Query Translation for XPath-based Security Views

Roel Vercammen*, Jan Hidders, and Jan Paredaens

University of Antwerp, Belgium

Abstract. Since XML is used as a storage format in an increasing number of applications, security has become an important issue in XML databases. One aspect of security is restricting access to data by certain users. This can, for example, be achieved by means of access rules or XML security views, which define projections over XML documents. The usage of security views avoids information leakage that may occur when we use certain access rules. XML views can be implemented by materialized views, but materialization and maintenance of views may cause considerable overhead. Therefore, we study translations from queries on views to equivalent queries on the original XML documents, assuming both the security views and the queries are specified by XPath expressions. Especially, we investigate which XPath fragments are closed under the composition of a view and a query.

1 Introduction

Access control mechanisms are essential for database systems used to store and share sensitive information. XML is used in an increasing number of applications, including those handling confidential information. As a consequence, some standards for XML access control have already emerged, such as XACL [11] and XACML [9]. Furthermore, several approaches for XML access control mechanisms have been proposed in the literature [6, 13, 4]. In most of these approaches, the policies are specified at the DTD level. Fundulaki and Marx developed a framework to compare XML access control mechanisms in terms of XPath [8]. Query answering that incorporates these access control policies can, for example, be performed by computing some (materialized) security view [14, 12] and then evaluating the query against this security view. This ensures that no information is exposed that is not supposed to be seen by the user, since the query is evaluated against an XML tree that contains exactly the information the user is allowed to see. However, the materialization of views causes an overhead that might be avoided if we can translate queries on the view to equivalent queries on the original data, without leaking information on "hidden" nodes [6, 2].

In this paper, we will not introduce a new XML access control mechanism, but instead we assume that security views are defined by path expressions p such

* Roel Vercammen is supported by IWT – Institute for the Encouragement of Innovation by Science and Technology Flanders, grant number 33581.

that access to a node is never granted, except when it is the root node or in the result of p. The obtained XML security views are similar to those of [6] and [12], but we specify them by means of path expressions instead of annotated DTDs. We investigate how to translate queriesst on views to equivalent queries on the original data. Since it is known that some XPath fragments can be evaluated very efficiently [10], we look at a number of XPath fragments to see which of these fragments are closed under the composition of a view and a query.

The rest of the paper is structured as follows. In Section 2 we introduce our XPath-based security views and some preliminary notions. In Section 3 we study the problem of translating queries on XML views to queries on the original (XML) data. We then use these results in Section 4 to examine which XPath fragments are closed under the composition of a view and a query. Finally, we compare our approach to existing query translation mechanisms for queries on XML views in Section 5 and conclude the paper in Section 6.

2 Preliminaries

In this section we introduce some preliminary notions that are used in the rest of our paper. First, we define the data model and the query language that we use in the theoretical exploration of this paper. Next, we introduce our XPath-based security views. Finally, we define the fragments that we investigate.

2.1 Data Model

Our data model is a simplification and abstraction of the full XML data model [7] and restricts itself to the element nodes. First of all, we postulate an infinite set of tag names Σ and an infinite set of nodes $\mathcal{N}$.

Definition 1 (XML Tree). *An* XML tree *is a tuple* $T = (N, \lhd, r, \lambda, \prec)$ *such that* $(N, \lhd, r)$ *is a rooted tree where* $N \subset \mathcal{N}$ *is a finite set of* nodes, $\lhd$ *is the* parent-child relationship, r *is the root,* $\lambda : N \to \Sigma$ *labels nodes with their tag name and* $\prec$ *is a strict total order*[1] *over* N *that represents the* document order *and defines a pre-order tree-walk, i.e., (1) every child is smaller than its parent, and (2) if two nodes are siblings then all descendants of the smaller sibling are smaller than the larger sibling*

In the following we let $\rhd$ denote the inverse relation of $\lhd$, $\lhd^+$ and $\rhd^+$ the transitive closure of resp. $\lhd$ and $\rhd$, and $\lhd^*$ and $\rhd^*$ the transitive and reflexive closure of resp. $\lhd$ and $\rhd$. The set of all XML trees is denoted by $\mathcal{T}$.

2.2 XPath Queries

We now define the set of XPath expressions we consider. We use a syntax in the style of [1] that abstracts from the official syntax [3] and is more suitable

[1] A strict total order is a binary relation that is irreflexive, transitive and total.

for formal presentations. The largest fragment of XPath that we study in this paper, called $\mathcal{P}$, is defined by the following abstract grammar:

$$\begin{aligned} p ::= {} & \epsilon \mid \Uparrow \mid l \mid \downarrow \mid \uparrow \mid \downarrow^* \mid \downarrow^+ \mid \uparrow^* \mid \uparrow^+ \mid \leftarrow^+ \mid \rightarrow^+ \mid \twoheadleftarrow \mid \twoheadrightarrow \mid \\ & p/p \mid p[p] \mid p \cap p \mid p \cup p \mid p - p \end{aligned}$$

where ϵ represents the empty path expression or *self* axis, $l \in \Sigma$ denotes a label test, $\uparrow$ and $\downarrow$ represent the *parent* and *child* axis, $\uparrow^*$, $\uparrow^+$, $\downarrow^*$ and $\downarrow^+$ represent the *ancestor-or-self*, *ancestor*, *descendant-or-self* and *descendant* axis, $\leftarrow^+$ and $\rightarrow^+$ represent the *preceding-sibling* and *following-sibling* axis, $\twoheadrightarrow$ and $\twoheadleftarrow$ represent the *following* and *preceding* axis, $\Uparrow$ represents the document root, p_1/p_2 represents the concatenation of p_1 and p_2, $p_1[p_2]$ represents a path p_1 with a *predicate* p_2 and finally $\cap$, $\cup$ and $-$ represent the set intersection, set union and set difference. For disambiguation, parentheses are added and the concatenation is assumed to have the highest precedence. The label tests of the form $l \in \Sigma$ behave as if they follow the *self* axis. This means that `a/b` corresponds to the conventional XPath expression `self::a/self::b` and *not* to the expression `child::a/child::b` as is the case for the so-called abbreviated XPath syntax. Based on [5] and similar to [1] we define the semantics as follows:

Definition 2 (XPath Semantics). *Given an XML tree $T = (N, \lhd, r, \lambda, \prec)$ we define the* semantics of a path expression p, $\llbracket p \rrbracket_T \subseteq N \times N$ *as follows:*

$$\begin{array}{llll}
\llbracket \Uparrow \rrbracket_T & = \{(n, n') \mid n' = r\} & & \\
\llbracket \uparrow \rrbracket_T & = \rhd & \llbracket \downarrow \rrbracket_T & = \lhd \\
\llbracket \uparrow^* \rrbracket_T & = \rhd^* & \llbracket \downarrow^* \rrbracket_T & = \lhd^* \\
\llbracket \uparrow^+ \rrbracket_T & = \rhd^+ & \llbracket \downarrow^+ \rrbracket_T & = \lhd^+ \\
\llbracket \twoheadleftarrow \rrbracket_T & = \succ - \rhd^+ & \llbracket \twoheadrightarrow \rrbracket_T & = \prec - \lhd^+ \\
\llbracket \leftarrow^+ \rrbracket_T & = \succ \cap (\rhd \circ \lhd) & \llbracket \rightarrow^+ \rrbracket_T & = \prec \cap (\rhd \circ \lhd) \\
\llbracket \epsilon \rrbracket_T & = \{(n, n') \mid n = n'\} & \llbracket l \rrbracket_T & = \{(n, n') \mid n = n' \wedge \lambda(n) = l\} \\
\llbracket p_1/p_2 \rrbracket_T & = \llbracket p_1 \rrbracket_T \circ \llbracket p_2 \rrbracket_T & \llbracket p_1 \cap p_2 \rrbracket_T & = \llbracket p_1 \rrbracket_T \cap \llbracket p_2 \rrbracket_T \\
\llbracket p_1 \cup p_2 \rrbracket_T & = \llbracket p_1 \rrbracket_T \cup \llbracket p_2 \rrbracket_T & \llbracket p_1 - p_2 \rrbracket_T & = \llbracket p_1 \rrbracket_T - \llbracket p_2 \rrbracket_T \\
\llbracket p_1[p_2] \rrbracket_T & = \{(n, n') \mid (n, n') \in \llbracket p_1 \rrbracket_T \wedge \exists n'' : (n', n'') \in \llbracket p_2 \rrbracket_T\} & &
\end{array}$$

Note that "$\circ$" denotes the concatenation of binary relations (and therefore also functions), i.e., $(x, y) \in (f \circ g) \Leftrightarrow \exists z : (x, z) \in f \wedge (z, y) \in g$. This is the reverse of the usual semantics. The length of a path expression p is denoted by $|p|$ and equals the size of the abstract syntax tree of p. Let p be a path expression. We define p^k as the concatenation of k times p, i.e., $p^0 = \epsilon$ and $p^{n+1} = p^n/p$.

Definition 3 (Query). *Let p be a path expression. The query $\mathcal{Q}[p]$ is a function $\mathcal{T} \rightarrow 2^{\mathcal{N}}$, defined as follows: $\forall T = (N, \lhd, r, \lambda, \prec) : (n \in \mathcal{Q}[p](T) \Leftrightarrow (r, n) \in \llbracket p \rrbracket_T)$*

Note that $\forall T \in \mathcal{T} : \llbracket p_1 \rrbracket_T = \llbracket p_2 \rrbracket_T$ implies $\mathcal{Q}[p_1] = \mathcal{Q}[p_2]$, but the reverse does not necessarily hold. For example, we know that $\mathcal{Q}[\downarrow/\uparrow^+] = \mathcal{Q}[\epsilon[\downarrow]]$, but for many XML trees T, $\llbracket \downarrow/\uparrow^+ \rrbracket_T \neq \llbracket \epsilon[\downarrow] \rrbracket_T$.

2.3 XPath-based Security Views

The XPath-based security views that we consider are similar to the XML security views of [6, 12]. However, we define security views by means of path expressions instead of annotating the DTD. Informally, a view defined by a path expression p maps an XML tree, called *input tree*, to an XML tree, called *view tree*, such that the view tree is the projection of the input tree on the nodes selected by p and the root of the input tree. We always include the root of the input tree in order to ensure that the projection yields a valid XML tree instead of a forest.

Definition 4 (View). *Let p be a path expression. The view $\mathcal{V}[p]$ is a function $\mathcal{T} \rightarrow \mathcal{T}$, defined as follows:* $\forall T_1 = (N_1, \lhd_1, r_1, \lambda_1, \prec_1), T_2 = (N_2, \lhd_2, r_2, \lambda_2, \prec_2)$: $\mathcal{V}[p](T_1) = T_2 \Leftrightarrow$

- $N_2 = \{n | (n = r_1) \vee ((r_1, n) \in [\![p]\!]_{T_1})\}$
- $\lhd_2 = \{(m, n) | (m, n \in N_2) \wedge (m \lhd_1^+ n) \wedge (\nexists n' \in N_2 : (m \lhd_1^+ n') \wedge (n' \lhd_1^+ n))\}$
- $r_2 = r_1$
- $\lambda_2 = \{(n, s) | (n \in N_2) \wedge (\lambda_1(n) = s)\}$
- $\prec_2 = \{(m, n) | (m, n \in N_2) \wedge (m \prec_1 n)\}$[2]

Example 1. A governmental organization has to check hospitals and the treatments that are performed by their doctors. For privacy reasons, hospitals are not allowed to transfer *any* information on their patients to this institute. The doctors of this hospital, however, have internally organized the information on treatments by collecting them per patient. Suppose the hospital database is the left tree in Fig. 1 and the government wants the data in the form of the right tree in this figure. We can obtain the right tree using an XPath-based security view, more precisely the view $\mathcal{V}[\downarrow/\texttt{Doctor}/(\epsilon \cup \downarrow^+/\texttt{Treatment}/\downarrow^*)]$ transforms input trees of the left form to view trees of the right form.

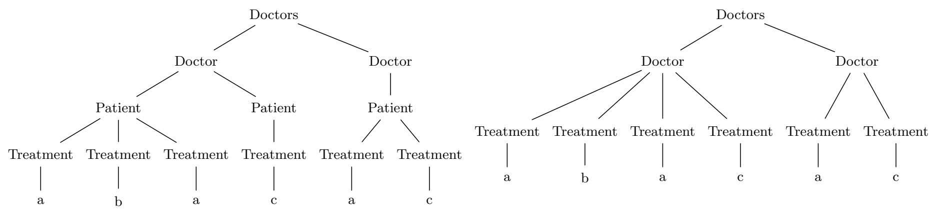


Fig. 1. Input and View Tree of Example 1

Note that the semantics of path expressions on the view tree in terms of the input tree differs from the semantics of the same path expression on the input tree. For example, in Fig. 1 a 'Treatment' node is a child of a 'Doctor' node in the view, while this is not true in the input tree. However, for some axes a it holds that they are "robust under view definition", i.e., $[\![a]\!]_{\mathcal{V}[p](T)} \subseteq [\![a]\!]_T$. The robust axes are $\epsilon, \uparrow^*, \uparrow^+, \downarrow^*, \downarrow^+, \twoheadleftarrow$, and $\twoheadrightarrow$. As we show in Section 3, these axes can easily be translated. Moreover, we can express all other axes in terms of these

[2] This defines a pre-order tree-walk, since $\prec_1$ is a strict total order and $\lhd_2^+ \subseteq \lhd_1^+$.

axes as follows: $\llbracket\downarrow\rrbracket_T = \llbracket\downarrow^+ - \downarrow^+/\downarrow^+\rrbracket_T$, $\llbracket\uparrow\rrbracket_T = \llbracket\uparrow^+ - \uparrow^+/\uparrow^+\rrbracket_T$, $\llbracket\rightarrow^+\rrbracket_T = \llbracket(\uparrow^+ - \uparrow^+/\uparrow^+)/(\downarrow^+ - \downarrow^+/\downarrow^+) \cap \twoheadrightarrow\rrbracket_T$, and $\llbracket\leftarrow^+\rrbracket_T = \llbracket(\uparrow^+ - \uparrow^+/\uparrow^+)/(\downarrow^+ - \downarrow^+/\downarrow^+) \cap \twoheadleftarrow\rrbracket_T$. In some query translations, we first transform path expressions to equivalent expressions only containing robust axes. Since none of our fragments contain the following and preceding axes, we afterwards remove them using following equalities: $\llbracket\twoheadrightarrow\rrbracket_T = \llbracket\uparrow^*/\rightarrow^+/\downarrow^*\rrbracket_T$ and $\llbracket\twoheadleftarrow\rrbracket_T = \llbracket\uparrow^*/\leftarrow^+/\downarrow^*\rrbracket_T$.

2.4 XPath Fragments

We now define the XPath fragments that we study in this paper and discuss some of their properties. These fragments are inspired by the fragments introduced in [1], but we have added sibling axes, intersection, union and set difference. Furthermore, their label tests l correspond to $\downarrow/l$ in our XPath model. Our fragments are defined by the axes that can occur in path expressions and the different operations we can use to combine two path expressions to a new path expression. We consider two groups of fragments. One is defined by a base fragment $\mathcal{X}$ and loosely corresponds to the fragments introduced in [1]; the other is defined by a base fragment $\mathcal{A}$ which is based on the abbreviated syntax [3].
The fragment $\mathcal{X}$ is defined as

$$p ::= \epsilon \mid \Uparrow \mid l \mid \downarrow \mid p/p.$$

We can extend this fragment by adding the parent axis ($\uparrow$), adding the sibling axes ($\leftarrow^+$ and $\rightarrow^+$), and adding the transitive and reflexive closure of axes (i.e., adding $\downarrow^*$ and if $\uparrow$ is in the fragment then also $\uparrow^*$). The three possible extensions can be combined arbitrarily and are respectively denoted by superscripts $\uparrow$, $\leftrightarrow$, and r.
The fragment $\mathcal{A}$ is defined as

$$p ::= \epsilon \mid \Uparrow \mid \downarrow \mid \downarrow/l \mid \downarrow^* \mid \uparrow \mid p/p.$$

All previous fragments can be extended with predicates ([ ]), set intersection ($\cap$), set union ($\cup$), and set difference ($-$). The addition of these extensions is denoted by subscripts.

Some fragments F contain path expressions that are equivalent to path expressions that are not in F. If the addition of a certain operation o to a fragment F does not increase the expressive power of path expressions defined in F then we say that o can be expressed in F. Furthermore, for some fragments F it holds that we cannot in general express an operation o in F, but we can express o if we assume that all path expressions are evaluated against the root. We then say that o can be expressed in queries of F. We now give some expressibility results for the XPath fragments that we have just defined.

Lemma 1. *The following expressibility properties hold for queries, i.e., path expressions evaluated against the root of a tree:*

1. *The union of two path expressions can be expressed in all fragments that can express the set difference and the descendant-or-self axis.*

2. *The intersection of two path expressions can be expressed in all fragments that can express the set difference.*
3. *Predicates can be expressed in all fragments that can express intersection.*
4. *Parent, ancestor and ancestor-or-self axes can be expressed in all fragments that can express intersection and descendant-or-self axes.*

The first two properties also hold for path expressions in general.

Proof. (Sketch)

1. This follows from $[\![p_1 \cup p_2]\!]_T = [\![(\Uparrow/\downarrow^*) - ((\Uparrow/\downarrow^*) - p_1 - p_2)]\!]_T$.
2. This follows from $[\![p_1 \cap p_2]\!]_T = [\![p_1 - (p_1 - p_2)]\!]_T$.
3. We can define a function $e : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$ such that $\mathcal{Q}[p_c/p] = \mathcal{Q}[p_c/e(p, p_c)]$ and its result does not contain predicates if the second argument does not contain predicates. For predicate operations the mapping is defined by $e(p_1[p_2], p_c) = e(p_1, p_c)/(\epsilon \cap e(p_2, p_c/e(p_1, p_c))/\Uparrow/p_c/e(p_1, p_c))$. For all other operations the definition is straightforward, e.g., $e(p_1 \cup p_2, p_c) = e(p_1, p_c) \cup e(p_2, p_c)$, and $e(p_1/p_2, p_c) = e(p_1, p_c)/e(p_2, p_c/e(p_1, p_c))$.
4. Similar to the previous part of this proof, we can define a function $e : \mathcal{P}\times\mathcal{P} \rightarrow \mathcal{P}$ such that $\mathcal{Q}[p_c/p] = \mathcal{Q}[p_c/e(p, p_c)]$ and $e(p, p_c)$ does not contain $\uparrow$, $\uparrow^+$ or $\uparrow^*$ if the second argument does not contain these axes. The mapping for $\uparrow^*$ is defined by $e(\uparrow^*, p_c) = \Uparrow/\downarrow^*[\downarrow^* \cap \Uparrow/p_c]$ and similar mappings can be defined for $\uparrow$ and $\uparrow^+$. The mapping of predicate operations differs from part 3: $e(p_1[p_2], p_c) = e(p_1, p_c)[e(p_2, p_c/e(p_1, p_c)]$. Since predicates can be expressed using intersection, we only need $\cap$ and $\downarrow^*$ axes to express $\uparrow$, $\uparrow^+$, and $\uparrow^*$.

□

From the previous lemma follows that $\mathcal{P}$ has the same expressive power as $\mathcal{X}^{r,\leftrightarrow}_{-}$. We conclude this section by showing that for some queries $\mathcal{Q}[p]$ we know that all nodes in $\mathcal{Q}[p](T)$ are on the same depth in T.

Lemma 2. *For all path expressions $p \in \mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap,-}$ it holds that for all XML trees T all nodes in the result of $\mathcal{Q}[p](T)$ are on the same level $d(p, 0)$, inductively defined as follows:*

$$\begin{array}{llllll}
d(\Uparrow, n) & = 0 & d(\epsilon, n) & = n & d(l, n) & = n \\
d(\downarrow, n) & = n + 1 & d(\uparrow, n) & = n - 1 & d(p_1/p_2, n) & = d(p_2, d(p_1, n)) \\
d(\leftarrow^+, n) & = n & d(\rightarrow^+, n) & = n & d(p_1[p_2], n) & = d(p_1, n) \\
d(p_1 \cap p_2, n) & = d(p_1, n) & d(p_1 - p_2, n) & = d(p_1, n) & &
\end{array}$$

Proof. (Sketch) For all $p \in \mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap,-}$ it can be shown by induction on the length of p that if n_1 is a node in T at depth n and $(n_1, n_2) \in [\![p]\!]_T$ then n_2 is a node at depth $d(p, n)$ in T. □

3 Composing Views and Queries

In this section we study the problem of composing a view and a query to a new query on the input tree that is equivalent to the query on the view tree. We propose two translations, one that can be used to translate path expressions on view trees to path expressions on input trees and one that can only be used to translate queries on view trees to queries on input trees.

The first translaction assumes that all axes in path expressions are robust, such that after each step we can restrict the result of the axis step to the nodes that are in the view tree.

Definition 5. *Let p be a path expression. The function $\tau_p : \mathcal{P} \rightarrow \mathcal{P}$ is defined as follows:*

$$\begin{array}{llll}
\tau_p(\Uparrow) & = \Uparrow & \tau_p(\epsilon) & = \epsilon \\
\tau_p(l) & = l & \tau_p(q_1/q_2) & = \tau_p(q_1)/\tau_p(q_2) \\
\tau_p(q_1[q_2]) & = \tau_p(q_1)[\tau_p(q_2)] & \tau_p(q_1 \cap q_2) & = \tau_p(q_1) \cap \tau_p(q_2) \\
\tau_p(q_1 \cup q_2) & = \tau_p(q_1) \cup \tau_p(q_2) & \tau_p(q_1 - q_2) & = \tau_p(q_1) - \tau_p(q_2) \\
\tau_p(\downarrow^*) & = \downarrow^* \cap \Uparrow/(p \cup \epsilon) & \tau_p(\uparrow^*) & = \uparrow^* \cap \Uparrow/(p \cup \epsilon) \\
\tau_p(\twoheadrightarrow) & = \twoheadrightarrow \cap \Uparrow/(p \cup \epsilon) & \tau_p(\twoheadleftarrow) & = \twoheadleftarrow \cap \Uparrow/(p \cup \epsilon)
\end{array}$$

We now show that this definition can be used to translate path expressions on view trees to path expressions on input trees.

Lemma 3. *Let p, q be path expressions. For all XML trees $T = (N, \lhd, r, \lambda, \prec)$ and $T' = (N', \lhd', r, \lambda', \prec')$ it holds that if $\mathcal{V}[p](T) = T'$ then $[\![\tau_p(q)]\!]_T \cap (N' \times N) = [\![q]\!]_{T'}$ and therefore $\mathcal{V}[p] \circ \mathcal{Q}[q] = \mathcal{Q}[\tau_p(q)]$. Furthermore, $|\tau_p(q)| = O(|p| \times |q|)$*

Proof. (Sketch) This lemma can be shown by induction on $|q|$. Note that since $\uparrow^*, \downarrow^*, \twoheadleftarrow$ and $\twoheadrightarrow$ are robust axes, they can be translated by following the same axis and restricting the result nodes to nodes in T', which are the result nodes of $\Uparrow/(p \cup \epsilon)$. Finally, $|\tau_p(q)| = O(|p| \times |q|)$, since each of the $|q|$ steps is translated into a path expression of size $O(|p|)$. □

The following example illustrates this translation.

Example 2. Consider the view defined in Example 1. Suppose the government wants to have a list of doctors who have performed "operation b". The query on the view can then be written as $\mathcal{Q}[\downarrow/\texttt{Doctor}[\downarrow/\texttt{Treatment}/\downarrow/b]]$. Intuitively, this expression can be translated to $\mathcal{Q}[\downarrow/\texttt{Doctor}[\downarrow^+/\texttt{Treatment}/\downarrow/b]]$, but according to the translation of Definition 5, we obtain the following query[3]:

$$\begin{array}{l}
\mathcal{Q}[(((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon) - \\
\qquad (((\downarrow^* \cap \Uparrow/(((\downarrow^+ - (\downarrow^+/\downarrow^+))/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon)/ \\
\qquad\quad ((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon) \\
\qquad))/ \\
\quad \texttt{Doctor}[(((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon) - \\
\qquad (((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon)/ \\
\qquad\quad ((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon) \\
\qquad))/\texttt{Treatment}/ \\
\qquad (((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon \ \cup \ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon) -
\end{array}$$

[3] Note that in order to use τ_p, we have to rewrite the path expression in the query such that it only contains robust axes.

$(((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon\ \cup\ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon)/$
$((\downarrow^* \cap \Uparrow/((\downarrow/\texttt{Doctor}/(\epsilon\ \cup\ \downarrow^+/\texttt{Treatment}/\downarrow^*)) \cup \epsilon)) - \epsilon)$
$))/\texttt{b}]$
$]$

Using the previous result, we can translate q to $\tau_p(q)$ such that q evaluated against a node n in the view tree and $\tau_p(q)$ evaluated against a node n' in the input tree always return the same result when $n' = n$. This property might be too strong, since for some fragments it can be impossible to find such a translation, but we can find a translation for path expressions evaluated against the root. Therefore, we introduce a second translation, which can only be used if we know that all nodes are on the same level.

Definition 6. *Let p, q be path expressions in $\mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap,-}$. The function $\rho_p : \mathcal{P} \times \mathbb{N} \to \mathcal{P}$ is defined as follows:*

$\rho_p(q, n)$	$= q$	*(if $n \in \{0,1\}$ and $q \in \{\Uparrow, \epsilon\} \cup \Sigma$)*	
$\rho_p(\downarrow, 0)$	$= p$	*(if $d(p,0) > 0$)*	
$\rho_p(\uparrow, 1)$	$= \Uparrow$	*(if $d(p,0) > 0$)*	
$\rho_p(\leftarrow^+, n)$	$= \Uparrow/p \cap (\bigcup_{l=0}^{d(p,0)} \uparrow^l/\leftarrow^+/\downarrow^l)$	$\rho_p(\rightarrow^+, n)$	$= \Uparrow/p \cap (\bigcup_{l=0}^{d(p,0)} \uparrow^l/\rightarrow^+/\downarrow^l)$
$\rho_p(q_1/q_2, n)$	$= \rho_p(q_1, n)/\rho_p(q_2, d(q_1, n))$	$\rho_p(q_1[q_2], n)$	$= \rho_p(q_1, n)[\rho_p(q_2, d(q_1, n))]$
$\rho_p(q_1 \cap q_2, n)$	$= \rho_p(q_1, n) \cap \rho_p(q_2, n)$	$\rho_p(q_1 - q_2, n)$	$= \rho_p(q_1, n) - \rho_p(q_2, n)$

In the cases not covered by the above equations, $\rho_p(q, n) = p_\emptyset$, where $p_\emptyset$ is a shorthand for a path expression that is not satisfiable, e.g., a/b with $a, b \in \Sigma$ and $a \neq b$.

Lemma 4. *If $p, q \in \mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap,-}$ then $\mathcal{Q}[\rho_p(q, 0)] = \mathcal{V}[p] \circ \mathcal{Q}[q]$. Furthermore, $|\rho_p(q, 0)| = O(|p|^2 \times |q|)$.*

Proof. (Sketch) We show by induction on $|q|$ that for all $p, q \in \mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap,-}$ it holds that if n_1 is a node at depth n in $\mathcal{V}[p](T)$ then $(n_1, n_2) \in [\![q]\!]_{\mathcal{V}[p](T)}$ iff $(n_1, n_2) \in [\![\rho_p(q, n)]\!]_T$. Afterwards, it clearly holds that $\mathcal{Q}[\rho_p(q, 0)] = \mathcal{V}[p] \circ \mathcal{Q}[q]$, since a query is always evaluated from the root node. From Lemma 2 we know that all nodes selected by p are on the same level and hence the view tree does not contain nodes at depth 2 or more. Consequently, we can sometimes determine statically whether a certain operation jumps out of the view tree, yielding an empty result set. If $d(p, 0) > 0$ then the set of nodes on level 1 in the view tree is $\mathcal{Q}[p](T)$. Hence following $\downarrow$ from level 0 in the view tree corresponds to evaluating p against the root in the input tree and following $\uparrow$ from level 1 corresponds to $\Uparrow$. The evaluation of $\rightarrow^+$ in the view tree corresponds to getting all following nodes on level $d(p, 0)$ in the input tree and checking whether they are in $\mathcal{Q}[p](T)$. The translation of $\leftarrow^+$ is similar and the translation for the other operations is straightforward and similar to τ_p.

Finally, $|\rho_p(q, 0)| = O(|p|^2 \times |q|)$, since each of the $|q|$ steps is translated into a path expression of size $O(|p|^2)$ (and $O(|p|)$ if q does not contain sibling axes). $\square$

4 Closure of XPath Fragments under View Composition

In the previous section, we defined two translations, τ_p and ρ_p, but as was shown in Example 2, the resulting path expressions can be large[4]. Moreover, the translation introduced set difference, which makes query answering more complex. Therefore, we will investigate in this section for each XPath fragment F, defined in Subsection 2.4, whether it is closed under view composition, i.e., $\forall p_1, p_2 \in F : \exists p_3 \in F : \mathcal{V}[p_1] \circ \mathcal{Q}[p_2] = \mathcal{Q}[p_3]$. Note that the purpose of this paper is to establish whether it is feasible to find translations within the same fragment. Whether an efficient translation exists, is left for further research.

4.1 View Composition for Positive XPath Fragments

The following table summarizes which positive fragments are closed under view composition. Each cell in this table denotes one fragment, i.e., the fragment that can be obtained by adding the operations in the column head to the fragment that is in the row head. If a "$\circ$" is in a certain cell then this fragment is not closed under view composition, otherwise there is a "$\bullet$" to denote that the fragment is closed under view composition. Next to each "$\bullet$" and "$\circ$" symbol there is a number that refers to the theorem that shows the result for this fragment.

		$[\,]$	$\cap$	$\cup$	$[\,],\cap$	$[\,],\cup$	$\cap,\cup$	$[\,],\cap,\cup$
$\mathcal{X}$	$\bullet_1$	$\bullet_1$	$\bullet_1$	$\circ_2$	$\bullet_1$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{\uparrow}$	$\bullet_1$	$\bullet_1$	$\bullet_1$	$\circ_2$	$\bullet_1$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{\leftrightarrow}$	$\circ_4$	$\circ_4$	$\circ_4$	$\circ_2$	$\circ_4$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{r}$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_2$	$\circ_3$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{\uparrow,\leftrightarrow}$	$\circ_4$	$\circ_4$	$\circ_4$	$\circ_2$	$\circ_4$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{\leftrightarrow,r}$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_2$	$\circ_3$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{\uparrow,r}$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_2$	$\circ_3$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{X}^{\uparrow,\leftrightarrow,r}$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_2$	$\circ_3$	$\circ_2$	$\circ_2$	$\circ_2$
$\mathcal{A}$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_3$	$\circ_3$

Theorem 1. *All fragments from $\mathcal{X}$ to $\mathcal{X}^{\uparrow}_{[\,],\cap}$ are closed under view composition.*

Proof. (Sketch) From Lemma 4 we know that if p, q in $\mathcal{X}^{\uparrow}_{[\,],\cap}$ then $\mathcal{Q}[\rho_p(q, 0)] = \mathcal{V}[p] \circ \mathcal{Q}[q]$. Note that in ρ_p parent axes, predicates and intersection only occur in the resulting path expression iff they occur in q or p. $\square$

The following lemma introduces a monotonicity property of path expressions that do not contain set difference. As we will see in the two following theorems, many composed queries do not have this property and hence they cannot be expressed by a query defined by a positive XPath expression, i.e., a path expression without set difference.

Lemma 5. *Let $p \in \mathcal{X}^{\uparrow,\leftrightarrow,r}_{[\,],\cap,\cup}$ and T an XML tree. If T' is T where some nodes are renamed to a new node name that does not occur in p, then $[\![p]\!]_{T'} \subseteq [\![p]\!]_T$.*

[4] The size of the translated path expression is in this case still linear to the product of the sizes of the path expressions of the view and the query.

Proof. (Sketch) We prove this lemma by induction on $|p|$. The semantics of axes in T and T' are the same. The semantics of label tests changes, but for all label tests l that occur in p it holds that $[\![l]\!]_{T'} \subseteq [\![l]\!]_T$. Finally, if $[\![p_1]\!]_{T'} \subseteq [\![p_1]\!]_T$ and $[\![p_2]\!]_{T'} \subseteq [\![p_2]\!]_T$, then for path expressions p of the form $p_1[p_2]$, $p_1 \cap p_2$ or $p_1 \cup p_2$ it clearly holds that $[\![p]\!]_{T'} \subseteq [\![p]\!]_T$. $\square$

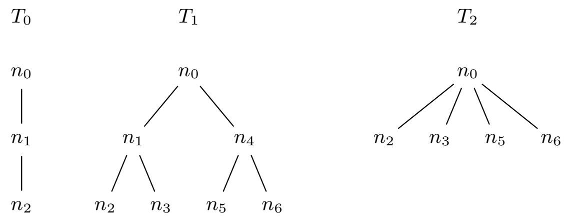


Fig. 2. Counter examples for proofs of Theorems 2, 3, and 4

Theorem 2. *All fragments from* $\mathcal{X}_{\cup}$ *to* $\mathcal{X}^{\uparrow,\leftrightarrow,r}_{[\,],\cap,\cup}$ *are not closed under view composition.*

Proof. (Sketch) Suppose $p \in \mathcal{X}^{\uparrow,\leftrightarrow,r}_{[\,],\cap,\cup}$and $\mathcal{Q}[p] = \mathcal{V}[(\downarrow/a) \;\cup\; (\downarrow/\downarrow)] \circ \mathcal{Q}[\downarrow]$. Let T be the tree T_0 shown in Fig. 2 with $\lambda(n_1) =$ "a" and T' be the same XML tree as T except that n_1 has a label which is different from "a" and all labels for which a test occurs in p. From Lemma 5 it follows that $[\![p]\!]_{T'} \subseteq [\![p]\!]_T$. However, $\mathcal{Q}[p](T) = \{n_1\}$ and $\mathcal{Q}[p](T') = \{n_2\}$. $\square$

Theorem 3. *All fragments from* $\mathcal{X}^r$ *to* $\mathcal{X}^{\uparrow,\leftrightarrow,r}_{[\,],\cap,\cup}$ *and from* $\mathcal{A}$ *to* $\mathcal{A}_{[\,],\cap,\cup}$ *are not closed under view composition.*

Proof. (Sketch) Suppose $p \in \mathcal{X}^{\uparrow,\leftrightarrow,r}_{[\,],\cap,\cup}$ and $\mathcal{Q}[p] = \mathcal{V}[\downarrow^*/\downarrow/a] \circ \mathcal{Q}[\downarrow]$. Let T be the tree T_0 shown in Fig. 2 with $\lambda(n_1) = \lambda(n_2) =$ "a" and T' be the same XML tree as T except that n_1 has a label which is different from "a" and all labels for which a test occurs in p. From Lemma 5 it follows that $[\![p]\!]_{T'} \subseteq [\![p]\!]_T$. However, $\mathcal{Q}[p](T) = \{n_1\}$ and $\mathcal{Q}[p](T') = \{n_2\}$. $\square$

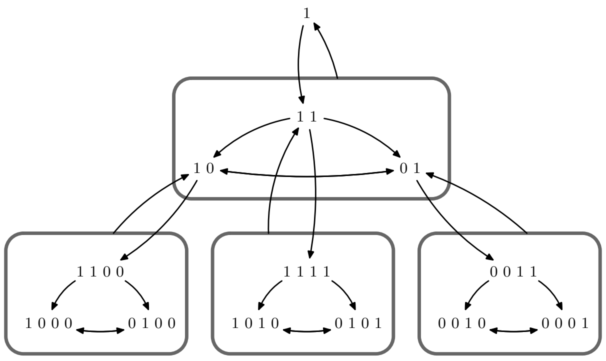
Finally, we show that positive XPath fragments with sibling axes and without set union or recursive axes are also not closed under view composition.

Theorem 4. *All fragments from* $\mathcal{X}^{\leftrightarrow}$ *to* $\mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap}$ *are not closed under view composition.*

Proof. (Sketch) Suppose $p \in \mathcal{X}^{\uparrow,\leftrightarrow}_{\cap}$ and $\mathcal{Q}[p] = \mathcal{V}[\downarrow/\downarrow] \circ \mathcal{Q}[\downarrow/\rightarrow^+]$. Since both the view and the query do not contain label tests, we may assume that p does not contain label tests. Let T_1 be a tree of the form shown in Fig. 2. The tree T_2 in this figure is obviously $\mathcal{V}[\downarrow/\downarrow](T_1)$ and hence $\mathcal{Q}[p](T_1) = \mathcal{Q}[\downarrow/\rightarrow^+](T_2) = \{n_3, n_5, n_6\}$.

From Lemma 2 we know $\mathcal{Q}[p](T_1)$ only contains nodes at depth $d(p,0)$ in T_1. We can encode $\mathcal{Q}[p](T_1)$ as a string of $0's$ and $1's$, where a 0 at position i denotes the absence of, and a 1 the presence of the i^{th} node at level $d(p,0)$ in $\mathcal{Q}[p](T)$. For example, $\mathcal{Q}[p](T_1)$, which is $\{n_3, n_5, n_6\}$, is encoded by 0111.

We show by induction on $|p|$ that $\mathcal{Q}[p](T_1)$ cannot be encoded by 0111. Since queries start from the root, the result of ϵ is encoded by 1. The following diagram shows all possible "state transitions" of $\uparrow, \downarrow, \leftarrow^+$, and $\rightarrow^+$. Note that we omit transitions to empty results, since these states are sink states.



Finally, the intersection combines two of the states in the previous diagram and, as can easily be verified, goes again to one of the states in this diagram. Hence, the encodings for all possible results of path expressions on T_1 in $\mathcal{X}_{\cap}^{\uparrow,\leftrightarrow}$ (without node tests) are listed in this diagram, which does not contain 0111, so p cannot be expressed in $\mathcal{X}_{\cap}^{\uparrow,\leftrightarrow}$ and by Lemma 1 also not in $\mathcal{X}_{[\,],\cap}^{\uparrow,\leftrightarrow}$. $\square$

4.2 View Composition for Fragments with Set Difference

The following table summarizes shows that all fragments with set difference are closed (●) under view composition and next to each "●" symbol there is a number that refers to the theorem that shows the result for this fragment.

	-	$[\,], -$	$\cap, -$	$\cup, -$	$[\,], \cap, -$	$[\,], \cup, -$	$\cap, \cup, -$	$[\,], \cap, \cup, -$
$\mathcal{X}$	$\bullet_5$	$\bullet_5$	$\bullet_5$	$\bullet_7$	$\bullet_5$	$\bullet_7$	$\bullet_7$	$\bullet_7$
$\mathcal{X}^{\uparrow}$	$\bullet_5$	$\bullet_5$	$\bullet_5$	$\bullet_7$	$\bullet_5$	$\bullet_7$	$\bullet_7$	$\bullet_7$
$\mathcal{X}^{\leftrightarrow}$	$\bullet_5$	$\bullet_5$	$\bullet_5$	$\bullet_7$	$\bullet_5$	$\bullet_7$	$\bullet_7$	$\bullet_7$
$\mathcal{X}^{r}$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$
$\mathcal{X}^{\uparrow,\leftrightarrow}$	$\bullet_5$	$\bullet_5$	$\bullet_5$	$\bullet_7$	$\bullet_5$	$\bullet_7$	$\bullet_7$	$\bullet_7$
$\mathcal{X}^{\uparrow,r}$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$
$\mathcal{X}^{\leftrightarrow,r}$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$
$\mathcal{X}^{\uparrow,\leftrightarrow,r}$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$
$\mathcal{A}$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$	$\bullet_6$

Theorem 5. *All fragments from $\mathcal{X}_{-}$ to $\mathcal{X}_{[\,],\cap,-}^{\uparrow,\leftrightarrow}$ are closed under view composition.*

Proof. (Sketch) Let $p, q \in \mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap}$. Using ρ_p we can create a query $\mathcal{Q}[\rho_p(q,0)] = \mathcal{V}[p] \circ \mathcal{Q}[q]$. Note that predicates only occur in $\rho_p(q,0)$ iff they occur in q or p. Since we have set difference, by Lemma 1 we can express intersection and hence predicates. We also can express a parent axis in $\mathcal{X}_{-}$, which can be shown by changing $e(\uparrow, p_c)$ of part 4 of the proof of Lemma 1 to $\Uparrow/(\downarrow)^{d(p_c,0)-1}[\downarrow \cap \Uparrow/p_c]$, because all "candidate parents" are at depth $d(p_c,0)-1$. Finally, the translation of the sibling axes can be expressed in $\mathcal{X}_{-}$. For example, $[\![\rho_p(\leftarrow^{+}, n)]\!]_T = [\![\Uparrow/p - (\Uparrow/p - \leftarrow^{+} - \uparrow/\leftarrow^{+}/\downarrow - \ldots - (\uparrow)^{d(p,0)}/\leftarrow^{+}/(\downarrow)^{d(p,0)})]\!]_T$ as can be verified. □

Theorem 6. *All fragments from $\mathcal{X}^{r}_{-}$ to $\mathcal{X}^{\uparrow,\leftrightarrow,r}_{[\,],\cap,\cup,-}$, and from $\mathcal{A}_{-}$ to $\mathcal{A}_{[\,],\cap,\cup,-}$ are closed under view composition.*

Proof. (Sketch) We use τ_p, for which we know that $\tau_p(q)$ does not contain sibling axes if they do not occur in p and q. Moreover, from Lemma 1 we know that $\uparrow^{*}$, $\cup$, $\cap$ and predicates can be expressed using set difference and $\downarrow^{*}$. □

Theorem 7. *All fragments from $\mathcal{X}_{\cup,-}$ to $\mathcal{X}^{\uparrow,\leftrightarrow}_{[\,],\cap,\cup,-}$ are closed under view composition.*

Proof. (Sketch) We use τ_p to prove this theorem. Since we can express intersection and predicates using set difference, we can eliminate these operations in $\tau_p(q)$. No recursive axes ($\downarrow^{*}, \uparrow^{*}$) are allowed in p and hence there is a depth k such that all nodes deeper than k cannot influence the result of q, since they can simply never be in the view. Hence, we can simulate the $\downarrow^{*}$ axes in $\tau_p(q)$ by $\bigcup_{i=0}^{k} \downarrow^{i}$ for some k of which the value depends on p. From part 4 of Lemma 1 then follows that we also can simulate the $\uparrow^{*}$ axes. Finally, $\tau_p(q)$ does not contain sibling axes if they do not occur in p and q. □

4.3 Summary of Results

All fragments with recursive axes, sibling axes, or set union and without set difference are not closed under view composition, while all other fragments are closed. It can easily be verified that for all fragments that are closed under view composition the size of translated queries for $\mathcal{V}[p] \circ \mathcal{Q}[q]$ is $O(|p| \times |q|)$, except for (1) fragments containing sibling axes and no recursive axes, where the size of the translated query is $O(|p|^2 \times |q|)$, due to the translation of sibling axes in ρ_p, and (2) fragments containing set union and set difference, and no recursive axis steps, where the size of the translated query is also $O(|p|^2 \times |q|)$ (see proof of Theorem 7).

5 Related Work

We briefly discuss two existing approaches for translating queries on XML views to queries on the original (XML) data and compare them to our approach.

Fan, Chan, and Garofalakis introduce the notion of XML security views in [6], where they specify views in terms of normalized DTDs. They present a query translation mechanism for their XPath fragment, which more or less corresponds to our fragment $\mathcal{X}^r_{[\,],\cup}$, augmented with predicates containing boolean operators $(\wedge, \vee, \neg)$ and comparisons of the contents of a node with constant values. They also look at the optimization of the obtained path expressions and their work is mainly geared towards finding efficient translations for path expressions in general, i.e., the translation can use all XPath features, whereas our work mainly focuses on the closure properties of XPath fragments under view composition, to see whether the composed query still has the same characteristics of the view and query.

Benedikt and Fundulaki investigate the specification and composition of XML subtree queries [2]. A subtree query is specified by a path expression and is, just like our XPath-based security views, a projection of nodes from an input tree. While in our views intermediate nodes can be hidden, subtree queries show also all descendants and ancestors. It is for example true that if one node in a view is a child of another node in the same view then the former is also a child of the latter in the input tree, which is not necessarily true in our notion of views. More fragments are closed under the composition of subtree queries than under the composition of our XPath-based views and queries. Note that our notion of views can also be used to express subtree queries: if p is a path expression that specifies a subtree query then this subtree query is equivalent to the view specified by $p/{\downarrow^*}/{\uparrow^*}$.

6 Conclusion and Future Work

In this paper we introduce XPath-based security views that define a projection of a tree that only contains the root and the nodes that are selected by an XPath expression. We investigate how to translate XPath queries on such views to XPath queries on the original trees. More specifically, we show which fragments are closed under such a composition of a view and a query. In future work we plan to investigate the translation of path expressions that start from arbitrary nodes in the view. We also plan to include extra knowledge that we can obtain from DTDs. Moreover, we want to see whether a DTD for the view tree can automatically be derived from the DTD of the input tree and the view definition.

References

1. M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. In *ICDT 2003*, pages 79–95, 2003.
2. M. Benedikt and I. Fundulaki. XML subtree queries: Specification and composition. In *DBPL 2005*, pages 138–153, 2005.
3. A. Berglund, S. Boag, D. Chamberlin, M. Fernández, M. Kay, J. Robie, and J. Siméon. XML path language (XPath) 2.0, W3C working draft, 2005. `http://www.w3.org/TR/xpath20`.
4. E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Trans. Inf. Syst. Secur.*, 5(3):290–331, 2002.

5. D. Draper, P. Frankhauser, M. Fernández, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 and XPath 2.0 formal semantics. W3C Working Draft, 2005.
6. W. Fan, C. Y. Chan, and M. N. Garofalakis. Secure XML querying with security views. In *SIGMOD Conference*, pages 587–598, 2004.
7. M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 data model (XDM), 2005. `http://www.w3.org/TR/xpath-datamodel/`.
8. I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. In *SACMAT 2004*, pages 61–69, 2004.
9. S. Godik and T. Moses, editors. *eXtensible Access Control Markup Language (XACML) Version 1.0.* February 2003.
10. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *PODS 2003*, pages 179–190, San Diego, California, 2003.
11. M. Kudo and S. Hada. XML access control. `http://www.trl.ibm.com/projects/xml/xacl/`.
12. G. Kuper, M. Fabio, and R. Nataliya. Generalized XML security views. In *SACMAT 2005*, pages 77–84, 2005.
13. M. Murata, A. Tozawa, M. Kudo, and S. Hada. XML access control using static analysis. In *CCS*, pages 73–84, 2003.
14. A. Stoica and C. Farkas. Secure XML views. In E. Gudes and S. Shenoi, editors, *DBSec*, volume 256 of *IFIP Conference Proceedings*, pages 133–146. Kluwer, 2002.