



**HAL**  
open science

# A Benchmark of Deterministic Automata for Regular XPath Queries

Antonio Al Serhali, Joachim Niehren

► **To cite this version:**

Antonio Al Serhali, Joachim Niehren. A Benchmark of Deterministic Automata for Regular XPath Queries. 2022. hal-03527888v1

**HAL Id: hal-03527888**

**<https://inria.hal.science/hal-03527888v1>**

Preprint submitted on 23 Feb 2022 (v1), last revised 3 Jun 2022 (v4)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A Benchmark of Deterministic Automata for Regular XPath Queries

Antonio Al Serhali and Joachim Niehren

Inria Lille, Université de Lille, France

**Abstract.** We provide a benchmark collection of deterministic automata for regular XPATH queries. For this, we select the subcollection of forward navigational XPATH queries from a corpus that Lick and Schmitz extracted from real world XSLT and XQuery programs, compile them to symbolic stepwise hedge automata (*SSHAs*) and determinize them. Large blowups by automata determinization are avoided by using schema-based determinization. Therefore, we show how to lift a recent algorithm for schema-based determinization from *SHAs* to symbolic *SHAs*. Our collection also provides deterministic symbolic nested word automata, that we obtain by compilation from deterministic symbolic *SHAs*.

**Keywords:** Automata, regular path queries, trees, nested words, XPATH.

## 1 Introduction

Algorithms for validating and querying data trees or other kinds of hierarchically structured documents are often based on deterministic automata [14,2,5]. Determinism is often essential to keep the computational complexity tractable, as well known from the universality and inclusion problems of deterministic automata for trees, unranked trees, hedges, or nested words [4,20]. This observation does also apply to the problem of answering regular path queries on hierarchical documents in streaming mode [7,11], and for the problem of enumerating query answers of document spanners in in-memory mode [8,19].

Collections of deterministic automata for regular queries nested words [12,15,9] are needed for benchmarking algorithm for these problems. But only a few deterministic automata are available so far. Niehren and Sakho provided deterministic nested word automata (NWAs) of decent size for the 10 forward navigational XPATH queries for the XPathMark benchmark [10]. The problem is that no sufficiently general compilation method exists so far, since determinization remains problematic, as already noticed in [6]: the usual determinization algorithm for *NWAs* [3,1,18] quickly leads to a size explosion even for simple regular path queries such as `/a/b`. This difficulty can be circumvented when avoiding top-down determinism of *NWAs* in favor of bottom-up and left-to-right determinism. This can be done by either restricting *NWAs* or more easily by using stepwise hedge automata (*SHAs*) [16]. But even the determinization of *SHAs* may lead to unreasonably large automata. As shown by recent work of the authors [17], schema-based determinization might be the key to solve this problem. We confirm this conjecture in the present paper on a practical benchmark.

We start with the collection of 21000 XPATH queries that Lick and Schmitz [13] extracted from real-world XQUERY and XSLT programs available on the Web. The purpose of this corpus is to reflect the form and distribution of XPATH queries in practical applications. The much smaller XPathMark benchmark, in contrast, [10] focuses on functionality testing. We then filter the subclass of 4500 forward navigational XPATH queries of Lick and Schmitz’s corpus. The other queries contain comparisons of data values, arithmetics, and functions, including higher-order functions to iterate over sequences, which may be nonregular. We also removed boolean queries and kept only node selection ones. We then selected the 180 largest queries of this subcorpus, as for instance:

03257	<code>./equation[title or info/title]</code>
07113	<code>following-sibling::*[self::dbk:appendix   self::dbk:article   self::dbk:book   self::dbk:chapter   self::dbk:part   self::dbk:preface   self::dbk:section   self::dbk:sect1   self::dbk:sect2   self::dbk:sect3   self::dbk:sect4   self::dbk:sect5]   following-sibling::dbk:para[@rnd:style = 'bibliography' or @rnd:style = 'bibliography-title' or @rnd:style = 'glossary' or @rnd:style = 'glossary-title' or @rnd:style = 'qandaset' or @rnd:style = 'qandaset-title']</code>

The query 03257, namely  $\mathbf{Q}_{03257}$ , is considered large due to the descendant axis.  $\mathbf{Q}_{07113}$  is large since it contains the following-sibling axis, and also since its parse tree has more than 15 nodes. We then removed duplicates of queries up to renaming of *XML* names and namespaces and syntactical details, such as `./author` or `descendant::author` or `descendant::corpauthor`. This lead us to the collection of 79 queries in the appendix.

We then improved the compilation chain to symbolic *SHAs* (*SSHAs*) from [16]. These are *SHAs* extended with two kinds of else rules and typed signatures. First, we improved our XPATH parser that maps forward navigational XPATH queries to FXP formulas. Second, we improved our compiler from FXP formulas to nested regular expressions. These are compiled to *SSHAs* in a third step, that could be preserved without change.

The remaining algorithmic question is how to lift schema-based determinization from *SHAs* to *SSHAs*. We contribute the needed extensions for the symbolic aspects. We elaborate only the treatment of else rules in the core of the paper. The extensions needed for apply-else rules and typed signatures had to be moved to the appendix even though nontrivial.

The application of our algorithm for schema-based determinization of *SSHAs* leads us to a corpus of deterministic *SSHAs* for regular XPATH queries that we compiled further to deterministic symbolic *NWAs*. All our automata are published at <https://gitlab.inria.fr/aalserha/xpath-benchmark>.

**Outline.** In Section 2, we start with preliminaries on nested words and *XML* documents. In Section 3, we recall schema-based determinization for NFAs that we extend to else rules in Section 4 and to *SSHAs* in Section 5. In Section 6, we present an *SSHA* schema for the *XML* data model, and discuss show how to defined regular queries by *SSHAs*. The benchmark of deterministic query automata derived from Lick’s and Schmitz’s collection is described in Section 7. Complete statistics can be found in the appendix.

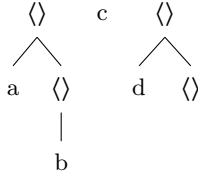
## 2 Words, Nested Words, and XML Documents

An alphabet  $\Sigma$  is a set. A word over with alphabet  $\Sigma$  is a sequence of letters in  $\Sigma^* = \cup_{n=0}^{\infty} \Sigma^n$ . The empty word is the unique element  $\varepsilon \in \Sigma^0$ . The concatenation of two words  $w, w' \in \Sigma^*$  by  $w \cdot w' \in \Sigma^*$ .

Nested words generalize on words by adding parenthesis that must be well-nested. Nested words also generalize on unranked trees and over sequences thereof that are often called hedges. We restrict ourselves to nested words with a single pair of opening and closing parenthesis  $\langle$  and  $\rangle$ , since named parenthesis can be encoded easily. Nested words in  $\mathcal{N}_{\Sigma}$  have the following abstract syntax.

$$w, w' \in \mathcal{N}_{\Sigma} ::= \varepsilon \mid a \mid \langle w \rangle \mid w \cdot w' \quad \text{where } a \in \Sigma.$$

We assume that concatenation  $\cdot$  is associative, and that the empty word  $\varepsilon$  is a neutral element, that is  $w \cdot (w' \cdot w'') = (w \cdot w') \cdot w''$  and  $\varepsilon \cdot w = w = w \cdot \varepsilon$ . Nested words can be identified with hedges, i.e., words of unranked trees and letters from  $\Sigma$ . Seen as a graph, the inner nodes are labeled by the tree constructor  $\langle \rangle$  and the leafs by symbols in  $\Sigma$  or the tree constructor. For instance  $\langle a \cdot \langle b \rangle \cdot \varepsilon \rangle \cdot c \cdot \langle d \cdot \langle \varepsilon \rangle \rangle$



corresponds to the hedge on the right.

*XML* documents are labeled unranked trees that can be serialized into a text, such as for instance:  $\langle s:a \text{ name}=\text{"uff"} \rangle \langle s:b \text{ gaga} \langle s:d \rangle \langle /s:b \rangle \langle s:c \rangle \langle /s:a \rangle$ . Labeled unranked trees satisfying the *XML* data model can be represented as nested words over a signature that contains the *XML* node-types (*elem*, *attr*, *text*, ...), the *XML* namespaces of the document(s), the *XML* names of the document ( $a, \dots, d, \text{nam}$ ), and the characters of the data values, say UTF8. For the above example, we get the nested word:

$$\langle \text{elem} \cdot s \cdot a \cdot \langle \text{attr} \cdot \text{nam} \cdot u \cdot f \cdot s \cdot f \rangle \langle \text{elem} \cdot s \cdot b \cdot \langle \text{text} \cdot g \cdot a \cdot g \cdot a \rangle \langle \text{elem} \cdot s \cdot d \rangle \rangle \langle \text{elem} \cdot s \cdot c \rangle \rangle$$

We also notice that the alphabet for *XML* document has 4 different types of letters (node-types, name-spaces, names, characters).

We are interested in finite state automata that are able to read any nested word encoding some *XML* document. Given that the sets of *XML* names and *XML* namespaces are infinite, such automata must admit infinite alphabets. The infiniteness calls for finite representation by symbolic automata. We will start with finite alphabets  $\Sigma$  and discuss infinite and typed alphabets later on.

## 3 Finite State Automata

We recall algorithms for schema-based cleaning [16] and schema-based determinization [17] in the case of finite state automata on words (NFAs).

**Definition 1.** An NFA is a tuple  $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$  such that the alphabet  $\Sigma$  is finite set of letters, the state set  $\mathcal{Q}$  is a finite set,  $I, F \subseteq \mathcal{Q}$  are subsets of initial and final states, and the set of transition rules is a subset  $\Delta \subseteq (\mathcal{Q} \times \Sigma) \times \mathcal{Q}$ . An NFA is called deterministic or equivalently a DFA if  $\Delta$  is a partial function and  $I$  contains at most one initial state.

$$\begin{array}{c}
\frac{I^A \neq \emptyset}{I^A \in I^{\det(A)} \quad I^A \in \mathcal{Q}^{\det(A)}} \quad \frac{Q \in \mathcal{Q}^{\det(A)} \quad Q \cap F^A \neq \emptyset}{Q \in F^{\det(A)}} \\
\frac{Q \in \mathcal{Q}^{\det(A)} \quad Q' = \{q' \in \mathcal{Q}^A \mid q \xrightarrow{a} q' \in \Delta^A, q \in Q\} \neq \emptyset}{Q \xrightarrow{a} Q' \in \Delta^{\det(A)} \quad Q' \in \mathcal{Q}^{\det(A)}}
\end{array}$$

Fig. 1: Accessible determinization  $\det(A)$  of an NFA  $A$ .

As usual we draw NFAs as graphs whose nodes are the states in  $\mathcal{Q}$  (see e.g. Fig. 6). A state  $q \in \mathcal{Q}$  is drawn with a circle  $\textcircled{q}$ , an initial state  $q \in I$  with an incoming arrow  $\rightarrow\textcircled{q}$  and a final states with a double circle  $\textcircled{\textcircled{q}}$ . A transition rule  $(q_1, a, q_2) \in \Delta$  is drawn as a black edge  $\textcircled{q_1} \xrightarrow{a} \textcircled{q_2}$  that is labeled by a letter.

Transitions for NFAs on words have the form  $q \xrightarrow{w} q'$  wrt  $\Delta$  where  $w \in \mathcal{N}_\Sigma$  and  $q, q' \in \mathcal{Q}$ . They are defined by the following inference rules:

$$\frac{q \in \mathcal{Q}}{q \xrightarrow{\varepsilon} q \text{ wrt } \Delta} \quad \frac{q \xrightarrow{a} q' \in \Delta}{q \xrightarrow{a} q' \text{ wrt } \Delta} \quad \frac{q_0 \xrightarrow{w_1} q_1 \text{ wrt } \Delta \quad q_1 \xrightarrow{w_2} q_2 \text{ wrt } \Delta}{q_0 \xrightarrow{w_1 \cdot w_2} q_2 \text{ wrt } \Delta}$$

The language  $\mathcal{L}(A)$  accepted by an NFA  $A$  is  $\mathcal{L}(A) = \{w \in \Sigma^* \mid q \xrightarrow{w} q' \text{ wrt } \Delta, q \in I, q' \in F\}$ . A sink state of an NFA is a state from which no final state can be accessed. We denote the set of all sink states of  $A$  by  $\text{sink}(A)$ . The accessible determinization  $\det(A)$  of an NFA can be computed by the rule system in Fig. 1, that describes the usual subset construction, but adapted in such a way that only accessible states are added.

The schema-based cleaning  $\text{scl}_S(A)$  of an NFA  $A$  with respect to a DFA  $S$  keeps only those transition of  $A$  that are needed to recognize some word in the language of the schema  $\mathcal{L}(S)$ . The NFA  $\text{scl}_S(A)$  can be obtained by projecting the accessible product  $A \times S$  to  $A$ . A one-shot rule system doing so is presented in Fig. 2. An alignment of  $q \in \mathcal{Q}^A$  with a schema state  $s \in \mathcal{Q}^S$  is denoted as  $q \sim s$ , meaning that  $(Q, s)$  is a state of the accessible product  $A \times S$ . Furthermore,  $q \in \mathcal{Q}^{\det_S(A)}$  iff  $q \in \mathcal{Q}^{\det(A)}$  and  $q \sim s$  for some  $s \notin \text{sink}(S)$ . Sinks of  $S$  are treated as if they were removed.

The schema-based determinization  $\det_S(A)$  runs the accessible determinization  $\det(A)$  in parallel with its schema-based cleaning for schema  $S$ . So only those states  $Q \in \mathcal{Q}^{\det(A)}$  are kept that can be aligned to some non-sink state  $s \in \mathcal{Q}^S \setminus \text{sink}(S)$  in the accessible product  $\det(A) \times S$ . The rules for schema-based determinization in Fig. 3 are analogous to those of schema-based cleaning in Fig. 2. An alignment of a subset  $Q \subseteq \mathcal{Q}^A$  with a schema state  $s$ , that is denoted as  $Q \sim s$ , means that  $(Q, s)$  is a state in the accessible product  $\det(A) \times S$ . Furthermore,  $Q \in \mathcal{Q}^{\det_S(A)}$  iff  $Q \in \mathcal{Q}^{\det(A)}$  and  $Q \sim s$  for some  $s \notin \text{sink}(S)$ . The correctness statement proven in [17] shows that  $\det_S(A) = \text{scl}_S(\det(A))$ . Moreover, the worst-case complexity of schema-based determinization is lower than of the accessible determinization followed by schema-based cleaning.

$$\begin{array}{c}
 \frac{q \in I^A \quad s \in I^S}{q \in I^{scl_S(A)} \quad q \sim s} \quad \frac{q \in F^A \quad s \in F^S \quad q \sim s}{q \in F^{scl_S(A)}} \\
 \\
 \frac{q_1 \xrightarrow{a} q \in \Delta^A \quad s_1 \xrightarrow{a} s \in \Delta^{exp(S)} \quad s \notin sink(S) \quad q_1 \sim s_1}{q_1 \xrightarrow{a} q \in \Delta^{scl_S(A)} \quad q \sim s} \quad \frac{q \sim s}{q \in Q^{scl_S(A)}}
 \end{array}$$

 Fig. 2: Schema-based cleaning  $scl_S(A)$  of a NFA  $A$  with respect to a DFA  $S$ .

$$\begin{array}{c}
 \frac{Q \in I^{det(A)} \quad I^S = \{s\}}{Q \in I^{det_S(A)} \quad Q \sim s} \quad \frac{Q \in F^{det(A)} \quad s \in F^S \quad Q \sim s}{Q \in F^{det_S(A)}} \\
 \\
 \frac{Q \xrightarrow{a} Q' \in \Delta^{det(A)} \quad s \xrightarrow{a} s' \in \Delta^S \quad s' \notin sink(S) \quad Q \sim s}{Q \xrightarrow{a} Q' \in \Delta^{det_S(A)} \quad Q' \sim s'} \quad \frac{Q \sim s}{Q \in Q^{det_S(A)}}
 \end{array}$$

 Fig. 3: The schema-based determinization  $det_S(A)$  for an NFA  $A$  and a DFA  $S$ .

## 4 Symbolic Finite State Automata

We next show how to lift schema-based cleaning and determinization to symbolic NFAs. These extend on NFAs by adding else rules.

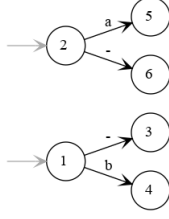
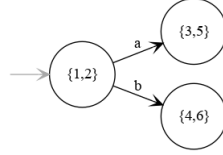
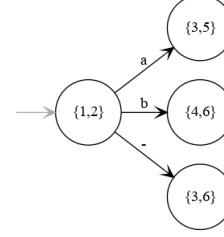
**Definition 2.** A symbolic NFA or equivalently an SNFA is a tuple  $A = (\Sigma, Q, \Delta, I, F)$  such that  $\Sigma$  is a finite alphabet,  $\Delta = (\Delta', \_{}^\Delta)$  and  $A' = (\Sigma, Q, \mathcal{P}, \Delta', I, F)$  is a NFA and  $\_{}^\Delta \subseteq Q^2$  a set of else transition rules. We call an SNFA deterministic or equivalently a SDFFA if  $A'$  is a DFA and  $\_{}^\Delta$  a partial function.

An else rule  $(q_1, q_2) \in \_{}^\Delta$  is denoted by  $q_1 \rightarrow q_2$  meaning that the automaton in state  $q_1$  can go to state  $q_2$  when reading any letter  $a \in \Sigma$  such that there exists no  $q_3$  with  $q_1 \xrightarrow{a} q_3 \in \Delta$ . Since  $\Sigma$  is finite, any else rule can be expanded to a set of internal rules, so any SNFA  $A$  can be expanded to an NFA  $exp(A)$ :

$$\frac{q \rightarrow q' \in \Delta^A \quad a \in \Sigma \quad \neg \exists q'' \in Q^A. q \xrightarrow{a} q'' \in \Delta^A}{q \xrightarrow{a} q' \in \Delta^{exp(A)}} \quad \frac{q \xrightarrow{a} q' \in \Delta^A}{q \xrightarrow{a} q' \in \Delta^{exp(A)}}$$

In Fig. 4, we lift the accessible determinization from NFAs to *SNFAs*. For this, some rule expansion is needed, but we limit it as much as possible. This is illustrated at the example of the SNFA  $E$  in Fig. 5. The else rule  $2 \rightarrow 6$  means that  $E$  go from state 2 to state 6 when reading any letter from  $\Sigma \setminus \{a\}$ . In a similar manner, the rule  $1 \rightarrow 3$  permits to go from state 1 to state 3 when reading any letter from  $\Sigma \setminus \{b\}$ . Note that the interpretation of the else rules depends on the alphabet  $\Sigma$ . The same holds for the result of accessible determinization. For  $\Sigma = \{a, b\}$ , the else rules has two instances for the letters  $a$  and  $b$  respectively. Since  $det(E)$  in Fig. 6 already has an internal transition for both letters, no else rule is produced any more. Whereas for  $det(E)$  in Fig. 7 with alphabet  $\Sigma = \{a, b, c\}$  an else rule is needed to capture letter  $c$  of the alphabet, which does not appear in the graph of  $E$ .

$$\begin{array}{c}
\frac{Q_1 \xrightarrow{a} Q_2 \in \Delta^{\det(\text{exp}(A))} \quad \exists q_1 \in Q_1 \exists q \in \mathcal{Q}^A. q_1 \xrightarrow{a} q \in \Delta^A}{Q_1 \xrightarrow{a} Q_2 \in \Delta^{\det(A)}} \\
\frac{Q_1 \in \mathcal{Q}^{\det(\text{exp}(A))} \quad Q_2 = \{q \in \mathcal{Q}^A \mid q_1 \xrightarrow{a} q \in \Delta^A, q_1 \in Q_1\} \neq \emptyset}{\exists a \in \Sigma \forall q_1 \in Q_1 \neg \exists q' \in \mathcal{Q}^A. q_1 \xrightarrow{a} q' \in \Delta^A} \\
\hline
Q_1 \xrightarrow{a} Q_2 \in \Delta^{\det(A)}
\end{array}$$

Fig. 4: Accessible determinization for *SNFAs* lifted from *NFAs*.Fig. 5: The *SNFA*  $E$ .Fig. 6:  $\det(E)$  for  $\Sigma = \{a, b\}$ .Fig. 7:  $\det(E)$  for  $\Sigma = \{a, b, c\}$ .

For *SNFA*, schema-based cleaning is shown in Fig. 8 and schema-based determinization in Fig. 9. As in the non-symbolic case, the two rule systems are analogous. The first rule on the left says that an  $a$ -transition of  $A$  (resp.  $\det(A)$ ) is preserved in  $scl_A$  (resp.  $\det_S(A)$ ), if it is aligned to an  $a$ -transition of the expansion of  $S$  into a non-sink. The second rule on the left says that an else transition of  $A$  (resp.  $\det(A)$ ) is preserved in  $scl_A$  (resp.  $\det_S(A)$ ), if it can be expanded to an  $a$ -transition that is aligned to an  $a$ -transition of the expansion of  $S$  into a non-sink. The rule on the right states that  $a$ -transitions of  $A$  (resp.  $\det(A)$ ) must be preserved in  $scl_A$  (resp.  $\det_S(A)$ ), if the meaning of some else rule of  $scl_A$  (resp.  $\det_S(A)$ ) would be changed otherwise.

Suppose that we are given a finite set of types  $T$  and that any letter of alphabet  $\Sigma$  can be given some types in  $T$ . It is then useful to extend *SNFAs*, with typed alphabets, with typed else rules of the form  $q \xrightarrow{a:\tau} q'$  where  $\tau \in T$ . Semantically, a typed else rule for type  $\tau$  can be expanded with all letters  $a \in \Sigma$  that can be given the type  $\tau$ . Untyped else rules as before can be covered by allowing for a universal type  $all \in T$ .

## 5 Symbolic Stepwise Hedge Automata

We next lift schema-based cleaning and determinization from *SNFAs* to symbolic stepwise hedge automata (*SSHAs*) on nested words. The case of *SHAs* was presented earlier [16]. The symbolic aspects can be taken over from *SNFAs*.

**Definition 3.** An *SSHA* is a tuple  $A = (\Sigma, \mathcal{Q}, \mathcal{P}, \Delta, I, F)$  where  $\Delta = (\Delta', \Delta'')$  so that  $A' = (\Sigma, \mathcal{Q}, \Delta', I, F)$  is an *SNFA*. Furthermore,  $\mathcal{P}$  is a finite set of tree states and  $\Delta'' = (\diamond^\Delta, @^\Delta, \dashrightarrow^\Delta)$ , such that  $\diamond^\Delta \subseteq \mathcal{Q}$  is a subset of tree initial states,  $@^\Delta \subseteq \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$  a set of apply rules, and  $\dashrightarrow^\Delta \subseteq \mathcal{Q} \times \mathcal{P}$  a set of tree

$$\begin{array}{c}
 \frac{q_1 \xrightarrow{a} q \in \Delta^A \quad s_1 \xrightarrow{a} s \in \Delta^{exp(S)} \quad s \notin sink(S) \quad q_1 \sim s_1}{q_1 \xrightarrow{a} q \in \Delta^{scl_S(A)} \quad q \sim s} \quad \frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad q_1 \rightarrow q \in \Delta^{scl_S(A)}}{q_1 \xrightarrow{a} q_2 \in \Delta^{scl_S(A)}} \\
 \\
 \frac{q_1 \rightarrow q \in \Delta^A \quad s_1 \xrightarrow{a} s \in \Delta^{exp(S)} \quad s \notin sink(S) \quad q_1 \sim s_1 \quad \neg \exists q_2. q_1 \xrightarrow{a} q_2 \in \Delta^A}{q_1 \rightarrow q \in \Delta^{scl_S(A)} \quad q \sim s}
 \end{array}$$

 Fig. 8: Schema-based cleaning to *SNFAs* with respect to *SDFAs*.

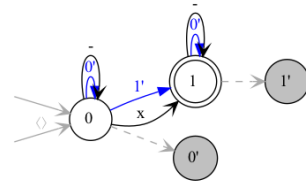
$$\begin{array}{c}
 \frac{Q_1 \xrightarrow{a} Q \in \Delta^{det(A)} \quad s_1 \xrightarrow{a} s \in \Delta^{exp(S)} \quad s \notin sink(S) \quad Q_1 \sim s_1}{Q_1 \xrightarrow{a} Q \in \Delta^{det_S(A)} \quad Q \sim s} \quad \frac{Q_1 \xrightarrow{a} Q \in \Delta^{det(A)} \quad \exists Q_2. Q_1 \rightarrow Q_2 \in \Delta^{det_S(A)}}{Q_1 \xrightarrow{a} Q \in \Delta^{det_S(A)}} \\
 \\
 \frac{Q_1 \rightarrow Q \in \Delta^{det(A)} \quad s_1 \xrightarrow{a} s \in \Delta^{exp(S)} \quad s \notin sink(S) \quad Q_1 \sim s_1 \quad \neg \exists Q_2. Q_1 \xrightarrow{a} Q_2 \in \Delta^{det(A)}}{Q_1 \rightarrow Q \in \Delta^{det_S(A)} \quad Q \sim s}
 \end{array}$$

 Fig. 9: Schema-based determinization for *SNFAs* wrt. *SDFAs*.

*final rules.* We call  $A$  deterministic or equivalently a *dSSHA*, if  $A'$  is a *SDFA*, there is at most one tree initial state in  $\langle \rangle^\Delta$ , at most one initial hedge state in  $I$ , and  $@^\Delta$  and  $\rightarrow^\Delta$  are partial functions.

The notion of determinism for *SSHAs* extends on the notion of left-to-right determinism of *SNFAs* and on the notion of bottom-up determinism of tree automata. The states in  $\mathcal{Q}$  are also called hedge states. The graphs of *SSHAs* extend on those for *SNFAs*. A tree state  $p \in \mathcal{P}$  is drawn in gray  $\textcircled{p}$ . A tree initial state  $q \in \langle \rangle^\Delta$  is drawn as a node  $\xrightarrow{\langle \rangle} \textcircled{q}$  with an incoming tree arrow. An apply rule  $(q_1, p, q_2) \in @^\Delta$  is drawn as a blue edge  $\textcircled{q_1} \xrightarrow{p} \textcircled{q_2}$  that is labeled by a state  $p \in \mathcal{P}$  rather than a letter  $a \in \Sigma$ . It states that a nested word in state  $q_1 \in \mathcal{Q}$  can be extended by a tree in state  $p \in \mathcal{P}$  and become a nested word in state  $q_2 \in \mathcal{Q}$ . A tree final rule  $(q, p) \in \rightarrow^\Delta$  by  $q \rightarrow p$ .

On the right we show the graph of the *SSHA*  $one^x$  that accepts all nested words over  $\Sigma \uplus \{x, \neg x\}$  with exactly one occurrence of letter  $x$ . Beside of the DFA recognizing all words over  $\Sigma \cup \{x, \neg x\}$  with a single occurrence of  $x$ , it has three additional apply rules  $(0, 0', 0)$ ,  $(0, 1', 1)$ ,



$(1, 0', 1) \in @^{\Delta^{one^x}}$  for reading the tree states assigned to subtrees. The hedge state 0 is the single tree initial state.



$$\begin{array}{c}
\frac{\diamond^{\Delta^A} \neq \emptyset}{\diamond^{\Delta^A} \in \mathcal{Q}^{det(A)}} \quad \frac{Q \in \mathcal{Q}^{det(A)} \quad P = \{p \in \mathcal{P}^A \mid \exists q \in Q, q \dashrightarrow p \in \Delta^A\} \neq \emptyset}{Q \dashrightarrow P \in \Delta^{det(A)} \quad P \in \mathcal{P}^{det(A)}} \\
\frac{Q \in \mathcal{Q}^{det(A)} \quad P \in \mathcal{P}^{det(A)}}{Q' = \{q' \in \mathcal{Q}^A \mid q @ p \rightarrow q' \in \Delta^A, q \in Q, p \in P\} \neq \emptyset} \\
\frac{Q @ P \rightarrow Q' \in \Delta^{det(A)} \quad Q' \in \mathcal{Q}^{det(A)}}{}
\end{array}$$

Fig. 10: Accessible determinization for *SSHAs* extending on *SNFAs* in Fig. 4.

$$\begin{array}{c}
\frac{\diamond^{\Delta^S} = \{s\}}{\diamond^{\Delta^A} \in \diamond^{\Delta^{det_S(A)}} \quad \diamond^{\Delta^A} \sim s} \quad \frac{Q \dashrightarrow P \in \Delta^{det(A)} \quad s \dashrightarrow r \in \Delta^S \quad r \notin \text{sink}(S) \quad Q \sim s}{Q \dashrightarrow P \in \Delta^{det_S(A)} \quad P \sim r} \\
\frac{Q_1 @ Q_2 \rightarrow Q' \in \Delta^{det(A)} \quad s_1 @ s_2 \rightarrow s' \in \Delta^S \quad s_1 \notin \text{sink}(S) \quad s_2 \notin \text{sink}(S) \quad Q_1 \sim s_1 \quad Q_2 \sim s_2}{Q_1 @ Q_2 \rightarrow Q' \in \Delta^{det_S(A)} \quad Q' \sim s'}
\end{array}$$

Fig. 11: Schema-based determinization for *SSHAs* on top of Fig. 9 for *SNFAs*.

Transitions of *SSHAs* have the form  $q \xrightarrow{w} q'$  wrt  $\Delta$  where  $w \in \mathcal{N}_\Sigma$  and  $q, q' \in \mathcal{Q}$ . They are defined by the inference rule for NFAs and the following rule:

$$\frac{q' \in \diamond^{\Delta} \quad q' \xrightarrow{w} q \text{ wrt } \Delta \quad q \dashrightarrow p \in \Delta \quad (q_1, p, q_2) \in @^{\Delta}}{q_1 \xrightarrow{\langle w \rangle} q_2 \text{ wrt } \Delta}$$

It says that a tree  $\langle w \rangle$  can transit from a state  $q_1$  to a state  $q_2$  if  $w$  can transit from some tree initial state  $q'$  to  $q$ , so that there is some tree final rule  $(q, p) \in \dashrightarrow^{\Delta}$  and some apply rule  $(q_1, p, q_2) \in @^{\Delta}$ . The language  $\mathcal{L}(A)$  of a *SSHA* is defined literally as for *SNFAs* except that now nested words may be recognized too.

The accessible determinization can be lifted from *SNFAs* to *SSHAs* by adding the rules in Fig. 10. It should be noticed that the three additional rules for *SSHAs* are fully analogous to the three rules for NFAs. The schema based cleaning can be lifted from *SNFAs* to *SSHAs* too. The analogous schema-based determinization  $det_S(A)$  of a *SSHA* with respect to a *dSSHA* is given in Fig. 11.

In our application, we will permit apply else rules for *SSHAs* in addition. These have the form  $(q, q') \in @_{-}^{\Delta}$  and are drawn as  $\textcircled{q} \dashrightarrow \textcircled{q'}$  in the graphs. Apply else rules work similarly to else rules: the underscore can be expanded to an else tree states  $p \in \mathcal{P}_{else}$  such that there does not exist  $q''$  with  $\textcircled{q} \xrightarrow{p} \textcircled{q''}$ . The set of else tree states  $\mathcal{P}_{else} \subseteq \mathcal{P}$  is distinguished by the automaton. Its elements  $p \in \mathcal{P}_{else}$  are marked as  $\textcircled{p}$  in the pictures. See Fig. 13 for an example of an *SSHA* with apply else rules and else tree states.

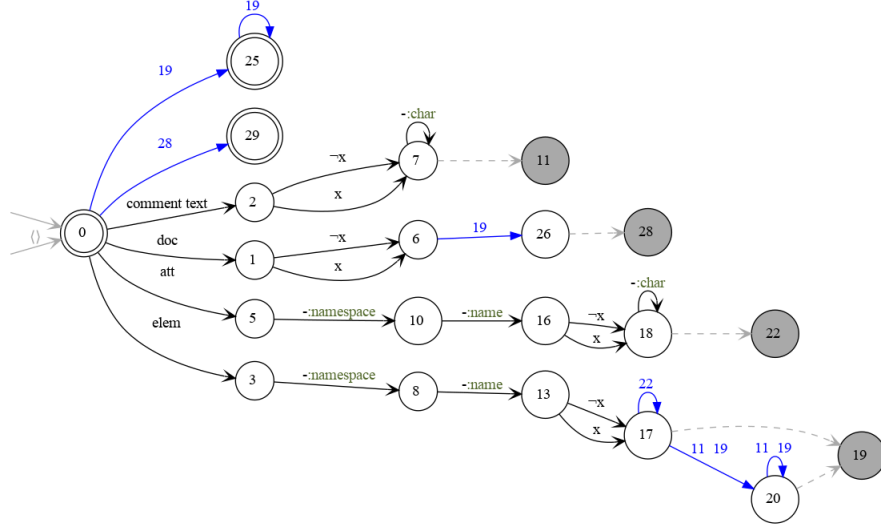


Fig. 12: The  $dSSHA$  schema  $x\text{-doc}$  for  $x$ -annotated  $XML$  documents.

## 6 Regular Schemas and Queries for XML Documents

We now discuss how to define regular schemas and queries on nested words by  $SSHAs$ . Since interested in monadic node selection queries, we fix a single variable  $x$  that is to be annotated to the selected node.

In Fig. 12, we present the schema  $x\text{-doc}$  as a  $dSSHA$  with typed else rules, that captures the  $XML$  data model. It accepts all valid  $XML$  documents, whose nodes are annotated by either  $x$  or  $\neg x$ . The alphabet  $\Sigma_{XML} \uplus \{x, \neg x\}$  has 5 types where  $\Sigma_{XML} = \Sigma_{types} \uplus \Sigma_{namespace} \uplus \Sigma_{name} \uplus \Sigma_{char}$ . Note that  $x\text{-doc}$  also accepts in state 25 any subdocument of any  $XML$  document, i.e., sequence of  $XML$  elements, texts, and comments.

Monadic queries on nested words select a subset of the nodes of its graph. For this we assume that the nodes of such graphs named by natural numbers in an arbitrary but unique manner. We denote the set of nodes of the graph of a nested word  $w$  by  $nod(w) \subseteq \mathbb{N}$ . A (total) monadic query  $\mathbf{Q}$  is a function that maps any nested word  $w \in \mathcal{N}_\Sigma$  to a subset of its nodes  $\mathbf{Q}(w) \subseteq nod(w)$ .

Monadic queries on nested words can be defined by query automata. We will use  $SSHAs$  that use a single variable  $x$ , relying on the notion of  $V$ -structures [21] where  $V = \{x\}$ . For any node  $\pi$  of a nested word  $w \in \mathcal{N}_\Sigma$ , let  $w * [\pi/x]$  be the nested word obtained by inserting  $x$  after the position of node  $\pi$  in  $w$ . The word  $w * [\pi/x]$  is a  $V$ -structure since  $x$  occurs exactly once in it. The annotation by  $x$  marks a node that is to be selected. Formally, we identify any total monadic query  $\mathbf{Q}$  with the language of  $V$ -structures such that  $L(\mathbf{Q}) = \{w * [\pi/x] \mid w \in \mathcal{N}_\Sigma, \pi \in \mathbf{Q}(w)\}$ . Such queries for nested words can be defined by  $SSHAs$   $A$  with alphabet  $\Sigma \uplus \{x, \neg x\}$ , such that  $L_{\mathbf{Q}} = \mathcal{L}(A) \cap \mathcal{L}(one^x)$ . So a node  $\pi$  of a nested word  $w \in \mathcal{N}_\Sigma$  is selected by  $\mathbf{Q}$  on  $w$  iff  $\pi \in \mathbf{Q}(w) \Leftrightarrow w * [\pi/x] \in \mathcal{L}(A)$ .

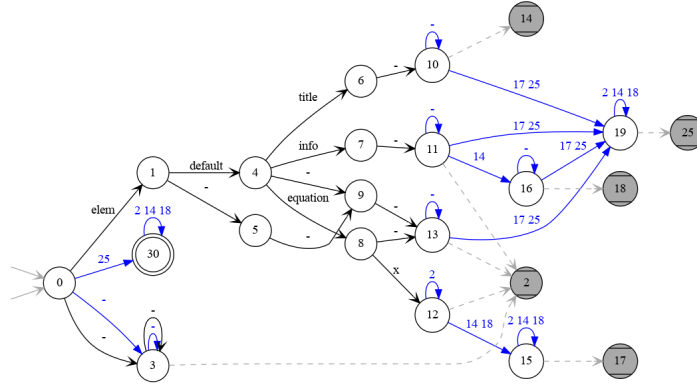


Fig. 13: The minimal  $dSSHA$  for  $\mathbf{Q}_{03257}$ : `//equation[title or info/title]`.

In Fig. 13, the  $dSSHA$  for the query  $\mathbf{Q}_{03257}$  of our XPATH benchmark is given. Here, the symbol `default` stands for the default namespace of XML elements and attributes.

## 7 Benchmark of Query Automata

We compile each of the forward navigational XPATH queries  $Q$  in the collection to a symbolic  $SHA$   $A = sha(Q)$  by applying the above compiler followed by accessibility cleaning and schema-based cleaning. The sizes of  $sha(A)$  for our subcorpus of 79 XPATH queries are given in the first column of Fig. 14. 40% of the  $SHAs$  have more than 100 states. The biggest is  $sha(\mathbf{Q}_{05684})$  with size 1276 and 580 states. This query is selecting an union of 37 attributes (36 distinct with one duplicate attribute) of the current node. For each of these attributes a symbolic  $SHA$  with 16 states is constructed, and  $16 * 36 = 576$  in addition to two states for the duplicate attribute and two others for reading any other tree.

The second column contains the sizes and number of states of the accessible determinization  $det(A)$  where no schema is used. We use a timeout of 5 minutes. Whenever this is not enough, the cell in the table is left blank. Indeed, the accessible determinization fails with this timeout for nearly 40% of the queries of our corpus. Roughly, the determinization fails for all  $SHAs$  with more than 100 states and succeeds otherwise. For instance, for query  $Q = \mathbf{Q}_{05460}$ ,  $A = sha(Q)$  the size 214 and 88 states, while  $det(A)$  has size 38361 and 610 states.

The third column reports on  $B = det_{one^x}(A)$  obtained by schema-based determinization with schema  $one^x$ . Indeed, the computation succeeds in all cases in less than 5 minutes, which is a tremendous improvement compared to schemaless determinization. Furthermore, only 88% of the  $dSHAs$  obtained in this way have less than 100 states. In other words, schema-based determinization decreased the automata size in at least 28% of the cases.

The fourth column applies schema-based cleaning schema  $x-doc$  to  $B$ . It turns out that  $scl_{x-doc}(B)$  often has considerably lower size than  $B$ , but always the same number of states. Furthermore, the computation with timeout of 5 minutes fails precisely for the 12% of the automata that have more than 100 states.

$Q$	A = <i>SSHA</i> (Q)	$det(A)$	B = $det_{one^x}(A)$	$scl_{x-doc}(B)$	C = $det_{one^x \times x-doc}(A)$	mini(C)
18330	89 (37)	400 (38)	127 (22)	73 (22)	73 (22)	60 (18)
09138	349 (152)		410 (59)	198 (59)	198 (59)	62 (15)
05460	214 (88)	38163 (610)	1039 (173)	628 (173)	628 (173)	69 (18)
06512	120 (49)	407 (42)	202 (34)	114 (34)	114 (34)	74 (22)
06176	1191 (550)		5271 (386)		1202 (386)	
12539	159 (68)	1965 (112)	311 (40)	137 (40)	137 (40)	100 (28)
11780	166 (70)	2286 (128)	317 (41)	142 (41)	142 (41)	98 (27)
05684	1276 (580)		4044 (226)		718 (226)	123 (16)
06947	704 (322)		1575 (129)	443 (129)	443 (129)	70 (14)
06794	256 (114)		399 (51)	177 (51)	177 (51)	63 (15)
04358	966 (424)		3354 (433)		2020 (433)	

Fig. 14: Few results on the automata for the queries in size(number-of-states).

In the fifth column, we apply schema-based determinization with both schemas  $one^x$  and  $x-doc$ , so we compute  $C = det_{one^x \times x-doc}(A)$ . The computation succeeds in all cases in less than 5 minutes. The correctness statement  $scl_{x-doc}(B) = det_{x-doc}(B) = C$  suggested by Theorem 2 in [17] is confirmed in all cases. Note that the theorem cannot be applied directly, since we deal with symbolic *SHAs* rather than *SHAs* in our experiments.

In the last column, we applied minimization to compute  $mini(C)$ . This worked out in 97% of the queries within a timeout of 5 minutes, but failed for the 2 biggest automata. One of them is for the query  $Q_{04358}$ .

All minimal *dSHAs*  $mini(C)$  that we could compute in less than 5 minutes have at most 50 states. The biggest minimal automata for  $Q_{07113}$  below has 49 states and size 365. Our experiments show that we can obtain deterministic automata of decent size based on schema-based determinization in practice. Furthermore, based on minimization we can even obtain quite small deterministic automata in 97% of the cases. The remaining 3% could possibly be improved by not using the naïve minimization algorithm.

## 8 Conclusion

We provide a benchmark of deterministic automata for regular XPath queries obtained with an algorithm for schema-based determinization of symbolic stepwise hedge automata (*SSHAs*) that we presented. Our benchmark is compiled from forward navigational XPath (FXP) queries : the 79 largest queries modulo renaming of the 4500 FXP queries of the corpus of Lick and Schmitz, and 10 FXP queries from the XPathMark benchmark. While 40% of *SSHAs* obtained from these FXP queries cannot be determinized in less than 5 minutes by standard determinization; schema-based determinization succeeds for 100% of them. Furthermore, all but 2 of the *dSSHAs* obtained are sufficiently small so that

they can be minimized with the naïve quadratic algorithm, leading us to small *dSSHAs* with at most 50 states in 97% of the cases.

## References

1. Alur, R.: Marrying words and trees. In: 26th ACM Symposium on Principles of Database Systems. pp. 233–242. (2007).
2. Bagan, G.: MSO queries on tree decomposable structures are computable with linear delay. In: Comput. Sci. Logic. LNCS, vol. 4646, pp. 208–222. (2006).
3. von Braunmühl, B., Verbeek, R.: Input driven languages are recognized in log n space. In: Theory of Computation, North-Holland Mathematics Studies, vol. 102, pp. 1 – 19.(1985).
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. <http://tata.gforge.inria.fr> (2007)
5. Courcelle, B.: Linear delay enumeration and monadic second-order logic. Discrete Applied Mathematics **157**(12), 2675–2700 (2009).
6. Debarbieux, D., Gauwin, O., Niehren, J., Sebastian, T., Zergaoui, M.: Early nested word automata for xpath query answering on XML streams. Theor. Comput. Sci. **578**, 100–125 (2015).
7. Muñoz, M., Riveros, C.: Streaming query evaluation with constant delay enumeration over nested documents.ICDT 2022.<https://arxiv.org/pdf/2010.06037.pdf>
8. Fagin, R., Kimelfeld, B., Reiss, F., Vansummeren, S.: Document spanners: A formal approach to information extraction. J. ACM **62**(2), 12:1–12:51 (2015).
9. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**(2), 194–211 (1979).
10. Franceschet, M.: Xpathmark performance test. <https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html>, accessed: 2020-10-25
11. Gauwin, O., Niehren, J., Tison, S.: Earliest query answering for deterministic nested word automata. FCT 2009 . LNCS, vol. 5699, pp. 121–132. (2009).
12. Libkin, L., Martens, W., Vrgoč, D.: Querying graph databases with xpath. In: ICDT 2013. p.129–140.
13. Lick, A.: Logique de requêtes à la XPath : systèmes de preuve et pertinence pratique. Theses, Université Paris-Saclay. (2019).
14. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML Schema. ACM TODS **31**(3), 770–813. (2006).
15. Martens, W., Trautner, T.: Evaluation and Enumeration Problems for Regular Path Queries. In: ICDT 2018. LIPIcs, vol. 98, pp. 19:1–19:21.
16. Niehren, J., Sakho, M.: Determinization and Minimization of Automata for Nested Words Revisited. Algorithms. (2021).
17. Niehren, J., Sakho, M., Al Serhali, A.: Schema-Based Automata Determinization. (2022), <https://hal.inria.fr/hal-03536045>, Working Paper.
18. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**(2), 47–67 (2014).
19. Schmid, M.L., Schweikardt, N.: A Purely Regular Approach to Non-Regular Core Spanners. In: ICDT 2021. LIPIcs, vol. 186, pp. 4:1–4:19.
20. Seidl, H.: Deciding equivalence of finite tree automata. STACS. LNCS, vol. 349, pp. 480–492. (1989)
21. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Progress in Computer Science and Applied Series, Birkhäuser (1994).

Table 2: The 79 largest forward navigational queries of the XPATH corpus of Lick’s and Schmitz, without equivalent queries up to renaming.

Id	XPATH Query
18330	/ descendant-or-self::node()/child::parts-of-speech
17914	/ descendant-or-self::node()/child::tei:back/descendant-or-self::node()/child::tei:interpGrp
10745	*//tei:imprint/tei:date[@type='access']
02091	*   ../refentry
00744	../@id   ../@xml:id
12060	../attDef
02762	../authorgroup/author   ../author
06027	../authorinitials   ../author
02909	../bibliomisc[@role='serie']
06415	../email   address/otheraddr/ulink
03257	../equation[title or info/title]
05122	../procedure[title]
09138	../rng:ref   ../tei:elementRef   ../tei:classRef   ../tei:macroRef   ../tei:dataRef
05460	../table//footnote   ../informaltable//footnote
12404	../tei:dataRef[@name]
10337	../tei:note[@place='end']
06639	../tgroup//footnote
14340	//*
13804	//GAP/@DISP
13896	//HEADER//IDNO[@TYPE='evans citation']
02194	//annotation
06726	//doc:table   //doc:informaltable
13640	//equiv[@filter]
05735	//glossary[@role='auto']
15766	//h:body/h:section[@data-type='titlepage']
15524	//h:section[@data-type='titlepage']
06512	//refentry//text()
06176	//set   //book   //part   //reference   //preface   //chapter   //appendix   //article   //colophon   //refentry   //section   //sect1   //sect2   //sect3   //sect4   //sect5   //indexterm   //glossary   //bibliography   //*[@id]
12539	//tei:elementSpec   //tei:classSpec[@type='atts']
11780	//tei:ref[@type='cite']   //tei:ptr[@type='cite']
11478	//xhtml:p[@class]

11227	/tei:TEI/tei:text//tei:note[@type='action']
05684	@abbr   @align   @axis   @bgcolor   @border   @cellpadding   @cellspacing   @char   @charoff   @class   @dir   @frame   @headers   @height   @id   @lang   @nowrap   @onclick   @ondblclick   @onkeydown   @onkeypress   @onkeyup   @onmousedown   @onmousemove   @onmouseout   @onmouseover   @onmouseup   @rules   @scope   @style   @summary   @title   @valign   @width   @xml:id   @xml:lang
06947	anchor   areaset   audiodata   audioobject   beginpage   constraint   indexterm   itemset   keywordset   msg   doc:anchor   doc:areaset   doc:audiodata   doc:audioobject   doc:beginpage   doc:constraint   doc:indexterm   doc:itemset   doc:keywordset   doc:msg
06794	articleinfo   chapterinfo   bookinfo   doc:info   doc:articleinfo   doc:chapterinfo   doc:bookinfo
06169	article   preface   chapter   appendix   refentry   section   sect1   glossary   bibliography
06924	authorblurb   formalpara   legalnotice   note   caution   warning   important   tip   doc:authorblurb   doc:formalpara   doc:legalnotice   doc:note   doc:caution   doc:warning   doc:important   doc:tip
11958	biblStruct//note
01705	book   article   part   reference   preface   chapter   bibliography   appendix   glossary   section   sect1   sect2   sect3   sect4   sect5   refentry   colophon   bibliodiv[title]   setindex   index
02086	book   article   topic   part   reference   preface   chapter   bibliography   appendix   glossary   section   sect1   sect2   sect3   sect4   sect5   refentry   colophon   bibliodiv[title]   setindex   index
02000	chapter   appendix   epigraph   warning   preface   index   colophon   glossary   biblioentry   bibliography   dedication   sidebar   footnote   glossterm   glossdef   bridgehead   part
02697	chapter   appendix   preface   reference   refentry   article   topic   index   glossary   bibliography
14183	content//rng:ref
07106	dbk:appendix   dbk:article   dbk:book   dbk:chapter   dbk:part   dbk:preface   dbk:section   dbk:sect1   dbk:sect2   dbk:sect3   dbk:sect4   dbk:sect5
05824	descendant-or-self::*
11368	descendant-or-self::tei:TEI/tei:text/tei:back
15848	descendant::*[@class='refname']
15462	descendant::h:span[@data-type='footnote']
04267	descendant::label

07113	following-sibling::*[self::dbk:appendix   self::dbk:article   self::dbk:book   self::dbk:chapter   self::dbk:part   self::dbk:preface   self::dbk:section   self::dbk:sect1   self::dbk:sect2   self::dbk:sect3   self::dbk:sect4   self::dbk:sect5]   following-sibling::dbk:para[@rnd:style = 'bibliography' or @rnd:style = 'bibliography-title' or @rnd:style = 'glossary' or @rnd:style = 'glossary-title' or @rnd:style = 'qandaset' or @rnd:style = 'qandaset-title']
03864	guibutton   guiicon   guilabel   guimenu   guimenuitem   guisubmenu   interface
15484	h:pre[@data-type='programlisting']//text()
15461	h:table[descendant::h:span[@data-type='footnote']]
11160	html:table   html:tr   html:thead   html:tbody   html:td   html:th   html:caption   html:li
06856	imageobject   imageobjectco   audioobject   videoobject   doc:imageobject   doc:imageobjectco   doc:audioobject   doc:videoobject
06458	info   refentryinfo   referenceinfo   refsynopsisdivinfo   refsectioninfo   refsect1info   refsect2info   refsect3info   setinfo   bookinfo   articleinfo   chapterinfo   sectioninfo   sect1info   sect2info   sect3info   sect4info   sect5info   partinfo   prefaceinfo   appendixinfo   docinfo
13710	persName   orgName   addName   nameLink   roleName   forename   surname   genName   country   placeName   geogName
06808	personname   surname   firstname   honorific   lineage   othername   contrib   doc:personname   doc:surname   doc:firstname   doc:honorific   doc:lineage   doc:othername   doc:contrib
04338	refsynopsisdiv/title   refsection/title   refsect1/title   refsect2/title   refsect3/title   refsynopsisdiv/info/title   refsection/info/title   refsect1/info/title   refsect2/info/title   refsect3/info/title
04358	section/title   simplesect/title   sect1/title   sect2/title   sect3/title   sect4/title   sect5/title   section/info/title   simplesect/info/title   sect1/info/title   sect2/info/title   sect3/info/title   sect4/info/title   sect5/info/title   section/sectioninfo/title   sect1/sect1info/title   sect2/sect2info/title   sect3/sect3info/title   sect4/sect4info/title   sect5/sect5info/title
13632	self::placeName   self::persName   self::district   self::settlement   self::region   self::country   self::bloc
01847	set   book   part   preface   chapter   appendix   article   reference   refentry   book/glossary   article/glossary   part/glossary   bibliography   colophon
05219	set   book   part   preface   chapter   appendix   article   topic   reference   refentry   book/glossary   article/glossary   part/glossary   book/bibliography   article/bibliography   part/bibliography   colophon



05226	set   book   part   preface   chapter   appendix   article   topic   reference   refentry   sect1   sect2   sect3   sect4   sect5   section   book/glossary   article/glossary   part/glossary   book/bibliography   article/bibliography   part/bibliography   colophon
03325	set   book   part   reference   preface   chapter   appendix   article   topic   glossary   bibliography   index   setindex   refentry   sect1   sect2   sect3   sect4   sect5   section
03410	set   book   part   reference   preface   chapter   appendix   article   topic   glossary   bibliography   index   setindex   refentry   refsynopsisdiv   refsect1   refsect2   refsect3   refsection   sect1   sect2   sect3   sect4   sect5   section
03407	set   book   part   reference   preface   chapter   appendix   article   glossary   bibliography   index   setindex   refentry   sect1   sect2   sect3   sect4   sect5   section
04245	set   book   part   reference   preface   chapter   appendix   article   glossary   bibliography   index   setindex   refentry   refsynopsisdiv   refsect1   refsect2   refsect3   refsection   sect1   sect2   sect3   sect4   sect5   section
04953	set   book   part   reference   preface   chapter   appendix   article   glossary   bibliography   index   setindex   topic   refentry   refsynopsisdiv   refsect1   refsect2   refsect3   refsection   sect1   sect2   sect3   sect4   sect5   section
07095	sf:stylesheet   sf:stylesheet-ref   sf:container-hint   sf:page-start   sf:br   sf:selection-start   sf:selection-end   sf:insertion-point   sf:ghost-text   sf:attachments
05463	table//footnote   informaltable//footnote
12960	tei:classSpec/tei:attList//tei:attDef/tei:datatype/rng:ref
12961	tei:classSpec/tei:attList//tei:attDef/tei:datatype/tei:dataRef
09123	tei:content//rng:ref[@name = 'macro.anyXML']
12514	tei:content/tei:classRef   tei:content//tei:sequence/tei:classRef
12964	tei:dataSpec/tei:content//tei:dataRef
08632	tei:front//tei:titlePart/tei:title
10595	tei:label   tei:figure   tei:table   tei:item   tei:p   tei:title   tei:bibl   tei:anchor   tei:cell   tei:lg   tei:list   tei:sp
12962	tei:macroSpec/tei:content//rng:ref

Table 3: Experiment results on the XPATH subcorpus from Lick and Schmitz in Table 2. For each automaton we present: size(number-of-states).

$Q$	A = $SHA(Q)$	$det(A)$	B = $det_{one^x}(A)$	$scl_{x-doc}(B)$	C = $det_{one^x \times x-doc}(A)$	mini(C)
18330	89 (37)	400 (38)	127 (22)	73 (22)	73 (22)	60 (18)

17914	161 (67)	4071 (151)	308 (49)	178 (49)	178 (49)	87 (24)
10745	155 (62)	586 (49)	256 (34)	131 (34)	131 (34)	125 (32)
02091	114 (48)	1088 (66)	128 (23)	79 (23)	79 (23)	65 (18)
00744	147 (62)	2038 (109)	183 (30)	97 (30)	97 (30)	53 (15)
12060	80 (32)	333 (34)	97 (18)	62 (18)	62 (18)	50 (14)
02762	153 (64)	3143 (132)	190 (34)	118 (34)	118 (34)	52 (14)
06027	147 (62)	11244 (263)	161 (28)	95 (28)	95 (28)	52 (14)
02909	112 (45)	594 (52)	202 (29)	106 (29)	106 (29)	94 (25)
06415	153 (64)	5736 (169)	247 (40)	149 (40)	149 (40)	109 (27)
03257	143 (58)	3961 (164)	380 (51)	256 (51)	256 (51)	98 (22)
05122	99 (40)	715 (55)	172 (27)	109 (27)	109 (27)	71 (18)
09138	349 (152)		410 (59)	198 (59)	198 (59)	62 (15)
05460	214 (88)	38163 (610)	1039 (173)	628 (173)	628 (173)	69 (18)
12404	100 (40)	521 (47)	154 (24)	83 (24)	83 (24)	71 (20)
10337	108 (43)	566 (50)	184 (27)	98 (27)	98 (27)	86 (23)
06639	134 (54)	1199 (79)	243 (42)	149 (42)	149 (42)	67 (18)
14340	69 (29)	186 (24)	94 (18)	57 (18)	57 (18)	44 (14)
13804	90 (37)	314 (35)	171 (28)	92 (28)	92 (28)	72 (22)
13896	116 (46)	146 (23)	124 (23)	92 (23)	92 (23)	80 (18)
02194	71 (29)	208 (26)	112 (20)	65 (20)	65 (20)	52 (16)
06726	129 (56)	1400 (87)	182 (30)	96 (30)	96 (30)	54 (16)
13640	90 (37)	314 (35)	170 (26)	85 (26)	85 (26)	75 (23)
05735	101 (41)	362 (39)	210 (30)	105 (30)	105 (30)	95 (27)
15766	134 (54)	609 (55)	318 (41)	154 (41)	154 (41)	134 (35)
15524	115 (46)	421 (44)	259 (35)	129 (35)	129 (35)	119 (32)
06512	120 (49)	407 (42)	202 (34)	114 (34)	114 (34)	74 (22)
06176	1191 (550)		5271 (386)		1202 (386)	
12539	159 (68)	1965 (112)	311 (40)	137 (40)	137 (40)	100 (28)
11780	166 (70)	2286 (128)	317 (41)	142 (41)	142 (41)	98 (27)
11478	91 (37)	315 (35)	171 (26)	86 (26)	86 (26)	76 (23)
11227	148 (59)	583 (53)	337 (42)	163 (42)	163 (42)	144 (37)
05684	1276 (580)		4044 (226)		718 (226)	123 (16)
06947	704 (322)		1575 (129)	443 (129)	443 (129)	70 (14)
06794	256 (114)		399 (51)	177 (51)	177 (51)	63 (15)
06169	328 (146)		516 (62)	218 (62)	218 (62)	66 (14)
06924	566 (258)		1141 (105)		361 (105)	66 (14)
11958	102 (40)	347 (35)	143 (24)	89 (24)	89 (24)	75 (20)
01705	732 (330)		2424 (172)	745 (172)	745 (172)	112 (19)

02086	767 (346)		2605 (180)	780 (180)	780 (180)	114 (19)
02000	608 (274)		1212 (110)		386 (110)	82 (14)
02697	363 (162)		589 (68)	239 (68)	239 (68)	68 (14)
14183	103 (40)	361 (36)	154 (25)	93 (25)	93 (25)	79 (21)
07106	433 (194)		747 (80)	281 (80)	281 (80)	72 (14)
05824	78 (32)	305 (32)	79 (16)	54 (16)	54 (16)	42 (12)
11368	118 (48)	1233 (76)	187 (32)	118 (32)	118 (32)	86 (22)
15848	120 (47)	293 (34)	175 (26)	100 (26)	100 (26)	97 (25)
15462	114 (44)	332 (36)	302 (36)	113 (28)	113 (28)	108 (27)
04267	83 (32)	145 (20)	80 (15)	53 (15)	53 (15)	50 (14)
07113	695 (296)		7574 (302)		7570 (302)	365 (49)
03864	258 (114)		382 (50)	176 (50)	176 (50)	62 (14)
15484	173 (66)	656 (62)	411 (50)	188 (47)	188 (47)	173 (42)
15461	134 (52)	685 (56)	423 (48)	155 (37)	155 (37)	141 (34)
11160	293 (130)		447 (56)	197 (56)	197 (56)	64 (14)
06856	290 (130)		465 (57)	197 (57)	197 (57)	58 (14)
06458	783 (354)		1777 (140)		491 (140)	92 (14)
13710	398 (178)		666 (74)	260 (74)	260 (74)	70 (14)
06808	497 (226)		948 (93)	320 (93)	320 (93)	64 (14)
04338	450 (196)		936 (135)	562 (135)	562 (135)	94 (22)
04358	966 (424)		3354 (433)		2020 (433)	
13632	132 (58)	339 (33)	216 (33)	113 (33)	113 (33)	47 (9)
01847	531 (238)		1142 (137)	542 (137)	542 (137)	91 (19)
05219	664 (298)		1422 (164)	650 (164)	650 (164)	93 (19)
05226	874 (394)		2184 (218)	866 (218)	866 (218)	105 (19)
03325	713 (322)		1539 (128)	449 (128)	449 (128)	88 (14)
03410	888 (402)		2164 (158)	554 (158)	554 (158)	98 (14)
03407	678 (306)		1426 (122)	428 (122)	428 (122)	86 (14)
04245	853 (386)		2031 (152)		533 (152)	96 (14)
04953	888 (402)		2164 (158)		554 (158)	98 (14)
07095	363 (162)		589 (68)	239 (68)	239 (68)	68 (14)
05463	150 (60)	1179 (70)	256 (38)	151 (38)	151 (38)	77 (20)
12960	160 (64)	1339 (81)	316 (46)	189 (46)	189 (46)	145 (33)
12961	159 (64)	1317 (80)	302 (45)	185 (45)	185 (45)	141 (32)
09123	157 (60)	704 (59)	367 (43)	174 (43)	174 (43)	163 (40)
12514	161 (66)	2733 (112)	238 (38)	145 (38)	145 (38)	113 (28)
12964	121 (48)	559 (48)	192 (31)	119 (31)	119 (31)	95 (24)
08632	121 (48)	628 (51)	192 (31)	119 (31)	119 (31)	95 (24)

Q	A= <i>sha</i> (Q)	<i>det</i> (A)	<i>scl</i> <sub><i>one</i><sup><i>x</i></sup></sub> ( <i>det</i> (A))	<i>det</i> <sub><i>one</i><sup><i>x</i></sup></sub> (A)	<i>scl</i> <sub><i>x-doc</i></sub> ( <i>scl</i> <sub><i>one</i><sup><i>x</i></sup></sub> ( <i>det</i> (A)))	<i>det</i> <sub><i>one</i><sup><i>x</i></sup> × <i>x-doc</i></sub> (A)
A1	135 (57)	149 (37)	149 (37)	149 (37)	107 (37)	107 (37)
A2	106 (42)	468 (58)	187 (36)	187 (36)	96 (32)	96 (32)
A3	113 (46)	332 (46)	211 (35)	211 (35)	116 (34)	116 (34)
A4	162 (67)	171 (42)	171 (42)	171 (42)	124 (42)	124 (42)
A5	181 (71)	440 (57)	420 (57)	420 (57)	259 (55)	259 (55)
A6	239 (91)	193 (45)	193 (45)	193 (45)	145 (45)	145 (45)
A7	161 (65)	174 (41)	170 (41)	170 (41)	126 (41)	126 (41)
A8	896 (268)	753 (124)	643 (118)	643 (118)	528 (118)	528 (118)

Fig. 15: Regular Path queries of XPathMark benchmark

10595	433 (194)		747 (80)	281 (80)	281 (80)	72 (14)
12962	122 (48)	575 (49)	204 (32)	123 (32)	123 (32)	99 (25)

## 9 Other Benchmarks

For our first set of experiments, we used the forward fragment that can be compiled to *SHA*, of the usual XML benchmark XPathMark [10]. The results are shown in Fig. 15. The queries A1–A8 are first compiled to their *SHA* equivalents, their respective size and number of states are presented in the second column, namely *A*. Then, we determinized, schema-cleaned *det*(*A*) and schema-based determinized *A*, firstly with *one*<sup>*x*</sup> alone, then with *one*<sup>*x*</sup> and *xdoc*. The results are shown in the consequent four columns and the size of the resultant automata were relatively small. The test on this collection confirmed that the integration of schema-based cleaning into determinization yields the same outcome that schema-cleaning after determinization with *x-doc* and *one*<sup>*x*</sup>, as reported previously in [16]. Since the size of the automata remains relatively small, the gain remains quite limited for this collection.

A second collection of XPATH queries that we devised is the following *Qn.m* that be scaled with for all naturals *n* and *m*:

```
(Qn.m)    /*[self::a0 or ... or self::an]
          [descendant::*[self::b0 or ... or self::bm]]">
```

This query pattern select all elements which name is *a0* or *a1*, ..., *an* under the condition that these elements have some descendants *b0* or *b1*, ..., *bm*. An example of such query is **Q2.2** : `// * [self :: a0 or self :: a1][descendant :: * [self :: b0 or self :: b1]]`. We performed the same steps as per the first collection and the results for the queries **Q2.1**, ..., **Q3.4** with schemas *one*<sup>*x*</sup> and *one*<sup>*x*</sup> × *x-doc* are reported in Fig. 16. It is clear that size of the deterministic automata *det*(*A*) is growing relatively to the number of the disjunctions in the query and quickly, the schema-cleaning process deems inefficient. This is shown for queries **Q2.3**, **Q2.4**, **Q3.2**, **Q3.3** and **Q3.4** where schema-based cleaning could

Q	A= <i>sha</i> (Q)	<i>det</i> (A)	<i>scl</i> <sub><i>one</i><sup><i>x</i></sup></sub> ( <i>det</i> (A))	<i>det</i> <sub><i>one</i><sup><i>x</i></sup></sub> (A)	<i>scl</i> <sub><i>x-doc</i></sub> ( <i>scl</i> <sub><i>one</i><sup><i>x</i></sup></sub> ( <i>det</i> (A)))	<i>det</i> <sub><i>one</i><sup><i>x</i></sup> × <i>x-doc</i></sub> (A)
Q2.1	165 (63)	2302 (125)	588 (67)	588 (67)	391 (67)	391 (67)
Q2.2	196 (74)	5290 (231)	1288 (117)	1288 (117)	973 (117)	973 (117)
Q2.3	227 (85)	11878 (433)		2668 (203)		2187 (203)
Q2.4	258 (96)	26650 (835)		5436 (361)		4705 (361)
Q3.1	205 (77)	4341 (193)	735 (82)	735 (82)	488 (82)	488 (82)
Q3.2	243 (90)	9849 (363)		1595 (144)		1210 (144)
Q3.3	281 (103)	22097 (697)		3345 (256)		2760 (256)
Q3.4	300 (112)	57170 (2219)		9324 (613)		8302 (613)
QN7	322 (132)	1633790 (10003)		499 (72)		248 (72)

Fig. 16: A scaling collection of path queries with schema  $one^x$  and  $x\text{-doc} \times one^x$ . The open fields could not be computed within 30 minutes or ran out of memory.

no more be applied afterwards. With schema-based determinization, however, we could obtain small deterministic automata without problems. One could think of scaling up to test the limits of schema-based determinization in practice, but it is clear that even with our killer example **QN7** of size more than 1.6 million, we were still able to determinize it with both  $one^x$  and  $one^x \times x\text{-doc}$ . **QN7** query is the following:

```
(QN7)    /a/b//(* | @* | comment() | text())
```

It selects all nodes of an XML document that are descendants of a  $b$ -element below an  $a$ -element at the root. The node may have any XML type: element, attribute, comment, or text. The nondeterministic *SHA* has 132 states and an overall size of 322. Its determinization however leads to an automaton with 10.003 states and an overall size of 1.633.790.