THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

XML Security Views Revisited

Citation for published version:
Groz, B, Staworko, S, Caron, A-C, Roos, Y & Tison, S 2009, XML Security Views Revisited. in *Database Programming Languages - DBPL 2009: 12th International Symposium, Lyon, France, August 24, 2009. Proceedings.* vol. 5708, Springer Berlin Heidelberg, pp. 52-67. https://doi.org/10.1007/978-3-642-03793-1_4

Digital Object Identifier (DOI):
10.1007/978-3-642-03793-1_4

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Database Programming Languages - DBPL 2009

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 28. Oct. 2021

XML Security Views Revisited*

Benoît Groz[123], Sławomir Staworko[1],
Anne-Cecile Caron[12], Yves Roos[12], and Sophie Tison[12]

[1] Mostrare project, INRIA Lille Nord-Europe & LIFL (CNRS UMR8022)
[2] University of Lille 1
[3] ENS Cachan

Abstract. In this paper, we revisit the view based security framework for XML without imposing any of the previously considered restrictions on the class of queries, the class of DTDs, and the type of annotations used to define the view. First, we show that the full class of Regular XPath queries is *closed under query rewriting*. Next, we address the problem of constructing a DTD that describes the view schema, which in general needs not be regular. We propose three different methods of *approximating* the view schema and we show that the produced DTDs are *indistinguishable* from the exact schema (with queries from a class specific for each method). Finally, we investigate problems of *static analysis* of security access specifications.

1 Introduction

The wide acceptance of XML as the format for data representation and exchange clearly demonstrates the need for a general and flexible framework of secure access for XML databases. While security specification and enforcement are well established in relational databases, their methods and approaches cannot be easily adapted to XML databases. This is because an XML document stores information not only in its data nodes but also in the way it is structured. Consequently, the problem of secure access to XML databases has its own particular flavor and requires dedicated solutions.

The view-based security framework for XML databases [19] has received an increased attention from both the theoretical and practical angle [3, 9, 4, 17, 22, 18, 5]. It can be summarized as follows:

- The administrator provides the schema of the document together with the security access specification (SAS) defining nodes accessible by the user.
- A virtual view comprising all accessible nodes is defined; the view is never materialized but the user is given its schema.
- Every query over the view is rewritten to an equivalent query over the underlying document and then evaluated.

* This work was partially supported by the Enumeration project ANR-07-blanc-.

The framework is parametrized by the class of queries, typically a fragment of XPath, and the type of formalism used to define the schema with the security access specification, typically an annotated DTD. Previous research often imposed various restrictions in order to facilitate the tasks relevant to the framework. For instance, taking the class of downward XPath queries allows to use the knowledge of the document DTD to benefit the query rewriting [3]. The task can be additionally simplified if the node accessibility is *downward closed*, i.e. all descendants of an inaccessible node are inaccessible as well [1]. The last restriction is also beneficial to constructing the view schema [4]. In fact, without it the set of possible views is not necessarily a regular set of trees, and in particular, needs not be representable with a DTD (cf. Example 3). For similar reasons, in some works only non-recursive DTDs are considered [3, 17].

The aforementioned restrictions may easily limit the versatility of the framework (cf. Example 2). In this paper we lift all the restrictions and revisit three most frequently considered problems: query rewriting, construction of the view schema, and static analysis of SAS. We make the following contributions:

1. We show that Regular XPath is closed under rewriting over XML views. The rewriting is quadratic (combined complexity) whereas a lower exponential bound has been shown for downward XPath queries [5, 6].
2. We propose a novel approach of *approximating* tree languages with a DTD. Basically, a good approximation is a DTD that is indistinguishable from the real schema by means of querying the document. We present three different approximation methods and identify the classes of queries unable to distinguish them from the real schema. We also argue about the optimality of our constructions.
3. We investigate the problem of comparing two SAS and propose two different semantics, *node-* and *query-based*. The first compares the sets of accessible nodes while the second compares the sets of queries that the user can pose on the underlying document through the rewriting process. We provide a detailed complexity analysis of those two notions. We also investigate the problem of verifying accessibility of information.

Because of space limitations we omit some of the proofs. They can be found in Appendix of the full version available online.[4]

Related work In [17] Rassadko extends the query rewriting approach of Fan et al. [3, 4] to XPath queries with ascending axes. However, only non-recursive DTDs are considered and it remains to be seen if this approach can be further extended to handle horizontal axes as well.

To ensure that the view schema can be captured with a DTD, Fan et al. [3, 4] altered the definition of the view for recursive DTDs. Essentially, some inaccessible nodes rather than be removed are obfuscated, i.e. their attributes are removed and names changed to dummy ones. While this approach guarantees the schema to be regular, it clearly limits the relationships one can hide.

[4] `http://researchers.lille.inria.fr/~staworko/papers/staworko-dbpl09.pdf`

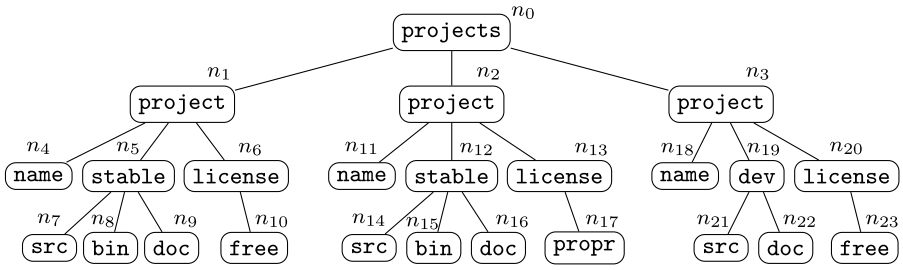
Vercammen et al. [22] propose a query rewriting algorithm for the class of XPath queries with intersection and difference. The set of accessible nodes is defined with a single XPath query Q. While this XPath dialect is commonly available in existing systems, the query rewriting is obtained by incorporating the query Q using the intersection and difference operators. In a straightforward evaluation of XPath, this is equivalent to materializing the view, a behaviour the original motivation tried to avoid.

Our method of verification of information accessibility is inspired by the work of Libkin and Sirangelo [10] where the opposite problem is studied, i.e. verifying that some information is secret. They consider the class of MSO queries, which is more suitable in this context, and show the problem to be decidable for downward-closed accessibility specifications. We recall that $\mathcal{X}Reg$ is known to be properly included in MSO [21], and thus checking whether a property can be expressed in MSO needs not be related to the same check w.r.t. $\mathcal{X}Reg$.

2 Preliminaries

XML Documents. We assume a finite set of node labels Σ and model XML documents with unranked ordered labeled trees T_Σ. Formally, a Σ-tree is a finite structure $t = (N_t, root_t, child_t, next_t, \lambda_t)$, where N_t is a set of nodes, $root_t \in N_t$ is a distinguished root node, $child_t \subseteq N_t \times N_t$ is the parent-child relation, $next_t \subseteq N_t \times N_t$ is the next-sibling relation, and $\lambda_t : N_t \to \Sigma$ is the function assigning to every node its label. We remark that we *do not assume* the set of nodes to be a prefix closed subset of $\mathbb{N}^*$. Moreover, equality of trees should not be confused with isomorphism: two trees are equal if and only if all the elements of their underlying structure are the same, including the node set.

Example 1. Figure 1 contains an example of a tree representing an XML database with information on software development projects. Every project has a name



$$N_{t_0} = \{n_0, n_1, n_2, \ldots\}, \quad root_{t_0} = n_0, \quad \lambda_{t_0} = \{(n_0, \texttt{projects}), (n_1, \texttt{project}), \ldots\},$$
$$child_{t_0} = \{(n_0, n_1), (n_0, n_2), \ldots\}, \qquad next_{t_0} = \{(n_1, n_2), (n_2, n_3), (n_4, n_5), \ldots\}.$$

Fig. 1. Tree t_0.

and a type of license (either free or proprietary). Projects under development come with their source codes and documentation, whereas stable projects have also binaries. □

Regular XPath queries. The class $\mathcal{X}Reg$ of Regular XPath queries [13] over Σ-trees is defined with the following grammar (with a varying over Σ):

$$\begin{aligned}\alpha &::\equiv \mathsf{self} \mid \Downarrow \mid \Uparrow \mid \Rightarrow \mid \Leftarrow\\ f &::\equiv \mathsf{lab}() = a \mid Q \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{not}\, f \mid f \;\mathsf{and}\; f \mid f \;\mathsf{or}\; f\\ Q &::\equiv \alpha \mid [f] \mid Q/Q \mid Q \cup Q \mid Q^*\end{aligned}$$

Essentially, $\mathcal{X}Reg$ query is a regular expression of base axes and filter expressions. Filter expressions are Boolean combinations of node label tests and existential path tests. We define several macros: α^+ is short for α^*/α, $Q[f]$ is $Q/[f]$, $\alpha{::}a$ stands for $\alpha[\mathsf{lab}() = a]$, and $\alpha{::}*$ is simply α, where a is a symbol, Q a query, f a filter expression, and α a base axis or its closure. The semantics of Regular XPath is defined in Fig. 2 (Boolean connectives are interpreted in the usual manner). $[\![Q]\!]_t$ is the binary reachability relation on the nodes of t defined

$$\begin{aligned}[\![\mathsf{self}]\!]_t &= \{(n,n) \mid n \in N_t\},\\ [\![\Downarrow]\!]_t &= \mathit{child}_t,\\ [\![\Uparrow]\!]_t &= \mathit{child}_t^{-1},\\ [\![\Rightarrow]\!]_t &= \mathit{next}_t,\\ [\![\Leftarrow]\!]_t &= \mathit{next}_t^{-1},\end{aligned}$$

$$\begin{aligned}[\![Q_1/Q_2]\!]_t &= [\![Q_1]\!]_t \circ [\![Q_2]\!]_t,\\ [\![Q_1 \cup Q_2]\!]_t &= [\![Q_1]\!]_t \cup [\![Q_2]\!]_t,\\ [\![Q^*]\!]_t &= [\![Q]\!]_t^*,\\ [\![[f]]\!]_t &= \{(n,n) \in N_t \mid (t,n) \models f\}\\ (t,n) &\models \mathsf{lab}() = a \text{ iff } \lambda_t(n) = a,\\ (t,n) &\models Q \text{ iff } \exists n' \in N_t.\, (n,n') \in [\![Q]\!]_t.\end{aligned}$$

Fig. 2. The semantics of Regular XPath.

by the query Q. By $(t,n) \models f$ we denote that the filter f is *satisfied* at the node n of the tree t. We say that a query Q is *satisfied* in the tree t if $(t, root_t) \models Q$. The set of *answers* to a query Q in a tree t is defined as

$$\mathit{Ans}(Q,t) = \{n \in N_t \mid (root_t, n) \in [\![Q]\!]_t\}.$$

For instance, the query $Q_0 = \Downarrow{::}\texttt{project}[\Downarrow{::}\texttt{stable}]/\Downarrow{::}\texttt{name}$ selects (the nodes storing) the names of all stable projects. The set of answers to Q_0 in t_0 (Fig. 1) is $\mathit{Ans}(Q_0, t_0) = \{n_4, n_{11}\}$.

We recall from [13] that $\mathcal{X}Reg$ is closed under inversion, i.e. for every query Q there exists a query Q^{-1} such that $[\![Q^{-1}]\!]_t = [\![Q]\!]_t^{-1}$ for any tree t. Basically, Q^{-1} is obtained by reversing the base axes and the order of composition on the top most level (filter expressions are unchanged). Naturally, $|Q^{-1}| = |Q|$.

DTDs and annotations. A *Document Type Definition* (DTD) over Σ is a function D that maps Σ to regular expressions over Σ. We allow regular expressions defined with the grammar $E ::\equiv \mathsf{empty} \mid a \mid E\text{","}E \mid E\text{"|"}E \mid E^*$, where empty defines the empty sequence, a is a symbol of Σ, E, E is the concatenation, $E|E$ is the union, and E^* is the Kleene closure. In the sequel, we present DTDs using rules of the form $a \to E$ and if for a symbol a the rule is not specified, then $a \to \mathsf{empty}$ is implicitly assumed. The dependency graph of a DTD D over Σ is a directed graph whose node set is Σ and the set of arcs contains (a, b) if $D(a)$ uses the symbol b. A DTD is *recursive* iff its dependency graph is cyclic.

A Σ-tree t *satisfies* D if for every node n with k children $n_1, \ldots, n_k$ (listed in the document order) we have $\lambda_t(n_1) \cdots \lambda_t(n_k) \in L(E)$, where $E = D(\lambda_t(n))$. By $L(D)$ we denote the set of all Σ-trees that satisfy D.

A *security access specification* (SAS) consists of a DTD and an annotation specifying the accessibility of document nodes. Although the DTD and the annotation typically come together, we will often operate on the annotation independently of the DTD. Formally, an *annotation* over Σ is a (possibly partial) function A that maps $\Sigma \times \Sigma$ to $\mathcal{X}Reg$ filter expressions. The size $|A|$ of annotation A is simply the sum of the sizes of all filter expressions used in A.

Annotations define accessibility of nodes as follows. A node b with the parent a is *accessible* w.r.t. A if the filter expression $A(a, b)$ is satisfied at the node b. If $A(a, b)$ is not defined, then accessibility of the parent is used (inheritance). Finally, the root node is always accessible.

Example 2. The DTD D_0 below captures the schema of XML databases described in Example 1.

$\texttt{projects} \to \texttt{project}^*$
$\texttt{project} \to \texttt{name}, (\texttt{stable} \mid \texttt{dev}), \texttt{license}$
$A_0(\texttt{project}, \texttt{stable}) = \mathsf{false}$
$A_0(\texttt{project}, \texttt{dev}) = \mathsf{false}$
$\texttt{license} \to \texttt{free} \mid \texttt{propr}$
$\texttt{stable} \to \texttt{src}, \texttt{bin}, \texttt{doc}$
$A_0(\texttt{stable}, \texttt{src}) = [\Uparrow^*\texttt{::project}/\Downarrow^*\texttt{::free}]$
$A_0(\texttt{stable}, \texttt{doc}) = \mathsf{true}$
$\texttt{dev} \to \texttt{src}, \texttt{doc}$
$A_0(\texttt{dev}, \texttt{src}) = [\Uparrow^*\texttt{::project}/\Downarrow^*\texttt{::free}]$
$A_0(\texttt{dev}, \texttt{doc}) = \mathsf{true}$

The annotation A_0 gives access to all projects but delinquently hides the information whether or not the project is stable (in particular, it hides binaries). Additionally, A_0 hides the source code of all projects developed under proprietary license.

In the tree t_0 from Fig. 1 the root node `projects` is accessible and all nodes `project` are accessible by inheritance. The nodes `name` and `license` with their children are accessible by inheritance as well. A_0 implicitly states that `stable` and `dev` are not to be accessible, and the nodes `bin` are inaccessible by inheritance. On the other hand, A_0 overrides the inheritance for nodes `doc` and makes them accessible. Finally, the accessibility of `src` nodes is conditional: only n_7 and n_{21} are accessible because only those satisfy the specified conditions, $A_0(\texttt{stable}, \texttt{src})$ and $A_0(\texttt{dev}, \texttt{src})$ resp. □

Now, given an annotation A over Σ and a Σ-tree t, the *view* $A(t)$ of t defined by A is the Σ-tree obtained by removing from t all inaccessible nodes. Removing

a node causes its children to be *adopted* by (or *linked* to) the parent of the node. Since the accessibility of every node is well defined and the root is always accessible, the view is a well defined Σ-tree. Figure 3 presents the view $A_0(t_0)$

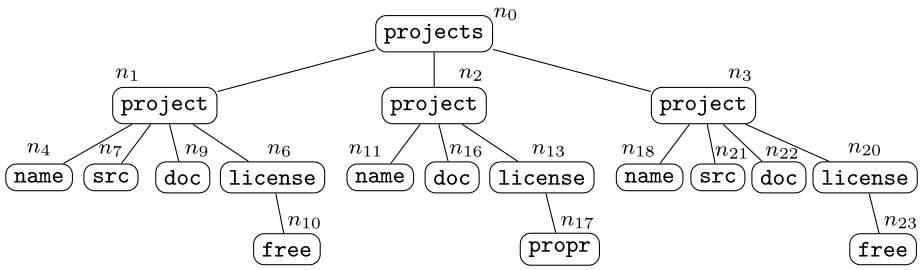


Fig. 3. The view $A_0(t_0)$.

(for t_0 from Fig. 1). We remark that the nodes of view constitute a subset of the nodes of the underlying document, i.e. $N_{A(t)} \subseteq N_t$.

3 Query rewriting over XML views

In this section we show that the class of Regular XPath queries is closed under query rewriting. The rewriting technique for downward queries [3] relies on the knowledge of the DTD. Our rewriting method works independently of the DTD. We remark, however, that the DTD contains important information about the expected structure of the document, and in practical applications, one should take advantage of it in order to improve the efficiency of rewriting. The method uses the fact that accessibility of a node can be defined with a single filter expression (Lemma 1). This filter is used to construct rewritings of the base axes (Lemma 2), which are used to rewrite the user queries.

Lemma 1. *For any annotation A there exists a filter expression f^A_{acc} such that for any tree t, a node $n \in N_t$ is accessible in t w.r.t. A if and only if $(t, n) \models f^A_{\mathrm{acc}}$. Moreover, f^A_{acc} can be constructed in $O(|A|)$ time.*

Proof. By $dom(A)$ we denote the set of pairs of symbols for which A is defined. We begin by defining two filter expressions. The first checks if A defines a filter expression for the current node

$$f_{\mathrm{dom}} \coloneqq \bigvee_{(a,b)\in dom(A)} \big(\mathsf{lab}() = b \text{ and } \Uparrow\!::a\big),$$

and if it is the case, the second filter expression is used to evaluate it

$$f_{\mathrm{eval}} \coloneqq \bigvee_{(a,b)\in dom(A)} \big(\mathsf{lab}() = b \text{ and } \Uparrow\!::a \text{ and } A(a,b)\big).$$

Finally, we restate the definition of accessibility using Regular XPath

$$f^A_{\mathrm{acc}} \coloneqq ([\mathsf{not}\, f_{\mathrm{dom}}]/\Uparrow)^*/[\mathsf{not}(\Uparrow)\ \mathsf{or}\ f_{\mathrm{eval}}]. \qquad \square$$

The filter expression f^A_{acc} plays an important role in the algorithms presented in this paper. Therefore, it is beneficial to optimize it, for instance, by incorporating the knowledge of the DTD. The filter expression $f^{A_0}_{\mathrm{acc}}$ in the presence of D_0 (Example 2) is equivalent to

$\mathsf{lab}()\neq\mathtt{stable}\ \mathsf{and}\ \mathsf{lab}()\neq\mathtt{dev}\ \mathsf{and}\ \mathsf{lab}()\neq\mathtt{bin}\ \mathsf{and}\ (\mathsf{lab}()\neq\mathtt{src}\ \mathsf{or}\ \Uparrow^*::\mathtt{project}/\Downarrow^*::\mathtt{free})$.

Now, we show how to rewrite the base axes.

Lemma 2. *For any annotation A and any $\alpha \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$ there exists a Regular XPath expression R^A_α such that $[\![R^A_\alpha]\!]_t = [\![\alpha]\!]_{A(t)}$ for every tree t. Moreover, $|R^A_\alpha| = O(|A|)$.*

Proof. Essentially, the rewriting R_α defines paths, traversing inaccessible nodes only, from one accessible node to another accessible node in a manner consistent with the axis α. For the vertical axes the task is quite simple:

$$R^A_{\Downarrow} \coloneqq [f^A_{\mathrm{acc}}]/\Downarrow/([\mathsf{not}\, f^A_{\mathrm{acc}}]/\Downarrow)^*/[f^A_{\mathrm{acc}}]$$

and $R^A_{\Uparrow} \coloneqq (R^A_{\Downarrow})^{-1}$. Rewritings of the horizontal axes are slightly more complex and we first define auxiliary filter expressions:

$$f^{\exists}_{\downarrow} \coloneqq ([\mathsf{not}\, f^A_{\mathrm{acc}}]/\Downarrow)^*/[f^A_{\mathrm{acc}}], \quad f^{\varnothing}_{\downarrow} \coloneqq \mathsf{not}\, f^{\exists}_{\downarrow}, \quad f^{\varnothing}_{\rightarrow} \coloneqq (\Rightarrow/[f^{\varnothing}_{\downarrow}])^*/[\mathsf{not}(\Rightarrow)].$$

$f^{\exists}_{\downarrow}$ checks that the current node or any of its descendants is accessible. Conversely, $f^{\varnothing}_{\downarrow}$ checks whether the current node and all of its descendants are inaccessible. Similarly, $f^{\varnothing}_{\rightarrow}$ verifies that only inaccessible can be found among the siblings following the current node and their descendants.

The expression $R^A_{\Rightarrow}$ seeks the next accessible node among the following siblings of the current node and their descendants. However, if there are no such nodes but the parent is inaccessible, then next accessible node is sought among the following siblings of the parent. The last step is repeated recursively if needed.

$$R^A_{\Rightarrow} \coloneqq [f^A_{\mathrm{acc}}]/([f^{\varnothing}_{\rightarrow}]/\Uparrow/[\mathsf{not}\, f^A_{\mathrm{acc}}])^*/\Rightarrow/(\,[(\mathsf{not}\, f^A_{\mathrm{acc}})\ \mathsf{and}\ f^{\varnothing}_{\downarrow}]/\Rightarrow \cup [(\mathsf{not}\, f^A_{\mathrm{acc}})\ \mathsf{and}\ f^{\exists}_{\downarrow}]/\Downarrow/[\neg\Leftarrow])^*/[f^A_{\mathrm{acc}}]$$

and $R^A_{\Leftarrow} \coloneqq (R^A_{\Rightarrow})^{-1}$. We observe that $|R^A_\alpha| = O(|A|)$ for every $\alpha \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$. $\square$

Finally, we state the main result.

Theorem 1. *$\mathcal{X}$Reg is closed under query rewriting, i.e. there exists a function* Rewrite *such that $Ans(Q, A(t)) = Ans(\mathsf{Rewrite}(Q, A), t)$ for any annotation A, any Regular XPath query Q, and any tree t. Moreover,* Rewrite(Q, A) *is computable in time $O(|A| * |Q|)$.*

Proof. The function $\mathsf{Rewrite}(Q, A)$ replaces in Q every occurrence of a base axis $\alpha \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$ with R^A_α. A simple induction over the size of Q shows that $[\![Q]\!]_{A(t)} = [\![\mathsf{Rewrite}(Q, A)]\!]_t$, Lemma 2 handling the nontrivial base cases. Since the root is always accessible, we get $Ans(Q, A(t)) = Ans(\mathsf{Rewrite}(Q, A), t)$. We note that the rewritten query is constructed in time $O(|A| * |Q|)$. □

We observe that the asymptotic complexity of our rewriting method is comparable to that of [4] but it handles a larger class of queries and DTDs. Currently, we investigate automata based formalisms to devise efficient evaluation algorithms. Also, we remark that our rewriting technique can be easily adapted to rewrite Conditional XPath queries, a strict subclass of Regular XPath which we believe to be more scalable.

4 View schema approximation

In general, the user will need a schema of the view to formulate her queries. In this section we address the problem of its construction. For a SAS (D, A) we define $\mathcal{V}_\mathcal{S}(D, A) = \{A(t) \mid t \in L(D)\}$.

Often constructing the view schema is simple. For example, the view schema for D_0 and A'_0 (Example 5) is

$$\mathtt{projects} \to \mathtt{project}^* \quad \mathtt{project} \to \mathtt{name}, \mathtt{doc}, \mathtt{license} \quad \mathtt{license} \to \mathtt{free} \mid \mathtt{propr}$$

In general, however, three potential obstacles may arise when inferring the view schema.

Example 3. Consider the security access specifications (D_1, A_1), (D_2, A_2), and (D_3, A_3) (the last example is due to [9]):

$$D_1 : \mathtt{r} \to \mathtt{a}, \mathtt{b}, \mathtt{c} \qquad \mathtt{b} \to \mathtt{d} \mid \mathtt{e}$$

$$D_2 : \mathtt{r} \to \mathtt{c} \qquad \mathtt{c} \to (\mathtt{a}, \mathtt{c}, \mathtt{b}) \mid \mathsf{empty}$$

$$D_3 : \mathtt{r} \to \mathtt{a}_k \quad \cdots \qquad \mathtt{a}_i \to \mathtt{a}_{i-1}, \mathtt{a}_{i-1} \quad \cdots \qquad \mathtt{a}_0 \to \mathtt{a}$$

$$A_1(\mathtt{r}, \mathtt{a}) = [\Rightarrow::*/\Downarrow::\mathtt{d}], \quad A_1(\mathtt{r}, \mathtt{c}) = [\Leftarrow::*/\Downarrow::\mathtt{d}],$$

$$A_2(\mathtt{r}, \mathtt{c}) = A_2(\mathtt{c}, \mathtt{c}) = \mathsf{false}, \quad A_2(\mathtt{c}, \mathtt{a}) = A_2(\mathtt{c}, \mathtt{b}) = \mathsf{true},$$

$$A_3(\mathtt{r}, \mathtt{a}_k) = \mathsf{false}, \quad A_3(\mathtt{a}_0, \mathtt{a}) = \mathsf{true}.$$

We observe that $\mathcal{V}_\mathcal{S}(D_1, A_1) = \{\mathtt{r}(\mathtt{a}, \mathtt{b}(\mathtt{d}), \mathtt{c}), \mathtt{r}(\mathtt{b}(\mathtt{e}))\}$ although regular cannot be captured with a DTD, $\mathcal{V}_\mathcal{S}(D_2, A_2) = \{\mathtt{r}(\mathtt{a}^k, \mathtt{b}^k) \mid k \in \mathbb{N}\}$ is not a regular tree set, and finally, $\mathcal{V}_\mathcal{S}(D_3, A_3) = \{\mathtt{r}(\mathtt{a}^{2^k})\}$ requires a DTD of size exponential in $|D_3|$. □

To alleviate the infeasibility of constructing a DTD representing the view schema, we investigate approximation methods.

4.1 Intermediate representation

The reason why $\mathcal{V}_\mathcal{S}(D_1, A_1)$ cannot be represented with a DTD is the correlation between the filter expressions $A_1(\mathtt{r}, \mathtt{a})$ and $A_1(\mathtt{r}, \mathtt{c})$, one is satisfied if and only if

the other is. A simple way to circumvent this problem is to consider only *simple annotations*, i.e. annotations using only the constant filter expressions true and false. It can be easily shown that the schema of a view defined by a simple annotation on a non-recursive DTD can be always captured with a DTD.

In the reminder of this section, we adapt the restriction to simple annotations. A comprehensive way to address the problem of correlation between filter expressions would be using a larger, more expressive class of schemas, for instance EDTDs [15, 12]. EDTDs allow defining different content models (types) for the same symbol. Then, the correlations between filter expressions are easily captured with different types. For instance, an EDTD defining $\mathcal{V}_S(D_1, A_1)$ is

$$\mathtt{r} \to (\mathtt{a}, \mathtt{b}_{(1)}, \mathtt{c}) \mid \mathtt{b}_{(2)} \qquad \mathtt{b}_{(1)} \to \mathtt{d} \qquad \mathtt{b}_{(2)} \to \mathtt{e}$$

Introducing EDTDs would unnecessarily complicate the presentation of our results. We conjecture, however, that our approximation methods generalized to handle arbitrary annotations if we use annotated EDTDs for SAS and EDTD for the schema view.[5]

To represent the exact view schema, we generalize the DTD by allowing the rules to use context-free grammars in place of regular expressions.

Definition 1. *A* generalized DTD *over Σ is a function H that maps Σ to context-free grammars over Σ. A Σ-tree t is* valid *w.r.t. H if for every node n with k consecutive children $n_1, \ldots, n_k$ we have $\lambda_t(n_1) \cdots \lambda_t(n_k) \in L(G)$, where $G = H(\lambda_t(n))$. By $L(H)$ we denote the set of all trees valid w.r.t. H.*

Proposition 1. *For every DTD D and every simple annotation A, there exists a generalized DTD $H_{(D,A)}$ such that $L(H_{(D,A)}) = \mathcal{V}_S(D, A)$. Moreover, $H_{(D,A)}$ can be computed in time $O(|D|)$.*

We also remark that for every generalized DTD H there exists a DTD D and a simple annotation A such that $\mathcal{V}_S(D, A) = L(H)$. Thus, from now on we focus on generalized DTDs. Recall that testing regularity of a context-free language is undecidable [8]. Consequently,

Proposition 2. *It is undecidable to test whether a generalized DTD is equivalent to some DTD.*

4.2 Indistinguishability of approximation

In our opinion, the main purpose of the schema is to guide the user in her attempt to formulate a meaningful query, and an approximation of the schema should be judged from this perspective. Consequently, we propose the following notion to assert the suitability of an approximation.

[5] The schema of a view defined by an annotated EDTD cannot be always captured with a EDTD but it can be shown that this is the case when the annotated EDTD is non-recursive.

Definition 2. *We say that two sets L_1 and L_2 of Σ-trees are* indistinguishable *by a class $\mathcal{C}$ of queries, denoted $L_1 \approx_{\mathcal{C}} L_2$, if for every $Q \in \mathcal{C}$ the query Q is satisfied by a tree in L_1 if and only if it is satisfied by a tree in L_2.*

In the sequel, instead of $L(H) \approx_{\mathcal{C}} L(D)$ we write $H \approx_{\mathcal{C}} D$. Also, we consider different subclasses of Regular XPath queries obtained by restricting the use of axes, filter expressions, and negation. By $\mathcal{X}Reg(\mathcal{A})$ we denote the class of Regular XPath queries using axes from $\mathcal{A}$. Using self and node label tests is always allowed, but using existential path test (negation) is allowed only if $\mathcal{A}$ contains $[\,]$ (not respectively). This way $\mathcal{X}Reg(\Downarrow, \Uparrow, \Rightarrow, \Leftarrow, [\,], \mathsf{not})$ denotes the class of all Regular XPath queries. We remark that $\mathcal{X}Reg(\mathcal{A})$ allows the use of all regular operators, including transitive closure and union. Thus, for instance, $\mathcal{X}Reg(\Downarrow^*) \subsetneq \mathcal{X}Reg(\Downarrow^+) \subsetneq \mathcal{X}Reg(\Downarrow)$.

Example 4. Let $H_2 = H_{(D_2,A_2)}$ for (D_2, A_2) from Example 3 and recall that $L(H_2) = \{r(\mathtt{a}^n\mathtt{b}^n) \mid n \in \mathbb{N}\}$. Consider the following three DTDs:

$$D_2^i : \mathtt{r} \to (\mathtt{a}, \mathtt{b})^* \qquad D_2^{ii} : \mathtt{r} \to \mathtt{a}^*, \mathtt{b}^* \qquad D_2^{iii} : \mathtt{r} \to (\mathtt{a} \mid \mathtt{b})^*$$

We observe that H_2 is indistinguishable from: (i) D_2^i by $\mathcal{C}_1 = \mathcal{X}Reg(\Downarrow, \Uparrow, [\,], \mathsf{not})$; (ii) D_2^{ii} by $\mathcal{C}_2 = \mathcal{X}Reg(\Downarrow, \Uparrow, \Rightarrow^+, \Leftarrow^+, [\,])$; (iii) D_2^{iii} by $\mathcal{C}_3 = \mathcal{X}Reg(\Downarrow)$. □

4.3 Three approximations

The only difference between generalized DTDs and standard DTDs is the use of context-free grammars instead of regular expressions. Consequently, we base our approximation techniques on existing methods that approximate context-free languages with regular expressions. We remark that it remains a nontrivial task to choose those methods that guarantee the approximated DTD to be indistinguishable by a possibly large class of queries. Later, we argue that the presented methods are optimal.

The first method is due to Parikh [16]. We fix an ordering of the alphabet $\Sigma = \{a_1, \ldots, a_n\}$ and define $\phi(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$, where $|w|_{a_i}$ denotes the number of occurrences of the symbol a_i in the word w. Parikh has shown that for every context-free grammar G there exists a regular expression E such that $\{\phi(w) \mid w \in L(G)\} = \{\phi(w) \mid w \in L(E)\}$, i.e. E defines a set of words obtained by some permutation of words of G. For instance, the expression for the context-free language $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$ is $(\mathtt{a}, \mathtt{b})^*$. In this paper, by $P(G)$ we denote the *Parikh* regular expression obtained with the method described in [7]. We remark that $|P(G)| = 2^{O(|G|)}$.

The second method uses subwords and was first described in [2]. Formally, u is a *subword* of w, denoted $u \sqsubseteq w$, if $u = u_1 \cdots u_k$ and there exist $v_0, v_1, \ldots, v_k \in \Sigma^*$ such that $v_0 u_1 v_1 \cdots v_{k-1} u_k v_k = w$. [2] shows that for every CFG G one can construct a regular expression $G{\downarrow}$ such that $L(G{\downarrow}) = \{u \mid \exists w \in L(G).\, u \sqsubseteq w\}$. Also this method produces an exponential regular expression, i.e. $|G{\downarrow}| = 2^{O(|G|)}$

The last method is very simple and for grammar G it constructs the expression $(alph(G))^*$, where $alph(G)$ is the set of all symbols used in words of G and can be easily obtained from a trimmed copy of G. Naturally, $|alph(G)| = O(|G|)$.

Definition 3. *Let H be a generalized DTD.*

(i) The Parikh approximation *of H is a DTD D_H^{P} defined as $D_H^{\mathrm{P}}(a) = P(H(a))$ for every $a \in \Sigma$.*
(ii) The subword approximation *of H is a DTD $D_H^{\downarrow}$ defined as $D_H^{\downarrow}(a) = H(a)\downarrow$ for every $a \in \Sigma$.*
(iii) The subset approximation *of H is a DTD D_H^{S} defined as $D_H^{\mathrm{S}}(a) = alph(H(a))^*$ for every $a \in \Sigma$.*

The approximations are illustrated in Example 4 where $D_2^i = D_{H_2}^{\mathrm{P}}$, $D_2^{ii} = D_{H_2}^{\downarrow}$, and $D_2^{iii} = D_{H_2}^{\mathrm{S}}$. The subset approximation of $\mathcal{V}_{\mathcal{S}}(D_3, A_3)$ is $\mathtt{r} \to \mathtt{a}^*$. We note that $|D_H^{\mathrm{P}}| = 2^{O(|H|)}$, $|D_H^{\downarrow}| = 2^{O(|H|)}$, and $|D_H^{\mathrm{S}}| = O(|H|)$. Finally, we identify the classes of queries relevant to each approximation method.

Theorem 2. *For any generalized DTD H we have*

(i) H and D_H^{P} are indistinguishable by $\mathcal{C}_1 = \mathcal{X}Reg(\Downarrow, \Uparrow, [\,], \mathsf{not})$.
(ii) H and $D_H^{\downarrow}$ are indistinguishable by $\mathcal{C}_2 = \mathcal{X}Reg(\Downarrow, \Uparrow, \Rightarrow^+, \Leftarrow^+, [\,])$.
(iii) H and D_H^{S} are indistinguishable by $\mathcal{C}_3 = \mathcal{X}Reg(\Downarrow)$.

We also remark that the subword and the subset methods construct a superset of the real schema. More precisely, $L(H) \subseteq L(D_H^{\downarrow}) \subseteq L(D_H^{\mathrm{S}})$. As for Parikh approximation, D_H^{P} correctly characterizes H if we consider unordered trees.

4.4 Optimality

First, we note that it seems rather unfeasible to construct methods yielding approximations indistinguishable for classes of queries larger than $\mathcal{C}_1$ or $\mathcal{C}_2$.

Proposition 3. *There exists a generalized DTD H for which there is no DTD indistinguishable from H by $\mathcal{X}Reg(\Downarrow, \Rightarrow)$ or $\mathcal{X}Reg(\Downarrow, \Rightarrow^+, [\,], \mathsf{not})$.*

We remark that the Parikh and subword methods construct DTD exponential in the size of the generalized DTD. It is not very surprising because, as seen in Example 3, $\mathcal{V}_{\mathcal{S}}(D_3, A_3)$ requires a DTD of exponential size. We show, however, that the exponential lower bound holds even when approximating w.r.t. quite strong restrictions of $\mathcal{C}_1$ and $\mathcal{C}_2$.

Theorem 3. *Take any class of queries $\mathcal{C}$ containing $\mathcal{X}Reg(\Downarrow, [\,])$ or $\mathcal{X}Reg(\Downarrow, \Uparrow)$. For all $k \in \mathbb{N}$ there exists a generalized DTD H_k whose size is $O(k)$ and such that any DTD D indistinguishable from H_k by $\mathcal{C}$ is of size $\Omega(2^k)$.*

5 Static analysis of security access specifications

In this section we investigate comparing security access specifications: equivalence and restriction. We assume the DTD to be unchanged which allows to eliminate the factor of comparing DTDs from our analysis. We recall that testing

equivalence and inclusion of DTDs is known to be PSPACE-complete [11]. Also, we use again arbitrary annotations.

We begin with testing the equivalence of two annotations which is essential for optimization of the query rewriting process. Next, we consider the problem of comparing restrictiveness of two SAS. When the administrator modifies the SAS to further restrict the user access to the document, it might be prudent to verify that indeed the user is not permitted access to any new data. To address this problem we propose the notion of *node-based* restriction of SAS, which compares the sets of accessible nodes before and after the change. We observe, however, that this test does not always guarantee that the user cannot obtain more information after the change (cf. Example 6). Consequently, we propose the alternative notion of *query-based* restriction, which compares the sets of queries the user is allowed to evaluate on the underlying document.

5.1 Equivalence

Two annotations are equivalent if they produce exactly the same view for every source document. We emphasise that in the context of security using isomorphism to compare the views defined by two annotations may be error prone: the same shape does not imply that the same data is accessible. Consequently, we base the notion of equivalence on equality of trees.

Definition 4. *Two annotations A_1 and A_2 are* equivalent *in the presence of the DTD D, denoted $A_1 \equiv^D A_2$, if and only if $A_1(t) = A_2(t)$ for every $t \in L(D)$.*

We observe that $A_1(t) = A_2(t)$ if and only if exactly the same nodes of t are accessible w.r.t. A_1 and A_2. Consequently, we use Lemma 1 to relate equivalence of annotations to equivalence of Regular XPath known to be EXPTIME-complete [14, 13].

Theorem 4. *Testing equivalence of annotations is EXPTIME-complete.*

5.2 Node-based comparison

Now, we focus on comparing restrictiveness of two SAS. The first approach to test restriction compares the set of nodes that are accessible in the views defined by the annotations.

Definition 5. *Given two annotations A_1 and A_2 and a DTD D, A_1 is a* node-based restriction *of A_2 in the presence of D, denoted $A_1 \preccurlyeq^D_{nb} A_2$, if $N_{A_1(t)} \subseteq N_{A_2(t)}$ for every $t \in L(D)$.*

Example 5. Recall $S_0 = (D_0, A_0)$ from Example 2 and consider an annotation A'_0 that is obtained from A_0 by making all `src` nodes inaccessible, i.e. $A'_0(\texttt{stable}, \texttt{src}) = A'_0(\texttt{dev}, \texttt{src}) = \mathsf{false}$. Clearly, $A'_0 \preccurlyeq^{D_0}_{nb} A_0$. □

In a manner analogous to Theorem 4, we can relate node-based restriction to containment of Regular XPath queries.

Corollary 1. *Testing node-based restriction of annotations is EXPTIME-complete.*

Given a DTD D over Σ, let $\mathcal{A}nn^D$ be the quotient set of all annotations by $\equiv^D$ (which is an equivalence relation). We observe that by Lemma 1 every equivalence class has a representative that is total, i.e. is defined for every $(a, b) \in \Sigma \times \Sigma$. Interestingly, $(\mathcal{A}nn^D, \leqslant^D_{nb})$ is a Boolean algebra, where $\mathcal{A}nn^D$ is the quotient set of all annotations by $\equiv^D$. The supremum, infimum, and complement operations are defined as: $[A_1 \sqcup A_2](a, b) = A_1(a, b)\,\mathsf{or}\,A_2(a, b)$, $[A_1 \sqcap A_2](a, b) = A_1(a, b)\,\mathsf{and}\,A_2(a, b)$, and $A^{\complement}(a, b) = \mathsf{not}\,A(a, b)$. The top and bottom elements are $A_\top(a, b) = \mathsf{true}$ and $A_\bot(a, b) = \mathsf{false}$. These operations can serve as general tools for manipulating SAS, for instance, merging two SAS.

5.3 Query-based comparison

Sometimes, reducing the amount of nodes in the view may inadvertently reveal some information that previously was hidden. To (roughly) identify the information that is made accessible by the annotation to the user we use this predicate:

$$\mathsf{Public}(D, A, Q) \coloneqq \exists Q' \in \mathcal{X}Reg.\, \forall t \in L(D).\, Ans(Q, t) = Ans(Q', A(t)).$$

Example 6. Recall again $S_0 = (D_0, A_0)$ from Example 2, and consider the query

$$Q = \Downarrow\!::\texttt{projects}/\Downarrow\!::\texttt{project}[\Downarrow\!::\texttt{dev}\,\mathsf{and}\,\Downarrow\!::\texttt{license}/\Downarrow\!::\texttt{free}]$$

that selects projects with free license that are currently under development. One can easily verify that $\mathsf{Public}(D_0, A_0, Q)$ does not hold. In fact, A_0 was deliberately designed to hide the development status of projects.

Now, consider A_0'' obtained from A_0 by denying access to the source code of all projects under development, i.e. $A_0''(\texttt{dev}, \texttt{src}) = \mathsf{false}$. Naturally, $A_0'' \leqslant^D_{nb} A_0$. However, the projects with free license that are currently under development can be selected with the following query

$$Q' = \Downarrow\!::\texttt{projects}/\Downarrow\!::\texttt{project}[\mathsf{not}(\Downarrow\!::\texttt{src})\,\mathsf{and}\,\Downarrow\!::\texttt{license}/\Downarrow\!::\texttt{free}].$$

Clearly, this proves $\mathsf{Public}(D_0, A_0'', Q)$. □

We use Public to compare the information made accessible by the annotations.

Definition 6. *Given two annotations A_1 and A_2 and a DTD D, A_1 is a* query-based restriction *of A_2 in the presence of D, denoted $A_1 \leqslant^D_{qb} A_2$, if for every Regular XPath query Q, $\mathsf{Public}(D, A_1, Q)$ implies $\mathsf{Public}(D, A_2, Q)$.*

We note that $A_1 \leqslant^D_{qb} A_2$ implies $A_1 \leqslant^D_{nb} A_2$ (consider $Q_1 = \Downarrow^*\!::*$). However, the converse is not necessarily true (cf. Example 6). We also remark that $(\mathcal{A}nn, \leqslant^D_{qb})$ needs not to be even a semi-lattice (an example can be found in Appendix).

Now, recall A_0' and A_0 from Example 5. Naturally, $A_0' \leqslant^{D_0}_{qb} A_0$. We observe also that $A_0'(t_0)$ can be seen as a view constructed on top of the view $A_0(t_0)$. Interestingly, this observation leads to an alternative characterization of query-based restriction (cf. Lemma 1).

Lemma 3. *Given two annotations A_1 and A_2 and a DTD D, $A_1 \preccurlyeq^D_{qb} A_2$ if and only if there exists a filter expression $f_{\mathbb{P}}$ such that for every $t \in L(D)$ and every $n \in N_t$, n is accessible in t w.r.t. A_1 if and only if $n \in N_{A_2(t)}$ and $(A_2(t), n) \models f_{\mathbb{P}}$.*

In the sequel, we say that the filter expression $f_{\mathbb{P}}$ *proves* $A_1 \preccurlyeq^D_{qb} A_2$. In Example 5, the filter expression that proves $A'_0 \preccurlyeq^D_{qb} A_0$ is $f_{\mathbb{P}} = \mathsf{lab}() \neq \mathtt{src}$.

Now, we focus on computational properties of query-based restriction. Unfortunately, the problem of testing regular separability of two context-free languages, known to be undecidable [20], can be reduced to testing query-based restriction.

Theorem 5. *Testing query-based restriction is undecidable even for simple annotations.*

Undecidability of testing query-based restriction may limit its applications. However, in certain settings, semi-automatic testing is often acceptable. For instance, we may require the administrator to provide an additional input that helps to *prove* that the restriction holds.

Corollary 2. *Given two annotations A_1 and A_2, a DTD D, and a filter expression f, testing whether f proves $A_1 \preccurlyeq^D_{qb} A_2$ is EXPTIME-complete.*

While Lemma 3 shows that testing query-based restriction is **RE** (recursively enumerable), for non-recursive DTDs there exists a **coRE** characterization.

Lemma 4. *Given a non-recursive DTD D and two annotations A_1 and A_2, $A_1 \preccurlyeq^D_{qb} A_2$ if and only if $A_2(t) = A_2(t')$ implies $A_1(t) = A_1(t')$ for every $t, t' \in L(D)$.*

Naturally, it shows that query-based restriction for non-recursive DTDs is decidable. However, we are able to say more about its complexity.

Theorem 6. *Testing query-based restriction for non-recursive DTDs is in EXPTIME and is PSPACE-hard.*

Finally, we investigate the complexity of testing $\mathsf{Public}(D, A, Q)$, which may be used to verify that the annotation A is allowing direct access to the information defined by Q on the underlying document.

Proposition 4. *Testing query-based restriction can be reduced in polynomial time to* Public*, and vice versa.*

By Theorems 5 and 6 we immediately get.

Corollary 3. *Testing* Public *is undecidable for arbitrary DTDs and is in EXPTIME (and PSPACE-complete) for non-recursive DTDs.*

6 Conclusions and future work

In this paper we have revisited problems central in the view-based XML security framework. We considered the full class of Regular XPath queries and recursive DTDs annotated with Regular XPath filter expressions. We have shown that the expressive power of annotated DTDs is equal to the expressive power of a single Regular XPath query (selecting all the accessible nodes). This result is central in the method of rewriting queries over the view.

In the second part of the paper we have investigated the problem of approximating the view schema with a DTD indistinguishable by a class of queries. We have provided an analysis of approximability, presented three different approximation methods, and shown lower bounds on the size of the view DTD. Figure 4 summarizes the results.

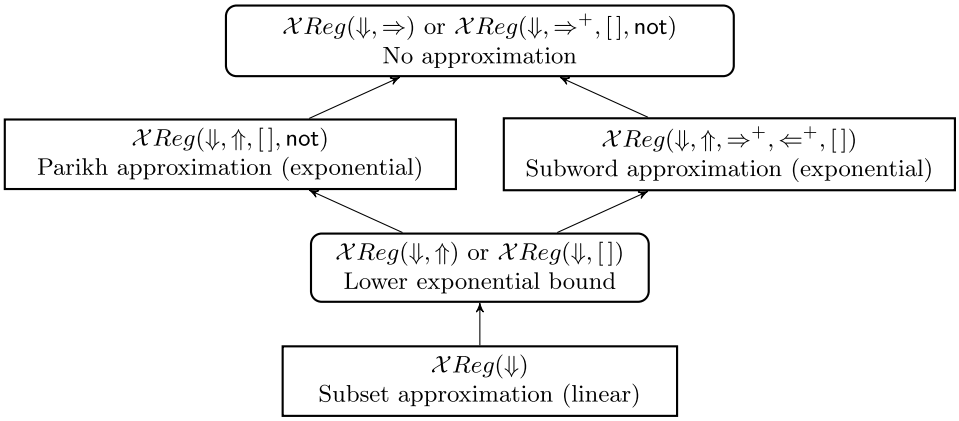


Fig. 4. Summary of approximation results (negative results in round boxes).

Finally, we have considered various problems of statical analysis of SAS. We proposed two different approaches to compare restrictiveness of SAS and analysed their computational implications. We have also studied the relationship between comparison of SAS and verification of information accessibility.

As future work, we plan to investigate the practicality of the proposed solutions by implementing a working system. We believe that in practical settings the proposed algorithms will behave efficiently. We would like to combine the Parikh and subword approximation methods to produce meaningful DTDs of acceptable sizes. We also intend to extend our approach to more expressive schema and query formalisms, for instance EDTDs and n-ary XPath queries. Our preliminary research in this direction is very promising.

References

1. M. Benedikt and I. Fundulaki. XML subtree queries: Specification and composition. In *International Symposium on Database Programming Languages (DBPL)*, pages 138–153, 2005.
2. B. Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.
3. W. Fan, C.-Y. Chan, and M. N. Garofalakis. Secure XML querying with security views. In *ACM SIGMOD International Conference on Management of Data*, pages 587–598, 2004.
4. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. SMOQE: A system for providing secure access to XML. In *International Conference on Very Large Data Bases (VLDB)*, pages 1227–1230. ACM, 2006.
5. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular XPath queries on XML views. In *International Conference on Data Engineering (ICDE)*, pages 666–675, 2007.
6. W. Fan, J. X. Yu, J. Li, B. Ding, and L. Qin. Query translation from XPath to SQL in the presence of recursive DTDs. *VLDB Journal*, 2009. *To appear.*
7. J. Goldstine. A simplified proof of Parikh's theorem. *Discrete Mathematics*, 19(3):235–239, 1977.
8. S. A. Greibach. A note on undecidable properties of formal languages. *Mathematical Systems Theory*, 2(1):1–6, 1968.
9. G. Kuper, F. Massacci, and N. Rassadko. Generalized XML security views. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 77–84. ACM, 2005.
10. L. Libkin and C. Sirangelo. Reasoning about XML with temporal logics and automata. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 97–112. Springer, 2008.
11. W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In *Mathematical Foundations of Computer Science (MFCS)*, pages 889–900, 2004.
12. W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML schema. *ACM Transactions on Database Systems (TODS)*, 31(3):770–813, 2006.
13. M. Marx. XPath with conditional axis relations. In *International Conference on Extending Database Technology (EDBT)*, pages 477–494, 2004.
14. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *International Conference on Database Theory (ICDT)*, pages 315–329. Springer-Verlag, 2003.
15. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 35–46, 2000.
16. R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
17. N. Rassadko. Policy classes and query rewriting algorithm for XML security views. In *IFIP WG 11.3 International Conference on Data and Applications Security*, pages 104–118. Springer, 2006.
18. N. Rassadko. Query rewriting algorithm evaluation for XML security views. In *Secure Data Management (VLDB Workshop)*, pages 64–80. Springer, 2007.
19. A. Stoica and C. Farkas. Secure XML views. In *IFIP WG 11.3 International Conference on Data and Applications Security*, pages 133–146. Kluwer, 2002.

20. T. G. Szymanski and J. H. Williams. Non-canonical parsing. In *14th Annual Symposium on Foundations of Computer Science*, pages 122–129. IEEE, 1973.
21. B. ten Cate and L. Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 251–260, 2008.
22. R. Vercammen, J. Hidders, and J. Paredaens. Query translation for XPath-based security views. In *EDBT Workshops (dataX)*, pages 250–263. Springer, 2006.

A Proof sketches

The sequel we use two alternative notations of regular expressions. On the one hand, we use the notation as defined in the paper, essentially when presenting DTDs. On the other hand, we use a more common notation when working with regular expressions, i.e. we use the following grammar $E ::\equiv \epsilon \mid a \mid E \cdot E \mid E + E \mid E^*$, where ϵ denotes the empty sequence, $a \in \Sigma$ is a single symbol, $E \cdot E$ is the concatenation, $E + E$ is the union, and E^* is the Kleene closure. We believe that using two notation will not introduce confusion.

Proposition 1. *For every DTD D and every simple annotation A, there exists a generalized DTD $H_{(D,A)}$ such that $L(H_{(D,A)}) = \mathcal{V}_{\mathcal{S}}(D, A)$. Moreover, $H_{(D,A)}$ can be computed in time $O(|D|)$.*

Proof. We construct extended context-free grammars, i.e. grammars which allow the use of regular expressions in the rules. Naturally, every extended context-free grammar can be converted to an equivalent standard context-free grammar in linear time.

Let Σ be the alphabet of D and A. We begin by constructing a global set of nonterminals $V = \{N_a^{\mathbb{1}} \mid a \in \Sigma\} \cup \{N_a^{\mathbb{0}} \mid a \in \Sigma\}$. Intuitively, $N_a^{\mathbb{1}}$ ($N_a^{\mathbb{0}}$) produces the words of symbols that can appear in a tree from $\mathcal{V}_{\mathcal{S}}(D, A)$ under a that is accessible (inaccessible resp). The set of productions P for every $a \in \Sigma$ contains the two rules:

1. $N_a^{\mathbb{1}} \to E'_a$, where E'_a is obtained from $D(a)$ by replacing b with $N_b^{\mathbb{0}}$ for every $b \in \Sigma$ such that $A(a, b) = \mathsf{false}$.
2. $N_a^{\mathbb{0}} \to E''_a$, where E''_a is obtained from $D(a)$ by replacing b with $N_b^{\mathbb{0}}$ for every $b \in \Sigma$ such that $A(a, b) = \mathsf{false}$ or $A(a, b)$ is undefined.

$H_{(D,A)}(a) = (\Sigma, V, N_a^{\mathbb{1}}, P)$. $L(H_{(D,A)}) = \mathcal{V}_{\mathcal{S}}(D, A)$ is shown with a simple inductive argument. Finally, we remark that $|H_{(D,A)}| = O(|\Sigma|^2|D|)$, and given the fact that Σ is considered fixed, this yields $|H_{(D,A)}| = O(|D|)$. □

Theorem 2 *For any generalized DTD H we have*

(i) H and D_H^{P} are indistinguishable by $\mathcal{C}_1 = \mathcal{X}Reg(\Downarrow, \Uparrow, [\,], \mathsf{not})$.
(ii) H and $D_H^{\downarrow}$ are indistinguishable by $\mathcal{C}_2 = \mathcal{X}Reg(\Downarrow, \Uparrow, \Rightarrow^+, \Leftarrow^+, [\,])$.
(iii) H and D_H^{S} are indistinguishable by $\mathcal{C}_3 = \mathcal{X}Reg(\Downarrow)$.

Proof. (i) It can be shown with a simple inductive argument that for every $t \in L(H)$ there exists $t' \in L(D_H^{\mathrm{P}})$ that differs from t only by the relative order of siblings, i.e. $next_{t'}$ is a permutation of $next_t$ whereas $N_{t'} = N_t$, $root_{t'} = root_t$, $child_t = child_{t'}$, and $\lambda_{t'} = \lambda_t$. Since the semantics of the queries in $\mathcal{C}_1$ does not depend on $next_t$, any query $Q \in \mathcal{C}_1$ is satisfied by $t \in L(H)$ if and only if Q is satisfied by the corresponding $t' \in L(D_h^{\mathrm{P}})$. The converse is proven analogously.

(ii) We observe that $L(H) \subseteq L(D_H^{\downarrow})$, and consequently, it suffices to show that for any query $q \in \mathcal{C}_2$ that is satisfied by a $t \in L(D_H^{\downarrow})$ there exists some $t' \in L(H)$ satisfying Q. We remark that, indeed, for every $t \in L(D_H^{\downarrow})$ there exists a $t' \in L(H)$ such that $N_t \subseteq N_{t'}$, $root_r = root_{t'}$, $child_t \subseteq child_{t'}$, $\lambda_t \subseteq \lambda_{t'}$, and $next_t \subseteq next_{t'}^+$. Since every $Q \in \mathcal{C}_2$ does not use negation nor horizontal axes without positive closure, adding subtrees under some nodes of t cannot invalidate Q. Consequently, t' satisfies Q

(iii) By $path(t)$ we denote the set of all descending paths from the root node to any node of t and we extend $path$ to sets of trees in the standard way. We observe, that $L_1 \approx_{\mathcal{C}_3} L_2$ if and only if $path(L_1) = path(L_2)$. Clearly, $path(L(H)) = path(L(D_H^{\mathrm{S}}))$. □

Proposition 3. *There exists a generalized DTD H for which there is no DTD indistinguishable from H by $\mathcal{C}_0 = \mathcal{X}Reg(\Downarrow, \Rightarrow^+, [\,], \mathsf{not})$ or $\mathcal{C}_0' = \mathcal{X}Reg(\Downarrow, \Rightarrow)$.*

Proof. For $\mathcal{C}_0$ we observe that for every tree there exists a query that is satisfied by that tree and isomorphic trees only, i.e. this query characterizes the tree up to isomorphism. For example, for $\mathtt{r}(\mathtt{a}, \mathtt{b})$ the query in $\mathcal{C}_0$ is

$$\mathsf{self}::\mathtt{r}\big[\Downarrow::\mathtt{a}[\mathsf{not}(\Downarrow)]/\Rightarrow^+::\mathtt{b}[\mathsf{not}(\Downarrow)\ \mathsf{and}\ \mathsf{not}(\Rightarrow^+)]\ \mathsf{and}\ \bigwedge_{x\in\Sigma}\mathsf{not}(\Downarrow::x/\Rightarrow^+::\mathtt{a}/\Rightarrow^+::\mathtt{b})\big].$$

Consequently $L_1 \approx_{\mathcal{C}} L_2$ iff $L_1 \equiv L_2$ for any $\mathcal{C}$ containing $\mathcal{C}_0$.

The proof for $\mathcal{C}_0'$ is a bit more intricate. Let H be a generalized DTD such that $L(H) = \{\mathtt{r}(\mathtt{c}, \mathtt{a}^k, \mathtt{b}^k, \mathtt{c}) \mid k \in \mathbb{N}\}$ and assume that there is a DTD D such that $H \approx_{\mathcal{C}_0'} D$.

We observe that $L(D)$ consists of trees of depth 1 since $\Downarrow::*/\Downarrow::*$ is not satisfied by any tree in $L(H)$. Also,

$$D(\mathtt{r}) \subseteq (\epsilon + \mathtt{c})\mathtt{a}^*\mathtt{b}^*(\epsilon + \mathtt{c}) \tag{1}$$

since no tree in $L(H)$ satisfies any of the queries

$$\mathsf{self}::\mathtt{r}/\Downarrow::*/\Rightarrow^+::\mathtt{c}/\Rightarrow^+::\mathtt{c}, \qquad \mathsf{self}::\mathtt{r}/\Downarrow::\mathtt{c}/\Rightarrow^+::\mathtt{c}/\Rightarrow^+::*,$$
$$\mathsf{self}::\mathtt{r}/\Downarrow::\mathtt{a}/\Rightarrow^+::\mathtt{b}/\Rightarrow^+::\mathtt{a}, \qquad \mathsf{self}::\mathtt{r}/\Downarrow::\mathtt{b}/\Rightarrow^+::\mathtt{a}/\Rightarrow^+::\mathtt{b}.$$

Define the following objects

$$R_1 = L(H(\mathtt{r})) = \{\mathtt{c}\mathtt{a}^k\mathtt{b}^k\mathtt{c} \mid k \in \mathbb{N}\},$$
$$R_2 = L(D(\mathtt{r})) \cap L(\mathtt{c}\mathtt{a}^*\mathtt{b}^*\mathtt{c}),$$
$$Q_{n,m} = \mathsf{self}::\mathtt{r}/\Downarrow::\mathtt{c}/(\Rightarrow::\mathtt{a})^n/(\Rightarrow::\mathtt{b})^m/\Rightarrow::\mathtt{c},$$

and note that R_2 is regular (being an intersection of two regular sets).

Given (1), $H \approx_{\mathcal{C}_0} D$ with $Q_{k,k}$ for $k \in \mathbb{N}$ implies $R_1 \subseteq R_2$, since for all $k \in \mathbb{N}$ the query $Q_{k,k}$ is satisfied by a tree in $L(H)$. In a similar way, we can show that for every $w = \mathtt{c}\mathtt{a}^{k_1}\mathtt{b}^{k_2}\mathtt{c} \in R_2$ we have $k_1 = k_2$, i.e. $w \in R_1$. Indeed, if there was some $w = \mathtt{c}\mathtt{a}^{k_1}\mathtt{b}^{k_2}\mathtt{c} \in R_2$ with $k_1 \neq k_2$, the query Q_{k_1,k_2} would be satisfied on D, and thus on H, which is not the case. Consequently, $R_2 = R_1$ is not a regular set, a contradiction. □

Theorem 3. *Take any class of queries $\mathcal{C}$ containing $\mathcal{C}_4 = \mathcal{X}Reg(\Downarrow, [\,])$ or $\mathcal{C}'_4 = \mathcal{X}Reg(\Downarrow, \Uparrow)$. For any $k \in \mathbb{N}$ there exists a generalized DTD H_k such that $|H_k| = O(k)$ and for any DTD D indistinguishable from H_k by $\mathcal{C}$ the size of D is $\Omega(2^k)$.*

Proof. We consider the generalized DTD H_k such that

$$\mathtt{r} \to \mathtt{a}^{2^k} \qquad\qquad \mathtt{a} \to \mathtt{b} \qquad\qquad \mathtt{b} \to \mathtt{b} \mid \mathtt{c}$$

Clearly, H_k can be constructed in a manner such that $|H_k| = O(k)$ (see S_3 in Example 3). Now, let D be any DTD indistinguishable from H_k by $\mathcal{C}_4$. It can be easily shown that

$$\begin{aligned} D(\mathtt{c}) &= \epsilon, & \mathtt{b} + \mathtt{c} \subseteq D(\mathtt{b}) &\subseteq \mathtt{b} + \mathtt{c}^*, \\ D(\mathtt{r}) &\subseteq \mathtt{a}^*, & \mathtt{b} \subseteq D(\mathtt{a}) &\subseteq b + \epsilon, \end{aligned}$$

We claim that: (i) $\mathtt{a}^{2^k} \in L(D(\mathtt{r}))$, and (ii) $L(D(\mathtt{r})) \subseteq \{\mathtt{a}^m \mid 0 \leqslant m \leqslant 2^k\}$. For (i) it suffices to consider the query $\mathsf{self}::\mathtt{r}/Q_1/\cdots/Q_{2^k}$, where

$$Q_i = \mathsf{self}::*[\Downarrow::\mathtt{a}/(\Downarrow::\mathtt{b})^i/\Downarrow::\mathtt{c}].$$

To show (ii) consider the query $\mathsf{self}::\mathtt{r}/Q_1/\cdots/Q_{2^k}$ for any $m > 2^k$. It is not satisfied by any tree in $L(H_k)$ and so it cannot be satisfied by any tree in $L(D)$. Since $L(D(r))$ is a set of words whose length is bounded by 2^k, then the length of the regular expression $D(r)$ must be at least 2^k. We prove the lower bound for $\mathcal{C}'_4$ with the same argument but using $Q_i = \Downarrow::\mathtt{a}/(\Downarrow::\mathtt{b})^i/\Downarrow::\mathtt{c}/(\Uparrow::*)^{i+2}$. □

Theorem 4. *Testing equivalence of annotations is EXPTIME-complete.*

Proof. We relate the problem of equivalence of annotations to equivalence of Regular XPath queries which is known to be EXPTIME-complete [14, 13]. To show membership we observe that $A_1 \equiv^D A_2$ iff the queries $\Downarrow^*::*[f^{A_1}_{\mathrm{acc}}]$ and $\Downarrow^*::*[f^{A_2}_{\mathrm{acc}}]$ are equivalent in the presence of D.

To show EXPTIME-hardness, take two Regular XPath queries Q_1 and Q_2 and a DTD D. Consider the annotations $A_1(a,b) = [Q_1^{-1}/\mathsf{self}::*[\mathsf{not}(\Uparrow)]]$ and $A_2(a,b) = [Q_2^{-1}/\mathsf{self}::*[\mathsf{not}(\Uparrow)]]$. Clearly, Q_1 and Q_2 are equivalent in the presence of D iff $A_1 \equiv^D A_2$. □

Throughout the remaining part of this appendix we use an alternative definition of query-based resriction of annotations.

Proposition 5. *Given two annotations A_1 and A_2 and a DTD D, $A_1 \preccurlyeq^D_{qb} A_2$ iff for every query Q_1 there exists a query Q_2 such that $Ans(Q_1, A_1(t)) = Ans(Q_2, A_2(t))$ for all $t \in L(D)$.*

Proof. We recall the definition of query-based restriction

$$A_1 \preccurlyeq^D_{qb} A_2 \equiv \forall Q \in \mathcal{X}Reg.\, \mathsf{Public}(D, A_1, Q) \Rightarrow \mathsf{Public}(D, A_2, Q),$$

where

$$\mathsf{Public}(D, A, Q) \coloneqq \exists Q' \in \mathcal{X}Reg.\, \mathsf{Rewrite}(Q', A) \equiv^D Q.$$

Our claim is

$$\begin{aligned} &\forall Q \in \mathcal{X}Reg.\, \mathsf{Public}(D, A_1, Q) \Rightarrow \mathsf{Public}(D, A_2, Q) && \text{iff} \\ &\forall Q_1 \in \mathcal{X}Reg.\, \exists Q_2 \in \mathcal{X}Reg.\, Ans(Q_1, A_1(t)) = Ans(Q_2, A_2(t)) \end{aligned}$$

For the *if* part take any Q such that $\mathsf{Public}(D, A_1, Q)$, i.e. there exists some Q_1 such that $\mathsf{Rewrite}(Q_1, A) \equiv^D Q$, i.e. $Ans(Q, t) = Ans(Q_2, A_2(t))$ for all $t \in L(D)$. Take any Q_2 such that $Ans(Q_1, A_1(t)) = Ans(Q_2, A_2(t))$ for every $t \in L(D)$. Clearly, $Ans(Q, t) = Ans(Q_2, A_2(t))$ for every $t \in L(D)$, and thus $\mathsf{Rewrite}(Q_2, A_2) \equiv^D Q$, i.e. $\mathsf{Public}(D, A_2, Q)$.

For the *only if* part take any Q_1 and let $Q'_1 = \mathsf{Rewrite}(Q_1, A_1)$. Naturally, $\mathsf{Public}(D, A_1, Q'_1)$ and thus also $\mathsf{Public}(D, A'_1, Q'_1)$. Hence there exists Q_2 such that $\mathsf{Rewrite}(Q_2, A_2) \equiv^D Q'_1 \equiv^D \mathsf{Rewrite}(Q_1, A_1)$, i.e. $Ans(Q_1, A_1(t)) = Ans(Q_2, A_2(t))$ for all $t \in L(D)$. □

In the following example we show a DTD D for which $(\mathcal{A}nn, \preccurlyeq^D_{qb})$ needs not be a semi-lattice.

Example 7. Consider the DTD $\mathtt{r} \to \mathtt{a}^*, \mathtt{b}, \mathtt{c}, \mathtt{a}^*$ and its two annotations:

$$\begin{aligned} &A_1(\mathtt{r}, \mathtt{a}) = [\Rightarrow^* :: \mathtt{b}], && A_2(\mathtt{r}, \mathtt{a}) = [\mathsf{not} \Rightarrow^* :: \mathtt{b}], \\ &A_1(\mathtt{r}, \mathtt{b}) = \mathsf{false} && A_2(\mathtt{r}, \mathtt{b}) = \mathsf{false} \\ &A_1(\mathtt{r}, \mathtt{c}) = \mathsf{false} && A_2(\mathtt{r}, \mathtt{c}) = \mathsf{false} \end{aligned}$$

We claim that A_1 and A_2 has no $\preccurlyeq^D_{qb}$-supremum, i.e. a unique $\preccurlyeq^D_{qb}$-minimal annotation A such that $A_1 \preccurlyeq^D_{qb} A$ and $A_2 \preccurlyeq^D_{qb} A$. In fact, $\{A_1, A_2\}$ has multiple $\preccurlyeq^D_{qb}$-minimal upper bounds and three examples follow:

$$\begin{array}{lll} A'(\mathtt{r}, \mathtt{a}) = \mathsf{true} & A''(\mathtt{r}, \mathtt{a}) = \mathsf{true} & A'''(\mathtt{r}, \mathtt{a}) = \mathsf{true} \\ A'(\mathtt{r}, \mathtt{c}) = \mathsf{true} & A''(\mathtt{r}, \mathtt{c}) = \mathsf{false} & A'''(\mathtt{r}, \mathtt{c}) = [\mathsf{not}(\Leftarrow :: \mathtt{a} / \Leftarrow :: \mathtt{a})^* / [\mathsf{not} \Leftarrow]] \\ A'(\mathtt{r}, \mathtt{d}) = \mathsf{false} & A''(\mathtt{r}, \mathtt{d}) = \mathsf{true} & A'''(\mathtt{r}, \mathtt{d}) = [\Leftarrow :: \mathtt{c} / (\Leftarrow :: \mathtt{a} / \Leftarrow :: \mathtt{a})^* / [\mathsf{not} \Leftarrow]] \end{array}$$

It can be shown that all of the annotations above are minimal upper bounds of $\{A_1, A_2\}$ yet they are incomparable. Similarly, we can construct a DTD and two annotations having no infimum (w.r.t. $\preccurlyeq^D_{qb}$). □

Lemma 3. *Given two annotations A_1 and A_2 and a DTD D, $A_1 \preccurlyeq^D_{qb} A_2$ if and only if there exists a filter expression $f_{\mathbb{P}}$ such that for every $t \in L(D)$ and every $n \in N_t$, n is accessible in t w.r.t. A_1 iff $n \in N_{A_2(t)}$ and $(A_2(t), n) \models f_{\mathbb{P}}$.*

Proof. For the *if* part, the filter $f_{\mathbb{P}}$ is used to rewrite any query Q_1 to an appropriate query Q_2 using the technique described in the proof of Theorem 1. For the *only if* part, let Q_2 be obtained from the definition of $\preccurlyeq^D_{qb}$ with $Q_1 = \Downarrow^*$. Q_2 can be easily converted to a filter expression $f_{\mathbb{P}} = Q_2^{-1}/\mathsf{self}::*[\neg \Uparrow]$ that is satisfied exactly in the nodes selected by Q_2. Clearly, $f_{\mathbb{P}}$ proves $A_1 \preccurlyeq^D_{qb} A_2$. □

Theorem 5. *Testing query-based restriction of annotations is undecidable.*

Proof. We reduce regular separability of two context-free grammars to query-based restriction. Recall that two context-free grammars G_1 and G_2 over the alphabet Γ are *regularly separable* if there exists a regular set R (over Γ) such that $L(G_1) \subseteq R$ and $L(G_2) \subseteq R^{\complement}$, where $R^{\complement}$ is the complement of R. Checking regular separability of two context-free is know to be undecidable [20].

The reduction constructs a DTD D which defines the set of all (accepting) derivation trees of G_1 and G_2. The annotation A_2 removes all nonterminals from the derivation tree, thus yielding a word of $L(G_1) \cup L(G_2)$. The annotation A_1 works similarly except that it also removes terminals derived from nonterminals of G_2; essentially, it yields only words of $L(G_1)$.

If G_1 and G_2 are separable by a regular set R, then the regular expression describing R can be easily rewritten into a filter expression that proves $A_1 \preccurlyeq^D_{qb} A_2$. Conversely, suppose that $f_{\mathbb{P}}$ proves $A_1 \preccurlyeq^D_{qb} A_2$. $f_{\mathbb{P}}$ is equivalent to a tree MSO formula φ [Bu60], and we remark that φ is interpreted on trees of height 1 only. Therefore, there exists a word MSO formula ψ that captures exactly the words consisting of labels of the consecutive children of the root node. This formula ψ can be converted into a regular expression [TB68] which defines a set separating G_1 and G_2. □

Corollary 2. *Given two annotations A_1 and A_2, a DTD D, and a filter expression f, testing whether f proves $A_1 \preccurlyeq^D_{qb} A_2$ is EXPTIME-complete.*

Proof. We relate this test to testing equivalence of two annotations (Theorem 4). To test that f proves $A_1 \preccurlyeq^D_{qb} A_2$ we take $f' = \mathsf{Rewrite}(f, A_2)$ and define $A'_2(a,b) = A_2(a,b)$ and f'. Clearly, $n \in N_{A'_2(t)}$ if and only if $n \in N_{A_2(t)}$ and $(A_2(t), n) \models f$. Consequently, $A_1 \equiv^D A'_2$ if and only if f proves $A_1 \preccurlyeq^D_{qb} A_2$. Conversely, we take any two A_1 and A_2 and observe that $A_1 \equiv^D A_2$ if and only if $f^{A_2}_{\mathrm{acc}}$ proves $A_1 \preccurlyeq^D_{qb} A_\top$, where $A_\top(a,b) = \mathsf{true}$ for every $a, b \in \Sigma$. □

Lemma 4 and Theorem 6 *Given a non-recursive DTD D and two annotations A_1 and A_2, $A_1 \preccurlyeq^D_{qb} A_2$ if and only if $A_2(t) = A_2(t')$ implies $A_1(t) = A_1(t')$ for every $t, t' \in L(D)$. Testing query-based restriction for non-recursive DTDs is in EXPTIME and is PSPACE-hard.*

Proof. We fix a non-recursive DTD D and two annotations A_1 and A_2 over Σ. Since D is non-recursive, the height of all trees in $L(D)$ is bounded by a constant $d \leqslant |\Sigma|$.

For the *only if* part, let f be the filter that proves $A_1 \preccurlyeq^D_{qb} A_2$ and take any $t, t' \in L(D)$ such that $A_2(t) = A_2(t')$. Therefore, $(A_2(t), n) \models f$ iff $(A_2(t'), n) \models f$ which implies that $N_{A_1(t)} = N_{A_1(t')}$. Since, $A_1(t)$ is constructed from nodes of $A_2(t)$, similarly $A_1(t')$ is constructed from nodes of $A_2(t')$, and $A_2(t) = A_2(t')$, we have $A_1(t) = A_1(t')$.

The *if* part is more intricate and requires introducing some notations. Given a Σ-tree t we decorate every node with a value indicating whether the node is accessible w.r.t. A_1 and A_2. Formally, $t \otimes A_1 \otimes A_2$ is a $(\Sigma \times \{\mathbb{0}, \mathbb{1}\} \times \{\mathbb{0}, \mathbb{1}\})$-tree $(N_t, root_t, child_t, parent_t, \lambda'_t)$, where $\lambda'_t(n) = (\lambda_t(n), x_1, x_2)$ with $x_i = \mathbb{1}$ iff n is accessible in t w.r.t. A_i for $i \in \{1, 2\}$ and $n \in N_t$.

We construct a nondeterministic tree automaton $\mathcal{A}$ that defines the set of all properly decorated trees, i.e. $L(\mathcal{A}) = \{t \otimes A_1 \otimes A_2 \mid t \in L(D)\}$. To this end we first define a Regular XPath query that characterizes $L(\mathcal{A})$. Let f_i be obtained by replacing in $f^{A_i}_{\text{acc}}$ every occurrence of $\mathsf{lab}() = a$ by $\mathsf{lab}() = (a, \_, \_)$ (short for $\bigvee_{x,y \in \{\mathbb{0},\mathbb{1}\}} \mathsf{lab}() = (a, x, y)$). Next, define the following query

$$Q = \mathsf{self}::*\big[\,\mathsf{not}(\Downarrow^*::(\_, \mathbb{0}, \_)[f_1])\ \mathsf{and}\ \mathsf{not}(\Downarrow^*::(\_, \mathbb{1}, \_)[\mathsf{not}(f_1)])\ \mathsf{and}\ \mathsf{not}(\Downarrow^*::(\_, \_, \mathbb{0})[f_2])\ \mathsf{and}\ \mathsf{not}(\Downarrow^*::(\_, \_, \mathbb{1})[\mathsf{not}(f_2)])\big].$$

Clearly t' satisfies Q iff $t' = t \otimes A_1 \otimes A_2$ for some Σ-tree t. Using the technique presented in [CDLV08] we convert Q to a nondeterministic tree automaton[6] $\mathcal{A}_Q$ whose size is exponential in the size of $|Q|$ and such that

$$L(\mathcal{A}_Q) = \{t \mid t \text{ satisfies } Q\}.$$

Finally, $\mathcal{A}$ is the product of $\mathcal{A}_Q$ and the automaton $\mathcal{A}_D$ that recognizes $L(D)$.

Now, we switch to finite word automata that work on linearizations of trees. Correctness of this procedure follows from the the fact that we work with non-recursive DTDs. A linearization of a tree is a sequence of opening and closing tags. For instance, the linearization of the tree $t = a(b, c)$ is $Lin(t) = ab\bar{b}c\bar{c}\bar{a}$.

We construct a NFA $\mathcal{B}$ such that $L(\mathcal{B}) = \{Lin(t) \mid t \in L(\mathcal{A})\}$. Basically, $\mathcal{B}$ simulates within its state a stack of depth at most $d \leqslant |\Sigma|$ ($\mathcal{A}$ is a visibly pushdown automaton). The size of $\mathcal{B}$ is $O(|\mathcal{A}|^{2d})$. We remark that this does not make $|\mathcal{B}|$ a doubly exponential function even if Σ is not assumed to be fixed. In fact, $|\mathcal{A}| = 2^{O(p(|A_1|+|A_2|)\log(|D|))}$ for some polynomial $p(n)$, and consequently, $|\mathcal{B}| = 2^{O(p(|A_1|+|A_2|)\log(|D|)|\Sigma|)}$.

Next, from $\mathcal{B}$ we construct a finite state transducer $\mathcal{B}^*$ as follows:

- $\mathcal{B}^*$ has the same set of states, the same set of initial states, and the same set of final states as $\mathcal{B}$.
- $\mathcal{B}^*$ has transition $q \xrightarrow{\varepsilon} p$ for every transition $q \xrightarrow{(a,\mathbb{0},\mathbb{0})} p$ or $q \xrightarrow{\overline{(a,\mathbb{0},\mathbb{0})}} p$ of $\mathcal{B}$
- $\mathcal{B}^*$ has a transition $q \xrightarrow{a/x_1} p$ for any transition $q \xrightarrow{(a,x_1,\mathbb{1})} p$ of $\mathcal{B}$

[6] The original construction uses an automata model that works on the first-child next-sibling encoding of ranked trees. In our proofs we work with visibly-pushdown automata. We remark that any top-down automaton working on *fcns* encoding can be easily transformed in polynomial time to VPA.

– $\mathcal{B}^*$ has a transition $q \xrightarrow{\bar{a}/x_1} p$ for any transition $q \xrightarrow{\overline{(a,x_1,\mathbb{1})}} p$ of $\mathcal{B}$

We claim that $A_1 \preccurlyeq^D_{qb} A_2$ if the following two conditions are satisfied:

(P_1) $\mathcal{B}$ has no transition of the form $q \xrightarrow{(\_,\mathbb{1},\mathbb{0})} p$.
(P_2) $\mathcal{B}^*$ is functional.

The first condition simply states that $A_1 \preccurlyeq^D_{nb} A_2$ and if it holds, then (P_2) allows to view $\mathcal{B}^*$ as a transducer that converts $A_1(t)$ into $A_2(t)$, i.e. it proves $A_1 \preccurlyeq^D_{qb} A_2$. Later we show how to construct from $\mathcal{B}$ a Regular XPath proof filter.

It can be easily shown that Lemma 4 is a reformulation of the two conditions above. The first condition is easy to test. Functionality of $\mathcal{B}^*$ can be also tested in polynomial time [GI83]. This proves that testing query-based restriction for non-recursive DTDs is in EXPTIME. PSPACE-hardness is proved by a simple reduction of containment of regular expressions [11].

Now, we show that if the conditions P_1 and P_2 are satisfied, then from $\mathcal{B}^*$ we can construct a filter that proves $A_1 \preccurlyeq^D_{qb} A_2$. First, we remove ε-transitions from $\mathcal{B}^*$ (using for instance the forward closure algorithm). W.l.o.g. we can assume that for any two transitions $q_1 \xrightarrow{\theta_1/x_1} p$ and $q_2 \xrightarrow{\theta_2/x_2} p$ of $\mathcal{B}^*$ if $x_1 = \mathbb{1}$, then $x_2 = \mathbb{1}$. In other words, there is set Q_s of *selecting* states of $\mathcal{B}^*$ that the control of $\mathcal{B}$ visits exactly after producing $\mathbb{1}$. Also w.l.o.g., we can suppose that for every state q of $\mathcal{B}^*$, all of its incoming transitions use either opening tags or closing tags. For simplicity, we assume a distinguished root node label `r`, and since we consider non-recursive DTDs, no other node can be labeled with `r`. Next, we construct an automaton $\mathcal{A}^{\blacksquare}$ as follows:

– $\mathcal{A}^{\blacksquare}$ has the same set of states, the same set of initial states, and the same set of final states as $\mathcal{B}$.
– for every transition $q \xrightarrow{a/x} p$ of $\mathcal{B}^*$ the automaton $\mathcal{A}^{\blacksquare}$ contains
  • $q \xrightarrow{a^{\blacksquare}} p$ if $\mathcal{B}^*$ contains a transition $q'' \xrightarrow{\bar{b}/x} q$ for some q'' and b,
  • $q \xrightarrow{a} p$ otherwise.
– for every transition $q \xrightarrow{\bar{a}/x} p$ of $\mathcal{B}^*$ the automaton $\mathcal{A}^{\blacksquare}$ contains
  • $q \xrightarrow{\bar{a}^{\blacksquare}} p$ if $\mathcal{B}^*$ contains a transition $q'' \xrightarrow{\bar{b}/x} q$ for some q'' and b,
  • $q \xrightarrow{\bar{a}} p$ otherwise.

The marker $\blacksquare$ is used to indicate that the previously read symbol was a closing tag. This helps us to determine whether the next opening move is toward the next sibling or the parent. Now, for every $q \in Q_S$ we construct two regular expressions: in_q and out_q such that

$$L(in_q) = \{w \mid q \in \delta_{\mathcal{A}^{\blacksquare}}(q_0, w) \wedge q_0 \in I_{\mathcal{A}^{\blacksquare}}\}$$
$$L(out_q) = \{w \mid \delta_{\mathcal{A}^{\blacksquare}}(q, w) \cap F_{\mathcal{A}^{\blacksquare}} \neq \varnothing\}.$$

The filter expression we are looking for is $\bigcup_{q \in Q_S} \big(W(in_q)[W(out_q)]\big)$, where the transformation W is defined as follows (we remark that symbol $\mathtt{r}^{\blacksquare}$ will never

occur in the constructed regular expressions):

$$W(\mathtt{r}) = \mathtt{r}, \qquad W(\bar{\mathtt{r}}) = W(\bar{\mathtt{r}}^{\blacksquare}) = \Uparrow\!::\mathtt{r},$$
$$W(a) = \Downarrow\!::a, \qquad W(\bar{a}) = \mathsf{self}::a,$$
$$W(a^{\blacksquare}) = \Rightarrow\!::a, \qquad W(\bar{a}^{\blacksquare}) = \Uparrow\!::a,$$
$$W(E_1 + E_2) = W(E_1) \cup W(E_2), \qquad W(E_1E_2) = W(E_1)/W(E_2),$$
$$W(E^*) = \mathsf{self}::* \cup (W(E))^*/W(E). \qquad \square$$

Proposition 4. *Testing query-based restriction can be reduced in polynomial time to* Public*, and vice versa.*

Proof. Take any annotations A, A_1, A_2, any DTD D, and any Regular XPath query Q. First, we observe that $A_1 \preccurlyeq^D_{qb} A_2$ iff $\mathsf{Public}(A_2, D, f^{A_1}_{\mathrm{acc}})$. Also, $\mathsf{Public}(A, D, Q)$ iff $A' \preccurlyeq^D_{qb} A$, where $A'(a, b) = [Q^{-1}/[\mathsf{not}(\Uparrow)]]$ for every $a, b \in \Sigma$. Clearly, the reductions can be performed in linear time. $\square$

Additional bibliography

[Bu60] A. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math*, 6:66–92, 1960.

[CDLV08] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Regular XPath: Constraints, query containment and view-based answering for XML documents. In *International Workshop on Logic in Databases (LiD)*, 2008.

[GI83] E. Gurari and O. Ibarra. A note on finitely-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.

[TB68] J. W. Thatcher and Wright J. B. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.