.

$$\begin{aligned}
path &:= step \mid path\ step \\
step &:= axis\ test \mid axis\ test\,[cond] \\
axis &:= / \mid // \\
test &:= label \mid * \\
cond &:= path \mid path\ comp\ const \mid \\
&\quad\ (cond) \mid \textbf{not}\ cond \mid \\
&\quad\ cond\ \textbf{and}\ cond \mid cond\ \textbf{or}\ cond
\end{aligned}$$

In the grammar rules, *label* is an element name, * is a wildcard that, in principle, can replace any label. The '/' and '//' are called *child axis* and *descendent axis*[2], respectively. A part of an expression that appears in square brackets (i.e., in [ ]) is called *condition.*[3] In the grammar rule of condition (i.e., of *cond*), *comp* represents a comparison operator such as $=, \neq, <, \leq, >, \geq$ and *const* stands for a constant value.

We now provide some basic definitions and then describe the semantics of queries.

Query Pattern and Query Skeleton A *path query* is a query that does not contain any condition in it. That is, a path query is defined by the grammar of $\mathcal{C}_{basic}$ except that the rule for *step* (the second rule) contains only the *axis test* option.

[2] To simplify our model, // is considered a descendent axis rather than a descendent-or-self axis.

[3] We allow only relative path expressions in a condition. In the XPath standard, a relative expression should start with a label or with a dot symbol. We omit the dot, to simplify the grammar.

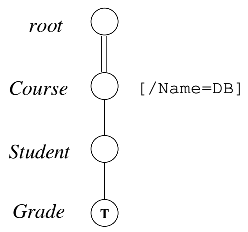

Fig. 3. The pattern of the query `//Course[/Name='DB']/Student/Grade`.

A path query has a representation as a rooted label directed graph that has the form of a path. This graph is called the *query pattern*.

Consider a path query Q. The query pattern P of Q is the following graph. For each appearance of a label l in Q, P has a node v labeled with l and we say that v *represents* this appearance of l. Note that if l appears n times in Q then there are n nodes with label l in P. In addition to the nodes that represent labels, P has a root node, denoted $root_P$, with the label *root* attached to it.

The edges of P are either *child edges* or *descendent edges*. For each fragment in Q of the form l_1/l_2, where l_1 and l_2 are labels, P has a child edge from the node that represents l_1 to the node that represents l_2. For a fragment $l_1//l_2$, P has a descendent edge from the node that represents l_1 to the node that represents l_2. If Q begins with $/l$ then there is a child edge from the root to the node that represents l. If Q begins with $//l$, then there is a descendent edge from the root to the node that represents l.

The rightmost label in a query is represented by a node that has no outgoing edge in the query pattern. This node is called the *target* of the pattern.

In the above definition of P, a wildcard in the query is treated in exactly the same way as a label.

When the query contains conditions, the query pattern is a graph with conditions attached to its nodes. Thus, if Q contains a fragment of the form $l[C]$, then the condition C is attached to the node that represents l in P.

Example 2. The pattern of the query `//Course[/Name='DB']/Student/Grade` is depicted in Figure 3. Child edges are drawn with a single line and descendent edges are drawn with a double line. The target node is marked by the letter 'T'. A condition is attached to the node with the label "Course". The condition appears inside square parentheses.

Formally, a query pattern P is a 6-tuple $(V, E_P, root_P, target_P, label\text{-}of_P, cond_P)$ such that V is a set of variables, which are also the nodes of P; $E_P = (E_c, E_d)$ is

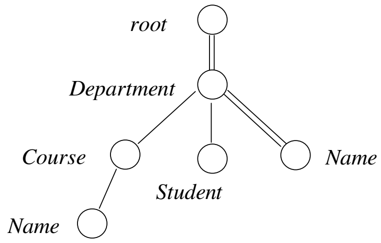

Fig. 4. The skeleton of the XPath expression `//Department[/Course/Name='DB' or //Name='CAD']/Student`.

a set of edges over V, where E_c is a set of child edges and E_d is a set of descendent edges; $root_P \in V$ is the root; $target_P \in V$ is the target node; $label\text{-}of_P$ is a function that maps the nodes of P to labels (for the root holds $label\text{-}of_P(root_P) = root$) and $cond_P$ maps some of the nodes to strings that represent a condition according to the XPath grammar. The nodes V and the edges E_P form a rooted directed path whose root is the node $root_P$.

We now define the *skeleton* of a query. In Section 4.2, we use skeletons for testing validity of queries. The skeleton of a query Q is a rooted labeled directed tree, denoted $\sharp(Q)$, that is created by *unfolding* the conditions in Q. The unfolding is a recursive operation. First, the skeletons of paths expressions in the conditions of Q are constructed by creating their pattern and applying unfolding to the conditions in the pattern. Then, the result tree is being connected to the query pattern of Q. We now describe this process in more detail.

Suppose that v is a node in the pattern P of Q and let C be the condition attached to v. In the unfolding, all the path expressions in C are extracted. This is done by erasing from C all the logical operators (`and`, `or` and `not`), parenthesis and condition fragments of the form θc, where θ is a comparison operator and c is a constant.

Suppose that $p_1, \ldots, p_k$ are the path expressions that are extracted from C. Then, for each such path p_i, the skeleton of P_i is created by a recursive call, i.e., the pattern of p_i is created and unfolding is applied to the conditions of p_i. Let T_i be the skeleton tree created from p_i, then T_i is being connected to the pattern P by replacing the root of T_i with the node v of P. That is, if e_i is the child of $root_{T_i}$, then the node $root_{T_i}$ and the edge $(root_{T_i}, e_i)$ are being removed from T_i. All the other nodes and edges of T_i are added to P. In addition, an edge (v, e_i) that connects P and T_i is added to P. The edge (v, e_i) is a child edge if $(root_{T_i}, e_i)$ is a child edge and a descendent edge otherwise.

The target node of P is considered the target node of the skeleton $\sharp(Q)$. Note that $\sharp(Q)$ contains all the nodes and edges of the pattern P of Q.

Example 3. The skeleton of the query `//Department[/Course/Name='DB' or`

`//Name='CAD']/Student` is depicted in Figure 4. The target of the skeleton is the node with the label "Student".

Given a query Q, the skeleton of Q is represented as a 4-tuple $\sharp(Q) = (V, E, \mathit{root}_{\sharp(Q)}, \mathit{label\text{-}of}_{\sharp(Q)})$, where V is a set nodes; $E = (E_c, E_d)$ is the edge set and consists of child edges (E_c) and descendent edges (E_d); $\mathit{root}_{\sharp(Q)}$ is the root and $\mathit{label\text{-}of}_{\sharp(Q)}$ is a function that maps the nodes to labels.

Semantics We now formally define the semantics of queries in $\mathcal{C}_{basic}$. The semantics is defined using the notion of *matching* a query pattern to the *transitive closure* of a document. Our definition of the semantics coincides with the known semantics of XPath and is presented in a way that will support later definitions.

First, we define the semantics of path queries and then extend the definition for other queries of $\mathcal{C}_{basic}$.

We start by defining the notion of matching a path query Q to a *document expansion.* An expansion of a document D is a graph that is created by adding to D *descendent edges* and a new root node. In the document expansion, there is a child edge from the new root to the root of D. The new root has the label *root* attached to it.

Let $D = (X, E_D, \mathit{root}_D, \mathit{label\text{-}of}_D, \mathit{value\text{-}of}_D)$ be a document. The expansion of D is a 3-tuple $D^{//} = (D, E_D^{//}, \mathit{root}_{D^{//}})$, where $E_D^{//}$ is the set of descendent edges that are added to D and $\mathit{root}_{D^{//}}$ is the new root.

Let $P = (V, E_P, \mathit{root}_P, \mathit{target}_P, \mathit{label\text{-}of}_P, \mathit{cond}_P)$ be a pattern of a path query Q. A *matching* of P w.r.t. $D^{//}$ is a mapping $h : V \to X$, from the variables of P to the elements of D, such that the following three requirements hold. *(1)* The root is mapped to the root: $h(\mathit{root}_P) = \mathit{root}_{D^{//}}$. *(2)* Labels are preserved: for every $v \neq \mathit{root}_{D^{//}}$ holds $\mathit{label\text{-}of}_D(h(v)) = \mathit{label\text{-}of}_P(v)$ or $\mathit{label\text{-}of}_P(v) = *$. *(3)* Child edges and descendent edges are preserved: if P has a child edge from v_1 to v_2 then there is a child edge in $E_D \cup \{(\mathit{root}_{D^{//}}, \mathit{root}_D)\}$ from $h(v_1)$ to $h(v_2)$. If P has a descendent edge from v_1 to v_2 then there is a descendent edge in $E_D^{//}$ from $h(v_1)$ to $h(v_2)$.

Let M be the set of the matchings of P w.r.t. the document expansion $D^{//}$. Consider the set of elements e such that there is a matching μ in M that maps the target node of P to e. This set is called the *evaluation result* of Q w.r.t. $D^{//}$. Note that the evaluation result can be viewed either as a set of eid's or as a set of document fragments.

The transitive closure of a document D is a document expansion that is created by adding a descendent edge between each two nodes that are connected by a path in D. The transitive closure of D is denoted $\bar{D}$. Formally, the transitive closure of a document $D = (X, E_D, \mathit{root}_D, \mathit{label\text{-}of}_D, \mathit{value\text{-}of}_D)$ is a document expansion $\bar{D} = (D, E_{\bar{D}}^{//}, \mathit{root}_{\bar{D}})$, where the set of transitive edges $E_{\bar{D}}^{//}$ consist of the following pair of nodes. For each two nodes n_1 and n_2 in X, such that n_2 is a descendent of n_1, $E_{\bar{D}}^{//}$ contains the pair (n_1, n_2). In addition, for each node n in X, $E_{\bar{D}}^{//}$ contains the pair $(\mathit{root}_{\bar{D}}, n)$.

The *answer* to a path query Q, when posed to a document D, is the evaluation result of Q w.r.t. the transitive closure $\bar{D}$.

We now extend the definition of the semantics to queries that contain conditions. When a query contain conditions, a matching of a pattern P to a document expansion $D^{//}$ should satisfy a fourth requirement, in addition to the three requirements mentioned above: *(4)* Conditions on nodes are satisfied: suppose that $cond_P(v) = C$, for some node v in P, then the condition C is satisfied w.r.t. the element $h(v)$. Satisfaction of conditions is described next.

Consider a document D and an element e in D. The *e-fragment* of D is the induced subgraph D_e of D whose nodes are e and all the nodes of D that are below e. Labels and atomic values in D_e are as in D.

Satisfaction of a condition C, w.r.t. the element e, is defined in the usual way. Suppose that C is a condition of the form θc, where θ is a comparison operator and c is a constant. Then, C is satisfied w.r.t. e if e is an atomic node with atomic value a_e such that $a_e \theta c$ holds.

If C is a query (i.e., a path expression) then C is satisfied if its evaluation result, w.r.t. the transitive closure of the e-fragment of D, is non-empty. If C is a conjunction of two conditions C_1 and C_2 then it is satisfied w.r.t. e if both C_1 and C_2 are satisfied w.r.t. e. If C is a disjunction of two conditions C_1 and C_2 then it is satisfied if either C_1 or C_2 is satisfied w.r.t. e. If C is a negation of a condition C_1 then it is satisfied w.r.t. e when C_1 is not satisfied w.r.t. e.

As for path queries, the answer to a query Q, when posed to a document D, is the evaluation result of Q w.r.t. the transitive closure $\bar{D}$.

Complexity of Query Evaluation Queries in $\mathcal{C}_{basic}$ can be computed in polynomial time in the size of the input, using a bottom-up evaluation w.r.t. the query.

Proposition 1. ([11, 12]) *Consider a query Q and a document D. The answer of Q w.r.t. D can be computed in $O(|Q| \cdot |D|)$ runtime, where $|Q|$ and $|D|$ are the sizes of Q and D, in correspondence.*

3 Access Control Rules

Our approach to access control in XML documents is based on rules rather than on using views. While views are 'positive' in the sense that they specify what the user is allowed to know, rules are 'negative' and specify what should be concealed from the user. Our access-control rules specify pairs of elements in the document and by this designate the relationships between these elements as being restricted. For example, suppose that in the university database a user should not be able to retrieve information on the enrollment of student in courses. In this case, querying the relationships between courses and student should be restricted by disallowing queries that do not obey the rules. Restrictions can be specified for pairs of elements that are connected by a path and not just for parent-child pairs. For example, it might be desirable to restrict queries from disclosing information on the grades given in some course, although grade elements are not immediate children of course elements.

3.1 The Syntax of Rules

Rules are formulated using XPath expressions. Each rule consists of two expressions. The two expressions specify pairs of ancestor and descendent elements that the relationship between them should not be revealed to users. The syntax of a rule is

$$\texttt{for}\ path_1\ \texttt{exclude}\ path_2$$

where $path_1$ and $path_2$ are XPath expressions.

Example 4. Suppose that we want to prevent queries from disclosing information on grades that were given to students. This restriction can be set by the following rule:

```
for //Student exclude /Grade
```

Example 5. Suppose that in the CS department, queries should not disclose information on grades of student and in addition should prevent queries from revealing the grades that were given in any course. For setting this restriction, two rules are used: the rule from the previous example and the following rule:

```
for /Department[/Name='CS']/Course exclude //Grade
```

Example 6. Suppose that we only want to prevent users from knowing which student was enrolled in which course. This can be done by using the following rule:

```
for /Department/Course exclude //Student
```

Note that this rule should not prevent execution of queries that 'jump' over the restricted edge. For example, a query that returns courses with the grades that were given in these course does not violate the rule. Neither does a query that returns students and their grades.

In many scenarios, different users should act according to different restrictions. For example, it could be that an institution would have a policy where a teacher of a course can have access to all the grades in courses that she teaches while students are able to see only their own grades. This requires the use of parameters that are provided by the system according to the user that poses the query. Parameters are written with a preceding dollar sign.

Example 7. Suppose that `$teacher` is equal to the name of the teacher that poses the query and is null for a user that is not a course teacher. Consider a policy where only a course teacher is permitted to see which grade each student has received in her course. This policy is set by the following rule:

```
for //Course[not(/Teacher=$teacher)]/Student exclude /Grade
```

Note that when `$teacher` is null the comparison `/Teacher=$teacher` is false.

Suppose now that students should be able to see their grades (in addition to the permission for the course teacher). Then the rule is modified to be:

```
for //Course[not(/Teacher=$teacher)]/Student [not(/SID=$s-id)] exclude
/Grade
```

where \$s-id is a parameter that contains the student id of the user (or null if the the user is not a student).

3.2 Local Validity

The access control rules that we present play a similar role to the role of views in the *Non-Truman Model* of [21]. Differently from the traditional *Truman Model*, the user is not encapsulated in a world that consists of only the data she is allowed to know. Instead, the user formulates her query knowing the original schema of the document. The query is then authorized and executed if it is *valid* and rejected otherwise. When using authorization views [21], a query is authorized if, in principle, it can be answered using only the information contained in the authorization views available to the user. When access-control rules are used, a query is considered valid, and is authorized, if it obeys all the rules. In Section 5, we show that obeying the rules means that the query does not disclose concealed information.

Local validity of a query is defined with respect to a document and a set of rules. First we consider the case where there is only a single rule. Let D be a document and Q be a query. Let $\rho = (x_1, x_2)$ be the rule, where x_1 and x_2 are the pair of XPath expressions in ρ.

Consider two elements e_1 and e_2 in D. The pair (e_1, e_2) is *restricted* if $e_1 \in x_1(D)$ and $e_2 \in x_1x_2(D)$, where x_1x_2 is the expression created by a concatenation of x_1 and x_2, $x_1(D)$ is the answer to x_1 over D and $x_1x_2(D)$ is the answer to x_1x_2 over D.

Let $\bar{D}$ be the transitive closure of D. The graph $\mathit{prune}_\rho(\bar{D})$ is called the *pruning* of $\bar{D}$ w.r.t. ρ and it is the result of removing from the edge set of $\bar{D}$ all the restricted pairs (e_1, e_2). Note that a pruning can cause the removal of both child edges and descendent edges.

When there are several access control rules, the definitions remain the same except that the pruning of $\bar{D}$ is done w.r.t. all the rules. Suppose that R is the set of rules. Then a pair of nodes in D is restricted if it is restricted according to one of the rules in R. We denote by $\mathit{Restricted}_R(D)$ the set of all restricted pairs w.r.t. D and R. That is, $\mathit{Restricted}_R(D) = \{(e_1, e_2) \mid \textit{exists } \rho = (x_1, x_2) \in R \textit{ such that } e_1 \in x_1(D) \textit{ and } e_2 \in x_1x_2(D)\}$. Pruning $\bar{D}$ is done by removing from the edge set of $\bar{D}$ all the pairs that are in $\mathit{Restricted}_R(D)$.

Definition 2 Local Validity. A query Q is *locally valid* w.r.t. D and the set of rules R if the following condition holds. For any document expansion $D^{//}$ such that $\mathit{prune}_R(\bar{D}) = \mathit{prune}_R(D^{//})$, the answer to Q w.r.t. D and the evaluation result of Q w.r.t. $D^{//}$ are equal.

Intuitively, a query Q is locally valid if it returns the answer $Q(D)$ for all the document expansions $D^{//}$ that are created by adding to the edges of

$prune_R(\bar{D})$ restricted pairs, i.e., extending the edge set of $prune_R(\bar{D})$ by a subset of $Restricted_R(D)$.

The following example explains why we need to consider, in Definition 2, all the document expansions $D^{//}$ such that $prune_R(\bar{D}) = prune_R(D^{//})$ instead of just considering the single document expansion $prune_R(\bar{D})$. In other words, consider the *equality on the pruning of the transitive closure (eptc)* condition. A query Q satisfies the eptc condition w.r.t. D if $Q(D) = Q(prune_R(\bar{D}))$. By Definition 2, the eptc condition is necessary but not sufficient for Q to be locally valid w.r.t. D. The following example demonstrates a security leak that could occur if the eptc condition was sufficient for local validity.

Example 8. Consider the courses-grades document D of Figure 1 and the rule ρ that is given in Example 4, where relationships between students and grades are concealed. Suppose that we authorize all the queries that provide the same answer when evaluated w.r.t. the document D and w.r.t. the pruning of the transitive closure $prune_\rho(\bar{D})$. A malicious user could then pose the following 101 queries: Q_i : `//Student[/SID='12345' and /Grade=`i`]`, for $i = 0, 1, \ldots, 100$. In all the 100 cases where $i \neq 98$, the query will be authorized and return an empty result. For $i = 98$, which is the grade of the student in the DB course, the query will not be authorized. This allows the malicious user to reveal the grade of a student in some course.

Such a leak of information does not occur when only locally valid queries, according to Definition 2, are authorized. To see why this is true, consider the document expansion $D^{//}$ that is constructed as follows. $D^{//}$ is the result of removing two edges and adding two new edges to the transitive closure $\bar{D}$. The removed edges are the two Student-Grade edges that connect Node 301 to Node 321 and Node 302 to Node 322. The two edges that are added are Student-Grade edges that connect Node 301 to Node 322 and Node 302 to Node 321. All these added and removed edges are Student-Grade edges and thus, are removed in a pruning w.r.t. ρ. That is, $prune_\rho(\bar{D}) = prune_\rho(D^{//})$. Yet, evaluating Q_{93} w.r.t. $D^{//}$ provides a different answer than the answer to Q_{93} over D. Thus, Q_{93} is not valid. A similar construction can show that Q_{78} is also not valid. All the three queries Q_{78}, Q_{93} and Q_{98} are rejected. Thus, all that the user can tell is that the grade of Student '12345' is one of the grades 78, 93, 98; however, this she could have learned from the result of the valid query `//Grade`.

The next proposition shows that when for two documents D_1 and D_2 the pruning of their transitive closure are two equal graphs, the answer to a locally valid query w.r.t. one of the documents is equal to the answer to the query over the other document.

Proposition 3. *Given a set of rules R and two documents D_1 and D_2, such that $prune_R(\bar{D}_1) = prune_R(\bar{D}_2)$, if a query Q is locally valid w.r.t. D_1 and R then $Q(D_1) = Q(D_2)$.*

Proof. (Sketch) For every edge (e_1, e_2) in D_i $(i = 1, 2)$, (e_1, e_2) is either an edge in D_{3-i} or a restricted pair according to R. Otherwise, (e_1, e_2) would be in $prune_R(\bar{D}_i)$ but will not be in $prune_R(\bar{D}_{3-i})$. Similarly, for every path in D_i, from e_1 to e_2, either there is a path in D_{3-i} from e_1 to e_2 or the pair (e_1, e_2) is a restricted pair according to R. Because Q is valid w.r.t. D_1 and R, the answer to Q depends only on the non restricted edges of $\bar{D}_1$. Therefore, $Q(D_1) = Q(D_2)$.

The next corollary shows that when the pruning of the transitive closure is equal for two documents D_1 and D_2, a locally valid query w.r.t. one of the documents is locally valid w.r.t. the other document.

Corollary 4. *Consider a set of rules R and let D_1 and D_2 be two documents such that $prune_R(\bar{D}_1) = prune_R(\bar{D}_2)$. If a query Q is locally valid w.r.t. D_1 and R then Q is also locally valid w.r.t. D_2 and R.*

Proof. Let $\mathcal{D}$ be the set of all document expansion $D^{//}$ such that $prune_R(D^{//}) = prune_R(D_1)$. Since Q is locally valid w.r.t. D_1 and R, $Q(D^{//}) = Q(D_1)$, for every $D^{//}$ in $\mathcal{D}$. Because $prune_R(\bar{D}_1) = prune_R(\bar{D}_2)$, $\mathcal{D}$ is also the set of all document expansion $D'^{//}$ such that $prune_R(D'^{//}) = prune_R(D_2)$. By Proposition 3, $Q(D_1) = Q(D_2)$ and hence, $Q(D_2) = Q(D'^{//})$, for every $D'^{//}$ in $\mathcal{D}$.

3.3 Global Validity

Global validity of a query is defined w.r.t. a set of rules R and a schema S. The query Q is globally valid if for every document D that conforms to S, Q it is locally valid w.r.t. D and R.

Global validity is more restrictive than local validity. That is, given a schema S, a document D that conforms to S, a set of access-control rules R and a query Q, it could be that Q is locally valid w.r.t. D and R but Q is not globally valid w.r.t. S and R. For example, suppose that R contains only the rule given in Example 4. This rule rejects queries that use the relationship between student entities and grade entities. Now, suppose that the following query is posed to the document in Figure 1: `//Student[/SID='00000']//Grade`. The query asks for the grade of the student with id '00000'. If there was a student with id '00000' in the document, then the query would not be considered locally valid and would have been rejected. Since there is no student with id '00000', there is no edge to prune and the query is locally valid. Note that the query does not reveal the grade of any existing student.

Although Q is locally valid, it is not globally valid w.r.t. R and the schema S shown in Figure 2. This is because it is possible to construct a document D' that conforms to S and contains a student with id '00000'. The query will not be locally valid w.r.t. D' and R and thus, is not globally valid w.r.t. S.

In some cases, global validity could be too restrictive; however, it does have some advantages over local validity. Suppose that there is a collection of documents and all the documents conform to the same schema. In this case, if a query is globally valid, w.r.t. the schema and the rules, then checking validity of the query w.r.t. each documents is spared. Another advantage is that validity of globally valid queries is not needed to be checked after each update of a document, as long as after the update the document still conforms to the schema.

4 Testing Validity of Queries

In this section, we discuss testing of local validity and global validity. Being able to efficiently test whether a query Q is globally valid, w.r.t. a schema S and a set of rules R, or locally valid, w.r.t. a document D and a set of rules R, is essential to having an effective access-control system.

4.1 Testing Local Validity

A naive way of testing local validity of Q, w.r.t. D and R, is by computing $Q(D^{//})$, for all the document expansions $D^{//}$ such that $prune_R(\bar{D}) = prune_R(D^{//})$; and checking that in all these cases holds $Q(D^{//}) = Q(D)$. Obviously, this naive test is inefficient, because the number of document expansions $D^{//}$, such that $prune_R(\bar{D}) = prune_R(D^{//})$, is exponential in the size of the set $Restricted_R(D)$—the set of all restricted pairs w.r.t. D and R.

We present now a polynomial-time test for local validity of *monotone* queries. A document D_1 is contained in a document D_2, denoted $D_1 \subseteq D_2$, if the element set (edge set) of D_1 is contained in the element set (edge set) of D_2. A query Q is *monotone* if for every pair of documents $D_1 \subseteq D_2$ holds $Q(D_1) \subseteq Q(D_2)$.

Next, we define the *full expansion* of a document. The full expansion is used for testing validity of monotone queries. Let $prune_R(\bar{D})$ be the pruning of the transitive closure of D w.r.t. R. Let $Restricted_R(D)$ be the set of all restricted pairs w.r.t. D and R (recall that $Restricted_R(D) = \{(e_1, e_2) \mid \textit{exists } \rho = (x_1, x_2) \in R \textit{ such that } e_1 \in x_1(D) \textit{ and } e_2 \in x_1x_2(D)\}$). The *full expansion* of D is a document expansion $FE_R(D)$ that is created by adding to the edge set of $prune_R(\bar{D})$ all the pairs in $Restricted_R(D)$. That is, if (e_1, e_2) is in $Restricted_R(D)$ then in $FE_R(D)$ there is an edge from e_1 to e_2.

Proposition 5. *Let D be a document and R be a set of access-control rules. A monotone query Q is locally valid w.r.t. D and R if and only if the following two conditions hold:*

- $Q(D) = Q(prune_R(\bar{D}))$
- $Q(D) = Q(FE_R(D))$

Proof. Suppose that Q is locally valid w.r.t. D and R, then from the definition of local validity follows that the conditions in the proposition hold. Since $prune_R(FE_R(D))$ is equal to $prune_R(\bar{D})$, it follows from Definition 2 that $Q(D) = Q(FE_R(D))$. From the same argument follows $Q(D) = Q(prune_R(\bar{D}))$.

We show now the other direction. To show that Q is locally valid, we need to show that $Q(D^{//}) = Q(D)$ for any document expansion $D^{//}$ such that $prune_R(\bar{D}) = prune_R(D^{//})$.

Let $D^{//}$ be such a document expansion. Then,

$$prune_R(\bar{D}) \subseteq D^{//} \subseteq FE_R(D).$$

Because Q is monotone,

$$Q(prune_R(\bar{D})) \subseteq Q(D^{//}) \subseteq Q(FE_R(D)).$$

Now, if $Q(D) = Q(prune_R(\bar{D}))$ and $Q(D) = Q(FE_R(D))$, then $Q(D^{//}) = Q(D)$.

Based on Proposition 5, the local validity, w.r.t. D and R, of a monotone query Q is tested by the next algorithm.

Check-Validity-of-Monotone-Query(Q, R, D)

1. Compute the set $Restricted_R(D)$ of all the restricted pairs according to R.
2. Compute $\bar{D}$ and $prune_R(\bar{D})$.
3. Compute $FE_R(D)$, the full expansion of D, by adding to the edge set of $\bar{D}$ all the pairs in $Restricted_R(D)$.
4. Evaluate Q w.r.t. the following three: the document D, the pruning of the transitive closure $prune_R(\bar{D})$ and the full expansion $FE_R(D)$. If $Q(D) = Q(prune_R(\bar{D}))$ and $Q(D) = Q(FE_R(D))$, then return "Q is locally valid". Else, return "Q is not locally valid".

The above algorithm shows that testing local validity has a polynomial time complexity.

Proposition 6. *Given a monotone query Q, a database D and a set of rules $R = \{r_1, \ldots, r_n\}$, testing local validity of Q w.r.t. D and R has time complexity $O(|D|^2 \cdot (\Sigma_{i=1}^{n}|r_i|^2 + |Q|^2))$.*

Proof. Using a bottom-up evaluation it is possible to compute the matchings of a tree query Q to a database D in time complexity $O(|Q|^2 \cdot |D|^2)$. Similarly, evaluation of the expressions in a rule r_i is in $O(|r_i|^2 \cdot |D|^2)$ runtime.

We say that a query Q is *strictly positive* if Q does not contain any negative condition, i.e., the logic operator **not** does not appear in any part of the query. A strictly positive query is monotone and, hence, validity of strictly positive queries can be tested using the above algorithm.

Testing local validity of non-monotonic queries cannot be done in polynomial runtime, unless $\mathsf{P} = \mathsf{coNP}$. Thus, it is unlikely to have a polynomial-time test for local validity of non-monotonic queries. This is shown by the following proposition.

Proposition 7. *Given a query Q, a set of rules R and a database D, deciding if Q is locally valid w.r.t. D and R is* coNP*-complete.*

Proof. The problem is in coNP. This is because for a given document expansion $D^{//}$ it is possible to verify in polynomial time, in the size of Q, R and D, that $prune_R(D^{//}) = prune_R(D)$ and that $Q(D) \neq Q(D^{//})$. That is, disqualifying local validity of Q w.r.t. D and R is in polynomial time in the size of the input.

We show that deciding local validity is coNP-hard by a reduction of the *validity of Boolean expressions* problem to testing local validity of queries. Validity of Boolean expressions is the problem of deciding whether a Boolean expression, over a set of propositional variables $p_1, \ldots, p_n$, is satisfied by every assignment to $p_1, \ldots, p_n$. It is known that validity of Boolean expressions is coNP-hard.

Let ψ be a Boolean expression over the propositional variables $p_1, \ldots, p_n$. We show how to construct a database D, a set of rules R and a query Q such that Q is locally valid w.r.t. D and R if and only if the propositional formula ψ is either a valid or an unsatisfiable Boolean expression.

The database D has the following nodes: a root $root_D$, a node e_a with label l_a and n nodes e_{p_i} $(1 \leq i \leq n)$ with labels l_{p_i}. In D, there is an edge from the root to e_a. From e_a there is an edge to each one of the n nodes e_{p_i}.

The set of rules R contains n rules of the form: `for` $//l_a$ `exclude` $//l_{p_i}$, where $1 \leq i \leq n$.

The query Q is constructed according to the propositional formula ψ. First, we create from ψ a condition C_ψ by replacing $\wedge$, $\vee$, and $\neg$ with `and`, `or` and `not`, respectively, and adding "/" before the propositional letters. For example, for a formula $\psi = p_1 \vee (p_2 \wedge \neg p_3)$ the condition C_ψ is: $/p_1$ `or` `(`$/p_2$ `and not` $/p_3$`)`. The query Q is $//l_a$`[`C_ψ`]`.

Let $\mathcal{D}$ be the set of document expansions $D^{//}$ that satisfy $prune_R(D^{//}) = prune_R(\bar{D})$. There is a bijection between $\mathcal{D}$ and the set of all possible assignments to the propositional variables $p_1, \ldots, p_n$. We now describe this bijection.

For every document expansion $D^{//}$ in $\mathcal{D}$ there is a corresponding assignment θ to the propositional variables $p_1, \ldots, p_n$. If there is an edge in $D^{//}$ from e_a to p_i $(1 \leq i \leq n)$, then θ assigns *true* to p_i. Else, θ maps p_i to *false*.

It is easy to see that the assignment θ satisfies the formula ψ if and only if the condition C_ψ is satisfied w.r.t. the node e_a in the document expansion that corresponds to θ. Hence, the following two are equivalent. *(1)* The formula ψ is either a valid or an unsatisfiable Boolean expression. *(2)* Q returns the same answer when posed to D and when posed to any document expansions in $\mathcal{D}$. The answer returned by Q is the empty set if ψ is unsatisfiable and it is the element e_a if ψ is valid. Therefore, Q is locally valid w.r.t. D and R if and only if ψ is either valid or unsatisfiable.

To decide validity of a Boolean expression ψ, one should construct Q, R and D as described above, and test local validity of Q w.r.t. D and R. If Q is not locally valid then ψ is not valid. If Q is valid then ψ is either valid or unsatisfiable. Computing the truth value of ψ for one arbitrary assignment can tell if indeed ψ is valid or that it is unsatisfiable.

We saw that deciding local validity of non-monotonic queries is intractable; however, there are cases where it is possible to efficiently verify local validity of non-monotonic queries by showing that the queries are globally valid. Testing global validity is discussed in the next section.

4.2 Testing Global Validity

Earlier, we discussed the importance of testing global validity of a query Q w.r.t. a schema S and a set of rules R. One of the advantages of testing global validity, on testing local validity, is that the input contains a schema instead of a database. Thus, the test for global validity has a potential of being more efficient then the test for local validity.

In the first part of this section we provide a sufficient condition for global validity. Then, we show that, with a small modification, this condition is both necessary and sufficient when both the query and the rules are *conjunctive expressions*, i.e., do not contain a disjunction (`or`) nor a negation (`not`). In the second part, we show that deciding if a query is globally valid is at least as hard as deciding containment of XPath queries. Hence, there are cases where testing global validity does not have a polynomial-time algorithm.

Query skeleton has been defined in Section 2.3. We present now a sufficient condition for global validity, based on *mapping* query skeletons to schemas.

A mapping of a query skeleton to a schema is similar to a matching of a query pattern w.r.t. a document, as defined in Section 2.3. The mapping is a function from the nodes of the skeleton, except the root, to the nodes of the schema. In a mapping of a skeleton to a schema, labeled are preserved. That is, a node with label l is mapped to a node with label l and a node with a wildcard can be mapped to any schema node. The mapping preserves the edges of the skeleton. That is, a pair of skeleton nodes that are connected by a child edge (descendent edge) is mapped to a pair of schema nodes that are connected by an edge (a path). The root of the skeleton is not mapped to a schema node, however, the child of the root, v, is mapped to the root of the schema if there is a child edge from the skeleton root to v.

The number of mappings of a skeleton to a schema depends on the number of wildcards in the skeleton and the size of the schema. It is easy to see that if there are k wildcards in the skeleton and m nodes in the schema, then there are at most m^k mappings.

We need some more definitions. Consider a rule $\rho = (x_1, x_2)$. The pattern P_ρ of ρ and the skeleton $\sharp(x_1 x_2)$ of ρ are the pattern and skeleton of the expression $x_1 x_2$. Recall that $x_1 x_2$ is the concatenation of x_1 and x_2. Also, recall that the difference between a pattern and a skeleton is that in a skeleton conditions are being unfolded and in a pattern they are not unfolded. Let P_1 and P_2 be the patterns of x_1 and x_2, respectively. The pair (t_1, t_2), where t_1 is the target of P_1 and t_2 is the target of P_2, consists of two nodes in P_ρ (and also in $\sharp(x_1 x_2)$). This pair is called the *target nodes* of the the pattern (skeleton) of ρ.

Let P'_ρ be a pattern that is similar to P_ρ except that conditions that are attached to nodes are removed. Given a schema S, we say that a pair (n_1, n_2) of nodes in S is *p-restricted* by ρ if there is a mapping of the pattern P'_ρ to S in which t_1 is mapped to n_1 and t_2 is mapped to n_2.

Now, given a set of rules R and a schema S, we denote by *P-Restricted*$_R(S)$ the set of all pairs in S that are p-restricted by a rule ρ in R.

Consider a schema S, a set of rules R and a query Q. Let $\sharp(Q)$ be the skeleton of Q and *P-Restricted*$_R(S)$ be the set of p-restricted nodes in S. We say that Q is *safe* w.r.t. S and R, if for every mapping μ of $\sharp(Q)$ to S and for every edge (v_1, v_2) in $\sharp(Q)$, μ does not map the nodes v_1 and v_2 to a pair of p-restricted nodes, i.e., to a pair in *P-Restricted*$_R(S)$.

Now we can provide the sufficient condition for global validity.

Proposition 8. *If a query Q is safe w.r.t. a schema S and a set of rules R, then Q is globally valid w.r.t. S and R.*

Proof. Suppose that Q is safe w.r.t. S and R. We need to show that for every document D that conforms to S holds that Q is locally valid w.r.t. D and R.

Let D be a document that conforms to S (i.e., there exists a conformance mapping μ_c from D to S) and let *P-Restricted*$_R(S)$ be the set of p-restricted nodes in S. We denote by E_r pairs of nodes in D that are mapped by μ_c to pairs in *P-Restricted*$_R(S)$. By E_n we denote all the other pairs of nodes in D.

Consider a document expansion $D^{//}$ of D and let D^* be a document expansion that is created from D by adding or removing edges (n_1, n_2) such that $(n_1, n_2) \in E_r$. The correctness of the proposition follows from the next observation. The result of posing Q to $D^{//}$ and the result of posing Q to D^* are equal. This is because Q is safe w.r.t. S and R. That is, in the matching of the pattern of Q (or the pattern of any expression that appears in a condition in Q) to $D^{//}$, all the edges of the pattern are mapped to edges that are in E_n while the added and removed edges are in E_r.

The pruning according to R can remove from a document expansion $D^{//}$ only edges (n_1, n_2) such that (n_1, n_2) is in E_r. Thus, $Q(D)$ is equal to $Q(prune_R(\bar{D}))$. In addition, $Q(prune_R(\bar{D}))$ is equal to $Q(D^{//})$, for every document expansion $D^{//}$ such that $prune_R(\bar{D}) = prune_R(D^{//})$. This shows that Q is locally valid w.r.t. D and R. and completes the proof.

Safety of a query is a sufficient condition for global validity w.r.t. a schema and a set of rules. When the number of wildcards in the query and in each rule is bounded, the condition can be tested in polynomial time in the size of the input.

Proposition 9. *Consider a schema S, a set of rules $R = \{\rho_1, \ldots, \rho_n\}$ and a query Q. Let k be the maximum on the number of times that a wildcard appears in Q or in one of the rules of R. Then, there is a test that decides in $O(s^2 + (q + \Sigma_{i=1}^{n} r_i)s^k)$ time if Q is safe w.r.t. S and R, where s is the number of nodes in S, q is the number of nodes in the skeleton of Q, and r_i ($1 \leq i \leq n$) is the number of nodes in the skeleton of the rule ρ_i.*

Proof. As an initial step, we create a data structure that for a node n and a label l returns in $O(1)$ the list of children (descendents) of n that have label l. This can be done in $O(s^2)$.

On the second step, *P-Restricted*$_R(S)$ is computed. The computation is by iterating over all the rules in R and computing the p-restricted nodes according to each rule. For each rule ρ_i in R, the mappings of P_{ρ_i} (the pattern of ρ_i) to S are computed using a top-down traversal over ρ_i. In each step of the traversal, the partial mappings, computed up to that step, are extended w.r.t. a new node of P_{ρ_i}. The extension is by mapping the new node to a schema node, according to the definition of a mapping. Since there are at most s^k mappings of ρ_i to S, computing all the mappings is in $O(r_i \cdot s^k)$. The elements of the set $P\text{-}Restricted_R(S)$ are given by the assignments to the target nodes of the rules. If the set $P\text{-}Restricted_R(S)$ is stored in a hash table, it requires $O(1)$ time to test if a pair of nodes in in $P\text{-}Restricted_R(S)$.

On the third step, the mappings μ of the skeleton $\sharp(Q)$ to S are computed. This can be done in $O(q \cdot s^k)$ in a similar fashion to the mapping of a rule pattern P_{ρ_i} to S. Finally, it is tested that for every mapping μ and edge (v_1, v_2) in Q, the nodes v_1 and v_2 are not mapped by μ to a pair in $P\text{-}Restricted_R(S)$. This can be done in $O(q \cdot s^k)$ time since there are $q - 1$ edges in Q and there are at most s^k mappings of $\sharp(Q)$ to S.

The class $\mathcal{C}_{conj}$ consists of conjunctive queries, i.e., queries in which `or` and `not` do not appear. Global validity has a complete characterization when the query Q and all the rules in R are in the class $\mathcal{C}_{conj}$. In the following, we provide some definitions and then the characterization.

Consider the skeleton $\sharp(\rho)$ of a rule ρ and let (t_1, t_2) be the pair of target nodes of $\sharp(\rho)$. Given a schema S, we say that a pair (n_1, n_2) of nodes in S is *$\sharp$-restricted* by ρ if there is a mapping of the skeleton $\sharp(\rho)$ to S in which t_1 is mapped to n_1 and t_2 is mapped to n_2. We say that (n_1, n_2) is *conditionally $\sharp$-*restricted, w.r.t. the comparison $\theta\, c$, if there is a comparison $\theta\, c$ in the condition attached to t_2 in the pattern of ρ.

Given a set of rules R and a schema S, we denote by $\sharp\text{-}Restricted_R(S)$ the set of all pairs in S that are $\sharp$-restricted by a rule ρ in R.

Consider a schema S, a set of rules R and a query Q. Let $\sharp(Q)$ be the skeleton of Q and $\sharp\text{-}Restricted_R(S)$ be the set of $\sharp$-restricted nodes in S. We say that Q is *$\mathcal{C}_{conj}$-safe* w.r.t. S and R, if for every mapping μ of $\sharp(Q)$ to S and for every edge (v_1, v_2) in $\sharp(Q)$, μ does not map the nodes v_1 and v_2 to a pair of $\sharp$-restricted nodes n_1 and n_2, i.e., to a pair (n_1, n_2) in $\sharp\text{-}Restricted_R(S)$, unless the following holds. In the pattern of Q there is a comparison condition $\theta_1\, c_1$ attached to v_2, the pair (n_1, n_2) is conditionally $\sharp$-restricted w.r.t. the comparison $\theta_2\, c_2$ and there is no constant c that can satisfy both $c\, \theta_1\, c_1$ and $c\, \theta_2\, c_2$.

Now we can provide a characterization for global validity to the case where the query and the rules are in $\mathcal{C}_{conj}$.

Proposition 10. *Suppose that Q is a query in $\mathcal{C}_{conj}$ and R is a set of rules such that the expressions in each rule are in $\mathcal{C}_{conj}$. Then, Q is $\mathcal{C}_{conj}$-safe w.r.t. a schema S and R if and only if Q is globally valid w.r.t. S and R.*

Proof. (sketch) First, we need to show that if Q is $\mathcal{C}_{conj}$-safe w.r.t. S and R then Q is globally valid w.r.t. S and R. The proof, in this case, is similar to the proof of Proposition 9.

For the other direction, assume that Q is not $\mathcal{C}_{conj}$-safe w.r.t. S and R. Then, we can construct a document D such that D conforms to S and Q is not locally valid w.r.t. D and R. This will show that Q is not globally valid w.r.t. S and R.

Next, we describe the construction of D. Since Q is not $\mathcal{C}_{conj}$-safe w.r.t. S and R, there is a mapping $\mu_{\sharp(Q)}$ of $\sharp(Q)$ to S and an edge (v_1, v_2) in Q, such that $\mu_{\sharp(Q)}$ maps the nodes v_1 and v_2 to a pair of schema nodes n_1 and n_2 that are $\sharp$-restricted, i.e., $(n_1, n_2) \in \sharp\text{-}Restricted_R(S)$. Further more, there is a rule ρ in R and a mapping μ_ρ of the skeleton $\sharp(\rho)$ to S, such that μ_ρ maps the target nodes t_1 and t_2 of the skeleton $\sharp(\rho)$ to n_1 and n_2.

The node set of D consists of all the nodes in $\sharp(Q)$ and all the nodes in $\sharp(\rho)$, except for the root and the nodes t_1 and t_2.

The edge set of D consists of all the edges in $\sharp(Q)$. In addition, it consists of all the edges of $\sharp(\rho)$ after replacing the root node with the root of $\sharp(Q)$ and the target nodes t_1 and t_2 with the nodes v_1 and v_2 of $\sharp(Q)$, in correspondence.

The atomic values attached to the atomic nodes are arbitrary if the atomic node does not have, in the expression pattern, an attached condition that includes a comparison to a constant. If, in the pattern, a comparison to a constant, of the form $\theta\, c$, appears in a condition that is attached to an atomic node v, then the atomic value that is attached to v in D must satisfy $\theta\, c$.

Conformance of D to S is due to the existence of a mapping of $\sharp(Q)$ and a mapping of $\sharp(\rho)$ to S.

The query Q is not locally valid w.r.t. D and R. This is because when posing Q to D, the answer is not empty. Yet, in the pruning of $\bar{D}$ according to R, the edge from v_1 to v_2 is removed because of the rule ρ. Thus, the answer to Q when posed to $prune_R(\bar{D})$ is empty. Consequently, Q is not locally valid and, hence, Q is not globally valid w.r.t. S and R.

The characterization for global validity that is provided in Proposition 10 can be tested in a similar fashion to the test described in Proposition 9. The time complexity of the two tests is also the same. Thus, when the query and the rules are all conjunctive expressions and the number of wildcards in each expression does not exceed a constant k, there is a polynomial time test for deciding if a query is globally valid w.r.t. a given schema and a given set of rules.

In the general case, when either the query or the rules are not conjunctive queries, deciding if a query is globally valid is a computationally hard problem. We present now a reduction of *XPath containment* to testing global validity. The reduction shows that testing global validity is as hard as testing containment of XPath expressions.

Given two XPath expressions x_1 and x_2, we say that x_1 is *contained* in x_2, denoted $x_1 \subseteq x_2$, if $x_1(D) \subseteq x_2(D)$, for any document D. XPath containment, under a class $\mathcal{C}$ of XPath expressions, is the following decision problem. Given two XPath expressions x_1 and x_2 in $\mathcal{C}$, is x_1 contained in x_2? XPath containment has been investigated for many classes of XPath expressions [6, 13, 15, 16, 20]. The most notable class is the one that contains `/`, `//`, `[ ]` and `*` but does not not allow disjunction or negation. For this class, deciding containment is coNP-complete. When adding disjunction, the problem becomes PSPACE-complete.

Proposition 11. *Consider a class $\mathcal{C}$ of XPath expressions. Given a pair x_1 and x_2 of expressions in $\mathcal{C}$, there is a query Q, a schema S and a rule ρ such that the size of Q is linear in the size of the input (x_1 and x_2), the size of S is quadratic in the size of the input, ρ is of constant size and the following two conditions are equivalent.*

1. x_1 *is contained in* x_2.
2. Q *is globally valid w.r.t.* S *and* $\{\rho\}$.

Proof. We construct a query Q, a schema S and a rule ρ as follows. Let L be the set of labels that appear in either x_1 or x_2. Let a and b be two labels that are not in L. The query Q is `/a/b[`x_1 `and (not` x_2`)]`. The rule ρ is: `for /a exclude /b`.

The schema S has a root $root_S$ with label a, and the nodes $L \cup \{b, star\}$, where *star* is a label that is not in L and, intuitively, represents labels, other than `a` and `b`, that do not appear in x_1 or in x_2. S has the following edges. There is an edge from the root (the node with label a) to b. There are edges in both directions between each pair of labels in $L \cup \{b, star\}$. For the all the nodes $L \cup \{star\}$, S includes loops, i.e., an edge from the node to itself.

We show that the two conditions above are equivalent. We start by showing that if the first condition holds then the second condition holds.

Suppose that x_1 is contained in x_2. Then, for every document expansion $D^{//}$, holds $Q(D^{//}) = \emptyset$. This is because the condition `[`x_1 `and (not` x_2`)]` cannot be satisfied if $x_1 \subseteq x_2$.

For a given document D that conforms to S, there are only two document expansion such that $prune_\rho(\bar{D}) = prune_\rho(D_j^{//})$. The first is $prune_\rho(\bar{D})$ and the second is $\bar{D}$. The difference between $prune_\rho(\bar{D})$ and $\bar{D}$ is that in $\bar{D}$ there is an edge from the node with label a to the node with label b while in $prune_\rho(\bar{D})$ there is no such edge. As been argued above, when Q is evaluated w.r.t. any document expansion, including $prune_\rho(\bar{D})$ and $\bar{D}$, the result is an empty set. Thus, Q is globally valid.

For the other direction, suppose that Q is globally valid w.r.t. S and ρ. We need to show that x_1 is contained in x_2, that is, $x_1(D') \subseteq x_2(D')$, for any document D'.

Let D' be a given document. We create from D' a new document D that conforms to S. We replace the root of D' with a new node e_b that is labeled with b (e_b is not the root of D). We add a new node e_a with label a as the parent of e_b. The node e_a is actually the new root $root_D$ and e_b is the only child of e_a. Finally, we replace every label that is not in L with the label *star*. It is easy to see that D conforms to S.

According to our assumption, Q is globally valid w.r.t. S and ρ. Thus, $Q(D) = Q(D^{//})$, for every document expansion $D^{//}$ such that $prune_\rho(\bar{D}) = prune_\rho(D^{//})$. In particular, $Q(D) = Q(prune_\rho(\bar{D}))$.

There is no edge from e_1 to e_2 in $prune_\rho(\bar{D})$, so $Q(prune_\rho(\bar{D}))$ is an empty set. Hence, $Q(D)$ is empty. From the emptiness of $Q(D)$ we can infer that the condition `[`x_1 `and (not` x_2`)]` is not satisfied w.r.t. D'. Note that replacing the labels that are not in L with the label *star* has no influence on the evaluation of the expressions x_1 and x_2. Consequently, $x_1(D') \subseteq x_2(D')$.

Since $x_1(D') \subseteq x_2(D')$ for any document D', x_1 is contained in x_2. This shows the equivalence between the two conditions and completes the proof.

In the above proof, we use a reduction in which the rule ρ is a conjunctive query while the query Q contains negation. We can change the reduction in the proof by defining Q to be `/a/b` and ρ to be `for /a exclude /b[`x_1 `and (not` x_2`)]`. In the modified reduction, Q is a conjunctive query while ρ contains negation.

5 Coherent Set of Rules

Consider the university document from Figure 1. Suppose that a privacy policy requires to conceal the relationships between courses and student grades. Removing the descendent edges between courses and grades is insufficient if the edges that connect courses to students and the edges that connect students to grades remain intact. This is because it is possible to infer the removed edges from the edges that were not removed. A set of rules is considered *coherent* if it is impossible to infer the removed edges from the edges that were not removed in the pruning according to the rules.

Before providing the formal definition for a coherent set of rules, we give an example of the two cases where a relationship can be inferred from a pair of known relationships.

Example 9. Suppose that in the university document it is known that the CAD course (eid 203) is given in the EE department (eid 102) and student 56789 (eid 303) enrolled the CAD course. In this case, the relationship between Element 102 and Element 303 can be derived from the other relationships, thus, there is no point in concealing it.

Suppose that it is known that Student 12345 (eid 301) studies in the CS department (eid 101) and has enrolled the DB course (eid 211). In this case, knowing that the document is a tree allows to infer that the DB course is given in the CS department (i.e., eid 201 and eid 301 are related).

We now define when a set of rules is coherent. Consider a document D and a set of rules R. The set R has an *incomplete concealment* in a document D if one of the following two cases hold.

1. Lack of transitive: D has three elements e_1, e_2 and e_3 such that $prune_R(\bar{D})$ has an edge from e_1 to e_2 and an edge from e_2 to e_3, but $prune_R(\bar{D})$ does not have an edge from e_1 to e_3.
2. Lack of reverse transitivity: *(1)* D is a tree, and *(2)* there are three elements e_1, e_2 and e_3 in D, such that $prune_R(\bar{D})$ has an edge from e_1 to e_3 and an edge from e_2 to e_3; however, $prune_R(\bar{D})$ does not have an edge from e_1 to e_2.

Definition 12 A Coherent Set of Rules. A set of rules R is *coherent* w.r.t. a document D if an incomplete concealment does not occur in D. The set R is coherent w.r.t. a schema S if R is coherent w.r.t. every document that conforms to S.

5.1 Creating a Coherent Set of Rules w.r.t. a Document

The following polynomial-time algorithm checks if a set of rules R is coherent w.r.t. a document D.

1. Compute the set E_R of restricted edges. To do this, for each rule ρ in R compute the two expressions that ρ comprises and add each pair of restricted elements to E_R.
2. Check for lack of transitivity as follows. Create a binary relation r from E_R. Let *source* and *target* be the attributes in the schema of r. Perform a self-join on r by joining tuples t_1 of r to tuples t_2 of r when $t_1.target = t_2.source$. Then, project the join on $t_1.source, t_2.target$. If there is a pair of elements (e_1, e_2), in the result of the join and projection, such that (e_1, e_2) is not in E_R then there exists a lack of transitivity and R is not coherent.
3. If the document D is a tree, check for the lack of reverse transitivity. To do so, create a relation r from E_R, as in the previous step. Then create the self-join and the projection as is Step 2, with the following difference. In the join, a pair of tuples t_1 and t_2 should be joined if $t_1.target = t2.target$. The projection should be on $t_1.source, t_2.source$. As in Step 2, it should be checked if the result of the join contains a pair (e_1, e_2) that is not in E_R. If there is such a pair then R is not coherent.
4. Unless in Step 2 or Step 3 the algorithm stops with the answer that R in not coherent, return that R is coherent.

Proposition 13. *For a tree document D and a set of rules R, there is a polynomial-time algorithm, in the size of R and D, that decides whether R is coherent w.r.t. D.*

So far, we described how to check that a set of rules is coherent w.r.t. a document. But the presented algorithm does not provide any guidance to creating a coherent set of rules. In the following, we provide two characterizations that try to fill this void.

We say that an edge (e_1, e_2) in a transitive closure $\bar{D}$ is *encapsulating* an edge (e'_1, e'_2) if there is a path ϕ in $\bar{D}$ that goes through the two edges (e_1, e_2) and (e'_1, e'_2) such that one of the following three cases holds: *(1)* e_1 appears on ϕ before e'_1 and e'_2 appears before e_2. *(2)* $e_1 = e'_1$ and e'_2 appears on ϕ before e_2. *(3)* e_1 appears on ϕ before e'_1 and $e_2 = e'_2$.

We now present a necessary condition for the coherency of a set of rules w.r.t. a document.

Proposition 14. *If a set of rules R is coherent w.r.t. a document D then the following condition holds. For every descendent edge (e_1, e_2), in $\bar{D}$, that is removed in the pruning of $\bar{D}$ w.r.t. R, there is an edge (e'_1, e'_2) in $\bar{D}$ such that (e'_1, e'_2) is encapsulated by (e_1, e_2) and (e'_1, e'_2) is removed in the pruning of $\bar{D}$.*

When a document D is not a tree, the condition provided in Proposition 14 is both necessary and sufficient.

We say that two edges that have the same source (e_1, e_2) and (e_1, e'_2) are *parallel* in $\bar{D}$ if either there is a path from e_2 to e'_2 or vice-versa. That is, these two edges do not lead to two disjointed parts of $\bar{D}$. We now use this definition in the next proposition, which provides a sufficient condition for coherency.

Proposition 15. *Let R be a set of rules and D be a document. If the following condition holds then R is coherent w.r.t. D. For every edge (e_1, e_2) that is removed in the pruning of $\bar{D}$ w.r.t. R, all the edges (e_1, e'_2) that are parallel to (e_1, e_2) are also removed in the pruning of $\bar{D}$.*

A private case to the condition in Proposition 15 is when for each element e in D either all the edges in $\bar{D}$, from e to some node, are being removed in the pruning or none of them is being removed.

5.2 Creating a Coherent Set of Rules w.r.t. a Schema

We provide now a condition that is sufficient for a set of rules to be coherent w.r.t. a schema. The condition uses containment of XPath expression, hence, in most cases testing the condition is not tractable.

Let x_1 and x_2 be two XPath expressions. We say that x_1 is contained in x_2 w.r.t. a schema S, denoted $x_1 \subseteq_S x_2$, if $x_1(D) \subseteq x_1(D)$ for every document D that conforms to S. It is easy to see that if $x_1 \subseteq x_2$ then $x_1 \subseteq_S x_2$.

Proposition 16. *Let R be a set of access control rules w.r.t. a schema S. Consider the following two conditions.*

1. *For every pair of rules $\rho^{(1)} = ({x_1}^{(1)}, {x_2}^{(1)})$ and $\rho^{(2)} = ({x_1}^{(2)}, {x_2}^{(2)})$ in R such that ${x_1}^{(1)}{x_2}^{(1)} \cap {x_1}^{(2)} \neq \emptyset$, exists in R a rule $\rho^{(3)} = ({x_1}^{(3)}, {x_2}^{(3)})$ such that ${x_1}^{(1)} \subseteq_S {x_1}^{(3)}$ and ${x_2}^{(2)} \subseteq_S {x_2}^{(3)}$.*
2. *For every pair of rules $\rho^{(1)} = ({x_1}^{(1)}, {x_2}^{(1)})$ and $\rho^{(2)} = ({x_1}^{(2)}, {x_2}^{(2)})$ in R such that ${x_1}^{(1)}{x_2}^{(1)} \cap {x_2}^{(2)} \neq \emptyset$, exists in R a rule $\rho^{(3)} = ({x_1}^{(3)}, {x_2}^{(3)})$ such that ${x_1}^{(1)} \subseteq_S {x_1}^{(3)}$ and ${x_1}^{(2)} \subseteq_S {x_2}^{(3)}$.*

If R satisfies Condition 1 and Condition 2, then R is coherent w.r.t. S.

Note that in the conditions of Proposition 16, the containment $\subseteq_S$ can be replaced with the usual query containment $\subseteq$. Correctness of the proposition follows from Definition 12.

6 Concealing Relationships

In this section, we show that a coherent set of rules indeed enables to conceal relationships. In the first part of the section, we discuss scenarios where a disclosure of concealed information occures and what should be done to avoid such disclosure. In the second part, we prove the effectiveness of rules in concealing relationships.

We start by formally defining the notation of concealing a relationship.

Definition 17 Concealing a Relationship. Consider a document D, a pair of elements e_1 and e_2 in D, and a set R of access-control rules. The relationship (e_1, e_2) is concealed by R if there exist two documents D_1 and D_2, over the element set of D, such that the following three conditions hold.

1. In D_1 there is a path from e_1 to e_2.
2. In D_2 there is no path from e_1 to e_2.
3. $Q(D_1) = Q(D_2)$, for every valid query Q.

6.1 The Singleton-Source Disclosure

The *singleton-source disclosure* occurs when a user can infer that two elements e_1 and e_2 are related, from the following two pieces of information. *(1)* e_2 must be related to an element that is reachable from the root by a path similar to the path from the root to e_1 (in many cases such information can be deduced from the schema). *(2)* There is no element, other than e_1, that is reachable from the root by a path similar to the path to e_1.

A special case of the problem is when there is an access-control rule $\rho = (x_1, x_2)$, defined over a document D, such that when the first expression in the rule is evaluated w.r.t. D the result consists of a single element. That is, $x_1(D)$ is a singleton. The problem is enlighten in the following examples.

Consider a university document that conforms to the schema in Figure 2 and that contains only a single department element. Consider the rule ρ: `for //Department exclude /Course`, which tries to conceals the relationships between departments and courses. A user that is familiar with the schema of the document and knows that the document contains only a single department can infer that every course element in the document is connected to the only department element.

A similar case to the case above is of a rule `for //Department[/Name='CS'] exclude /Course`, when defined over the document in Figure 1. Consider two queries, Q_1 that looks for all the courses in the document and Q_2 that looks for all the courses in departments other than 'CS'. By applying set difference on the answers to Q_1 and Q_2, it is possible to infer what are the courses in the 'CS' department.

As a third example, consider the case where there are several departments, in the university document, but only one department element has course elements below it. In this case, according to the schema shown in Figure 2, every student in the document must be a descendent of the department that has course elements below it.

We now formalize the problem. Let D be a tree document, R be a coherent set of rules, and $\rho = (x_1, x_2)$ be a rule in R. Suppose that e_1 is an element in $x_1(D)$ and e_2 is an element in $x_1x_2(D)$, where the expression x_1x_2 is the concatenation of x_1 and x_2.

Let $\bar{D}$ be the transitive closure of D, and let $prune_R(\bar{D})$ be the pruning of $\bar{D}$ w.r.t. R. We say that an element e' is *disconnected* from e_2, in $prune_R(\bar{D})$, if

$prune_R(\bar{D})$ does not contain a path from e' to e_2 and does not contain a path from e' to a descendent of e_2.

Claim: if e_1 is an ancestor of e_2 then it is disconnected from e_2. We show the correctness of the claim. In $prune_R(\bar{D})$ there is no edge from e_1 to e_2 because, according to ρ, the pair (e_1, e_2) is restricted. The set R is coherent, thus, in $prune_R(\bar{D})$ there is no path from e_1 to e_2 (if there was a path from e_1 to e_2, transitivity would require the existence of an edge between these elements). Similarly, because of reverse transitivity in the definition of a coherent set of rules, there is no path from e_1 to a descendent of e_2.

Let e_{ld} be the lowest disconnected ancestor (*lda*) of e_2, in $prune_R(\bar{D})$. Note that there exists an lda for e_2 since e_1 is a disconnected ancestor of e_2. Let e_{cld} be the child of e_{ld} that is an ancestor of e_2 (or equal to e_2). Let $\rho_{ld} = (x_1^{ld}, x_2^{ld})$ be a rule in R that causes the pair (e_{ld}, e_{cld}) to be restricted. There is such a rule since the edge from e_{ld} to e_{cld} is removed in the pruning of $\bar{D}$ w.r.t. R (recall that there is a path in $prune_R(\bar{D})$ from e_{cld} to e_2).

An *lda alternative* of e_2 is a node e'_{ld} such that the following two conditions hold. First, e'_{ld} is not a descendent of e_1. Secondly, let D' be a document that is created from D by removing the edge (e_{ld}, e_{cld}) from D and adding the edge (e'_{ld}, e_{cld}). Then, $prune_R(\bar{D}) = prune_R(\bar{D}')$. We say that there is a singleton-source disclosure in D w.r.t. R if e_2 does not have an lda alternative.

Intuitively, the singleton-source disclosure does not occur if the document is "full enough" and the rules do not have "too restrictive" sources.

By a document that is full enough we refer to the existence of alternative elements as explain in the next example. If the rules conceal relationships between students and courses then there should be enough course elements so that it will be possible to replace, for a student element, the parent course element. Furthermore, if the rules conceal relationships between students and departments then, for a student element s below a department element d, there should be a course element in a department, other than d, that can replace the parent of s.

We now discuss the problem of rules with a too restrictive source. We say that a rule $\rho = (x_1, x_2)$ has a *singleton source* w.r.t. a document D if $x_1(D)$ is singleton. Note that a rule ρ can have a singleton source w.r.t. one document while not having a singleton source w.r.t. a different document.

Using only rules that do not have a singleton source w.r.t. the document is a heuristic and not a necessary condition, as demonstrated in the following example. Suppose that the two rules ρ_1: `for //Department[/Name='CS'] exclude /Course` and ρ_2: `for //Department[/Name='EE'] exclude /Course` are defined over the document in Figure 1. These two rules have a singleton source w.r.t. the document; however, they can be replaced with an equivalent rule `for //Department[/Name='CS' or /Name='EE'] exclude /Course` that does not have a singleton source w.r.t. the document.

6.2 Concealment by a Coherent Set of Rules

We try now to formalize the notion of concealment by a set of rules. We consider a pair of elements as related if there is a path between them in the document. The following theorems shows that when a rule, in a coherent set of rules, conceals the relationship between two elements e_1 and e_2, indeed, the user cannot tell, by seeing the result of merely valid queries, if there is a relationship between e_1 and e_2.

Theorem 18. *Consider a tree document D and a coherent set of rules R such that there is no singleton-source disclosure in D w.r.t. R. Let $\rho = (x_1, x_2)$ be a rule in R, comprises the two expressions x_1 and x_2, and let e_1 and e_2 be elements that ρ should conceal, i.e., $e_1 \in x_1(D)$ and $e_2 \in x_1x_2(D)$. Suppose that there is a path in D from e_1 to e_2. Then, there is a document D_2 such that, for every locally valid query Q w.r.t. D and R, the following conditions hold:*

1. *Q is valid w.r.t. R and D_2.*
2. *$Q(D) = Q(D_2)$.*
3. *In D there is a path from e_1 to e_2 (the elements are related) and in D_2 there is no path from e_1 to e_2 (the elements are not related).*

Proof. We need to show that there exists D_2 that satisfies the conditions of the theorem.

Let $\bar{D}$ be the transitive closure of D, and $prune_R(\bar{D})$ be the pruning of $\bar{D}$ w.r.t. R. Since there is no singleton-source disclosure in D, e_2 has an lda alternative e'_{ld}. That is, the following two conditions hold. First, e'_{ld} is not a descendent of e_1. Secondly, let D_2 be a document that is created from D by removing the edge (e_{ld}, e_{cld}) from D and adding to D_2 the edge (e'_{ld}, e_{cld}). Then, $prune_R(\bar{D}) = prune_R(\bar{D}_2)$.

We show now that the three conditions of the theorem hold for the documents D and D_2.

We show that the first condition holds. Q is valid w.r.t. R and D. In addition, $prune_R(D) = prune_R(\mathcal{D}_2)$. So, by Corollary 4, Q is valid w.r.t. R and D_2.

From Proposition 3 follows that $Q(D) = Q(D_2)$. Thus, the second condition holds.

The third condition holds because in D there is a path from e_1 to e_2; however, in the modified document D_2 there is no path from e_1 to e_2. This completes the proof.

The next theorem deals with the case where e_1 and e_2 are not related. The theorem has a necessary condition, which requires the existence of an lda alternative of e_2, that seems to be too restrictive. Before presenting the theorem, we provide an example that explains the necessity of the condition.

Suppose that in a university document there is a pair of student element s and department element d that are not related. If below d there are no course elements then, obviously, s cannot be a descendent of d. Hence, it is a necessary condition to have a course element below d in order to have a document in which d and s are related.

Theorem 19. *Consider a tree document D and a coherent set of rules R. Let $\rho = (x_1, x_2)$ be a rule in R, comprises the two expressions x_1 and x_2, and let e_1*

and e_2 be elements that ρ should conceal, i.e., $e_1 \in x_1(D)$ and $e_2 \in x_1x_2(D)$. Suppose that there is no path in D from e_1 to e_2; however, there is an lda alternative e_{lda} of e_2 where e_{lda} is an ancestor of e_1. Then, there is a document D_2 such that, for every valid query Q w.r.t. D and R, the following conditions hold:

1. *Q is valid w.r.t. R and D_2.*
2. *$Q(D) = Q(D_2)$.*
3. *In D there is no path from e_1 to e_2 (the elements are not related); however, in D_2 there is a path from e_1 to e_2 (the elements are related).*

The proof is similar to the proof of Theorem 18.

Theorem 18 and Theorem 19 are showing the effectiveness of access-control rules in the case where the document is a tree. There are similar characterization to the case where the document is not a tree. The main difference between the case of a tree document and the case where the document can be any graph is that for a node we need to consider more than one lda. Actually, the number of lda's to consider, for a node e, is equal to the number of paths from the document root to e. Indeed, in each such path we need to find an appropriate lda. Then, it can be shown that the rules conceal the appropriate relationships for any document.

7 Conclusion

In this work, we presented a rule-based access control for XML. Rules are formulated using XPath and are used for specifying relationships, between elements, that should be concealed from users. We defined the semantics of rules and showed how to test validity w.r.t. a document or w.r.t. a schema. Coherency of a rule set was defined and discussed. A set of rules is coherent if concealed relationships cannot be inferred from non-concealed relationships. We showed how to construct a coherent set of rules. Finally, we showed that when access control is performed using a coherent set of rules, a user cannot deduce the existence of a concealed relationship from the result of merely valid queries.

Traditionally, access control has been performed using views. Rules can be used either instead of views or in addition to views. There are cases where rules can be written concisely while using view is cumbersome. For example, when only a small fraction of the data should be concealed from users. On the other hand, when most of the data should be concealed, defining the policy using view might be easier than using rules. Yet, rules have the advantage that when the set is coherent, it is assured that concealed relationships cannot be inferred from the result of merely valid queries. It is difficult to provide such assurance when using views. Therefore, rule can be used for verifications of views by validating views w.r.t. a set of rules.

For future work, we plan to develop optimization techniques to the different validity tests. Another important issue that we leave for future work is dealing with queries that include aggregate functions.

References

1. E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5(3):290–331, 2002.
2. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery version 1.0: An XML query language, June 2001. Available at `http://www.w3.org/TR/xquery`.
3. J. Clark. XSL Transformations (XSLT) version 1.0. Available at `http://www.w3.org/TR/xslt`, 1999.
4. J. Clark and S. DeRose. XML Path Language (XPath) version 1.0. Available at `http://www.w3.org/TR/xpath`, 1999.
5. E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 5(3):169–202, 2002.
6. A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases*, 2001.
7. W. Fan, C. Chan, and M. Garofalakis. Secure XML querying with security views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Paris (France), 2004.
8. I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*, Yorktown Heights (New York, USA), 2004.
9. A. Gabillon and E. Bruno. Regulating access to xml documents. In *15th Annual Working Conference on Database and Application Security*, Niagara on the Lake (Ontario, Canada), 2002.
10. S. Godik and T. Moses. eXtesible Access Control Markup Language (XACML) Version 1.0. Available at `http://www.oasis-open.org/committees/xacml`, 2003. OASIS Standard.
11. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proceddings of the 28th International Conference on Very Large Data Bases*, Hong Kong (China), 2002.
12. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *Proceedings of the 22nd Symposium on Principles of Databae Systems*, San Diego (California, USA), 2003.
13. G. Grahne and A. Thomo. Query containment and rewriting using views for reular path queries under constraints. In *Proceeding of the 22nd Symposium on Principles of Database Systems*, San Diego (California, USA), 2003.
14. S. Hada and M. Kudo. XML Access Control Language: provisional authorization for XML documents. Available at `http://www.trl.ibm.com/projects/xml/xacl`.
15. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proceedings of the 21st Symposium on Principles of Database Systems*, 2002.
16. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.
17. G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Paris (France), 2004.
18. A. Motro. An access authorization model for relational databases based on algebric manipulation of view definitions. In *Proceedings of the 5th International Conference on Data Engineering*, Los Angeles (California, USA), 1989.

19. M. Murata, A. Tozawa, M. Kudo, and S. Hada. XML access control using static analysis. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washingtion (DC, USA), 2003.
20. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proceedings of the 9th International Conference on Database Theory*, Siena (Italy), 2003.
21. S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Paris (France), 2004.
22. W. Rjaibi and P. Bird. A multi-purpose implementation of mandatory access control in relational database management systems. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto (Ontario, Canada), 2004.
23. A. Rosenthal and E. Scoire. View security as the basis for data warehouse security. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses*, Stockholm (Sweden), 2000.
24. A. Rosenthal and E. Scoire. Administering permissions for distributed data:factoring and automated inference. In *15th Annual Working Conference on Database and Application Security*, Niagara on the Lake (Ontario, Canada), 2001.
25. T. World Wide Web Consortium (W3C). The XML Schema Web page. Available at `http://www.w3c.org/XML/Schema`.

This article was processed using the LaTeX macro package with LLNCS style