A Benchmark Collection of Deterministic Automata for XPath Queries

Antonio Al Serhali, Joachim Niehren

A Benchmark Collection of Deterministic Automata for 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 (SHAs), and determinize them. Large blowups by automata determinization are avoided by using schema-based determinization. The schema captures the *XML* data model and the fact that a variable must be bound to a single node by any query answer. Our collection also provides deterministic symbolic nested word automata that we obtain by compilation from deterministic symbolic SHAs.

Keywords: *XML*, regular path queries, automata, nested words, trees, hedges.

1 Introduction

XML is one of the most used standardized formats for representing exchanging stuctured data between various tools and applications. *XML* documents form unranked data trees. Processing *XML* documents in both in-memory and streaming modes are widely studied [23,24,27,26,25] for many years. The most frequent tasks are validating, querying and transforming data trees or hedges, also called nested words [22].

Algorithms for validating and querying data trees or other kinds of hierarchically structured documents are often based on automata [14,2,5]. Determinism is 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 also applies to the problem of answering regular path queries in streaming mode [7,11]. It is also relevant for enumerating query answers of document spanners in in-memory mode [8,19].

Deterministic automata for regular queries nested words [12,15,9] are needed for benchmarking algorithms for these problems. But only a few deterministic automata are available so far. For instance, [16] provide deterministic nested word automata (NWAs) of decent size for the 10 forward navigational XPath queries for the XPathMark benchmark [10]. The main problem leading to this situation is that automata determinization remains problematic. As already noticed in [6], the usual determinization algorithm for nested word automata (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 by avoiding the top-down determinism of NWA*s*. This can be done by either restricting NWA*s* so that they satisfy the weak single entry property, or more directly, by using stepwise hedge automata (SHAs) [16]. Both approaches have the same expressiveness up to quadratic time transformations.

As illustrated recently [17], even the determinization of SHAs may lead to unreasonably large automata, when trying to determinize stepwise automata for XPATH queries. The first solution to the problem is determinizing the product of the query automaton with the schema automaton. The schema here captures the *XML* data model and that the fixed selection variable x must be bound to exactly one node of the *XML* document by any query answer. Somehow surprisingly at the first moment, determinizing the potentially larger product often leads to considerably smaller deterministic automata. The intuition is that the deterministic schema automaton reduces the number of subsets of states that are to be considered during determinization since all states in such subsets must be aligned to the same schema state. A second approach is to clean the determinized automaton with respect to the schema. This means removing all states and transitions that cannot be aligned to the schema. Schema-based cleaning has the advantage of always yielding smaller automata. Unfortunately, however, it is not always computationally feasible, if the automaton produced by determinization is too large. The third solution is to use schema-based determinization [17]. The idea is to integrate schema-based cleaning directly into the determinization algorithm, avoiding large blowup from the beginning.

We implemented all three solutions and show that the first and third solution work adequately for all the regular XPATH queries in the benchmark corpus that Lick and Schmitz [29,13] harvested from XSLT and XQUERY programs available online (docbook, teixml, htmlbook, ...). The third solution based on schema-based determinization followed by minimization [17] yields surprisingly small deterministic automata with at most 58 states for the whole benchmark collection. In average there are 22 states and 71 transitions per automaton. All automata are published at `https://gitlab.inria.fr/aalserha/xpath-benchmark`.

The fact that we can indeed determinize the automata of most if not all practical XPATH queries with a mild size increase, gives new hope to improve the situation on *XML* streaming in the near future, building on approaches requiring deterministic automata [7,11,28].

Outline. We present our selection of regular XPATH benchmark queries from the corpus of Lick and Schmitz[29] in Section 2. Nested words and their relationship to *XML* documents are recalled in Section 3. A deterministic symbolic hedge automata defining the schema of valid *XML* documents is given in Section 3. A formal definition of symbolic stepwise hedge automata follows for the sake of self-containedness in Section 5. In Section 6 we discuss our compiler from XPath expressions to deterministic automata, and illustrate it by example automata from our benchmark collection. In Section 7 we discuss how we tested our automata for correctness on a sample of annotated *XML* documents produced from the XPATH query based on Saxon XSLT. The sizes of automata in our benchmark collection of symbolic dSHAs are discussed in Section 8.

2 XPath Benchmark Queries

We start with the collection of 21000 XPath queries that Lick and Schmitz [29] 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 [10], in contrast, focuses on functionality testing.

We then filter the subclass of around 4500 forward navigational XPath queries of Lick's 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 queries. We then selected the 180 largest queries of this subcorpus.

Finally, we removed duplicates of queries up to renaming of *XML* names and namespaces and syntactical details, such as $.//author$ or $descendant-or-self::author$ or $descendant-or-self::corpauthor$. This leads us to the collection of 79 queries. The first 10 queries are shown in Table 1. The full list of all 79 queries can be found in Table 3 of the appendix.

Table 1: The first 10 of the 79 queries of the benchmark collection (see Table 3).

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

We note that the XPath query 18339 is considered as large since it contains the recursive axis `descendant-or-self`. Other queries are considered as large since having a parse tree with more than 15 nodes, for instance 05684 and 05684.

3 Nested Words and XML Documents

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. Let Σ be a set that we call the alphabet. Nested words in $\mathcal{N}_\Sigma$ have the following abstract syntax.

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

We assume that concatenation $\cdot$ is associative, and that the empty word ε 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 Σ.

XML documents are labeled unranked trees that can be serialized into a text, such as for instance:

$$\langle s{:}a\ name{=}\text{“}uff\text{”}\rangle\langle s{:}b\rangle gaga\langle s{:}d/\rangle\langle/s{:}b\rangle\langle s{:}c/\rangle\langle/s{:}a\rangle$$

Labeled unranked trees satisfying the *XML* data model can be represented as nested words over a signature that contains the *XML* node-types ($elem, attr, text$, $comment$), the *XML* namespaces of the document(s), the *XML* names of the document ($a, \ldots, d, nam$), and the characters of the data values, say UTF8. For the above example, we get the nested word:

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

We also notice that the alphabet Σ_{XML} of *XML* document has 4 different types of letters: node types, namespaces, names, and characters.

4 A Schema for *XML* Documents with x-Annotation

Stepwise hedge automata (SHAs)[16] provide a graphical way to define regular languages of nested words. They can be compiled back and forth to nested word automata (NWAs) [18,1] in linear time, but are easier to determinize. Given that we have to deal with infinite alphabets, symbolic automata with else rules offer a natural way to define regular languages of *XML* documents. An example of symbolic SHA is given in Fig. 1; the formal definition follows in Section 5.

The most frequent XPATH queries select nodes of *XML* documents. For referring to selected nodes, we fix a single selection variable x. We call an *XML* document or subdocument, in which a single node is annotated by x, an *x-annotated example*. An x-annotated example is called *positive* for a query if the query selects the x-annotated node in the *XML* document, and *negative* otherwise. The symbolic SHA xml&one^x in Fig. 1 recognizes the set of all x-annotated examples: these must satisfy the *XML* data model and contain exactly one occurence of x.

The automaton starts in hedge state 0 where it expects to read a nested word $\langle w \rangle$, that can be evaluated to tree state 28, in order to go to the final state 29, where it accepts The sequence of children w of the tree must be evaluated form the tree initial state, which is equally the hedge state 0. If w starts with letter *doc* indicating an *XML* document node at the root, the automaton moves from state 0 to state 5. There it may either read the variable x and go to state 5, where it expects a subtree in state 21, i.e. an *XML* element of which no node is

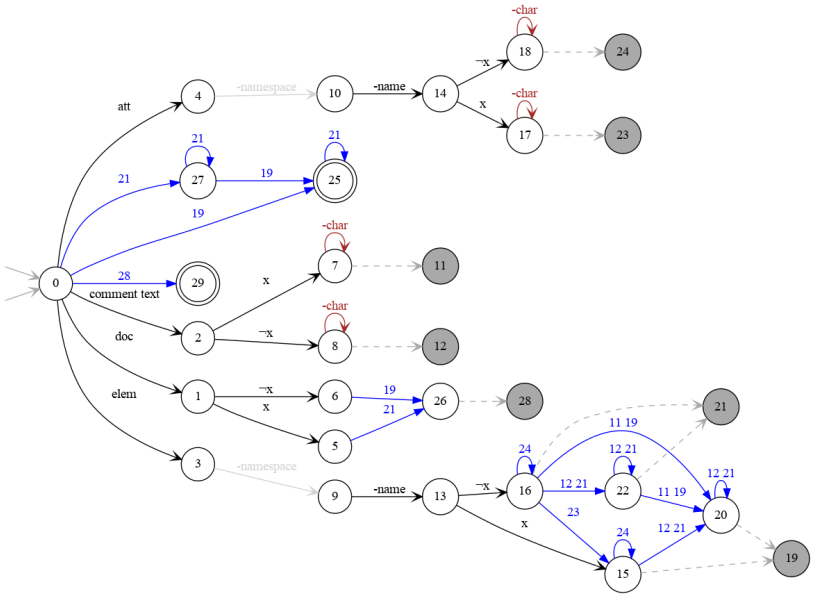

Fig. 1: The symbolic SHAS $\texttt{xml\&}one^x$: a schema for x-annotated *XML* documents.

annotated by x. Or it may read the symbol $\neg x$ and move to state 6, where it expects a subtree in state 19, i.e. an *XML* element of which exactely one node is annotated by x. In both cases it can go to the hedge state 26 and from there to the tree state 28. The automaton also states the relationships of elements, attributes, text and comment nodes according to the *XML* data model.

The alphabets of names and namespaces of *XML* documents are infinite. In order to represent infinite sets of transition rule symbolically in a finite manner, the automaton use type else rules. The typed else rule in state 3, for instance, is labeled by $-namespace$, permitting to read any namespace and to go to state 9. State 9 in turn has an else rule labeled by $-name$ which permits to read any (local) name and move to state 13.

5 Symbolic Stepwise Hedge Automata

Stepwise hedge automata on nested words extend on finite state automata on words. Both kinds of automata can be made symbolic by adding else rules. We start with the case of untyped alphabets.

Definition 1. *A* symbolic NFA *is a tuple* $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ *such that* Σ *is a possibly infinite set called the alphabet,* $\Delta = (\Delta', \_^{\Delta})$ *and* $A' = (\Sigma, \mathcal{Q}, \mathcal{P}, \Delta', I, F)$ *is a* NFA *with a finite set of transition rules* $\Delta' \subseteq (\mathcal{Q} \times \Sigma) \times \mathcal{Q}$ *and* $\_^{\Delta} \subseteq \mathcal{Q}^2$ *a set of (untyped) else rules.*

As usual for finite state automata, we draw SHAs as graphs whose nodes are the states. A hedge state $q \in \mathcal{Q}$ is drawn with a circle $\textcircled{q}$, an initial state

$q \in I$ with an incoming arrow $\to(q)$, and final states with a double circle $((q))$. A transition rule $(q_1, a, q_2) \in \Delta'$ is drawn as a black edge $(q_1) \xrightarrow{a} (q_2)$ that is labeled by a letter $a \in \Sigma$. We now come to the symbolic aspects. An (untyped) else rule $(q, q') \in \_^{\Delta}$ is drawn as $(q) \dashrightarrow (q')$. It means that the automaton in state q can go to state q' when reading any letter $a \in \Sigma$ such that there exists no q'' with $q \xrightarrow{a} q'' \in \Delta$. Any else rule can be expanded to a set of internal rules as follows:

$$\frac{q \dashrightarrow q' \in \Delta \quad a \in \Sigma \quad \neg\exists q'' \in \mathcal{Q}.\ q \xrightarrow{a} q'' \in \Delta}{q \xrightarrow{a} q' \in \Delta^{exp}} \qquad \frac{q \xrightarrow{a} q' \in \Delta}{q \xrightarrow{a} q' \in \Delta^{exp}}$$

Definition 2. *A symbolic* SHA *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 a symbolic* NFA*. Furthermore,* $\mathcal{P}$ *is a finite set of tree states and* $\Delta'' = (\langle\rangle^{\Delta}, @^{\Delta}, {-\!\!\to}^{\Delta})$*, such that* $\langle\rangle^{\Delta} \subseteq \mathcal{Q}$ *is a subset of tree initial states,* $@^{\Delta} \subseteq (\mathcal{Q} \times \mathcal{P}) \times \mathcal{Q}$ *a set of apply rules, and* ${-\!\!\to}^{\Delta} \subseteq \mathcal{Q} \times \mathcal{P}$ *a set of tree final rules.*

We draw symbolic SHAs as graphs extending on the graphs of symbolic NFAs. A tree state $p \in \mathcal{P}$ is drawn in gray (p). A tree initial state $q \in \langle\rangle^{\Delta}$ is a hedge state is drawn as $\xrightarrow{\langle\rangle}(q)$ with an incoming tree arrow. An apply rule $(q_1, p, q_2) \in @^{\Delta}$ is drawn by a blue edge $(q_1) \xrightarrow{p} (q_2)$ carrying 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 go into state $q_2 \in \mathcal{Q}$. A tree final rule $(q, p) \in {-\!\!\to}^{\Delta}$ is drawn as $(q) \dashrightarrow (p)$. It states that if w is a nested word in state $q \in \mathcal{Q}$ then $\langle h \rangle$ is a tree in state $p \in \mathcal{P}$.

Transitions of symbolic SHAs have the form $q \xrightarrow{w} q'$ wrt Δ where $w \in \mathcal{N}_\Sigma$ and $q, q' \in \mathcal{Q}$. They are defined by the inference rules:

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

$$\frac{q' \in \langle\rangle^{\Delta} \qquad q' \xrightarrow{w} q \text{ wrt } \Delta \qquad q -\!\!\to p \in \Delta \qquad q_1 \xrightarrow{p} q_2 \in \Delta}{q_1 \xrightarrow{\langle w \rangle} q_2 \text{ wrt } \Delta}$$

The last inference rule says that when reading a tree $\langle w \rangle$ the automaton can transit from a state q_1 to a state q_2 if with w it can transit from some tree initial state q' to q, so that there is some tree final rule $q -\!\!\to p \in \Delta$ and some apply rule $q_1 \xrightarrow{p} q_2 \in \Delta$. The language $\mathcal{L}(A)$ of a symbolic SHA is defined as usual for NFAs, just that nested words may be recognized too:

$$\mathcal{L}(A) = \{w \in \mathcal{N}_\Sigma \mid q \xrightarrow{w} q' \text{ wrt } \Delta,\ q \in I,\ q' \in F\}$$

The notion of determinism for symbolic SHAs extends on the notion of left-to-right determinism of symbolic NFAs and on the notion of bottom-up determinism of tree automata.

Definition 3. *We call a symbolic* Nfa A *deterministic or equivalently a symbolic* Dfa *if* Δ' *and* $\_^{\Delta}$ *are partial function. We call a symbolic* Sha A deterministic *or equivalently a symbolic* d*Sha, if the contained finite automaton* A' *is a symbolic* Dfa*, there is at most one tree initial state in* $\langle\rangle^{\Delta}$*, and* $@^{\Delta}$ *and* $\longrightarrow^{\Delta}$ *are partial functions.*

Adding Typed Else Rules. Suppose that the alphabet Σ is typed, in that any letter $a \in \Sigma$ can be given some types in some type set T. We can then add typed else rules o the following form symbolic Nfas and symbolic Shas:

$$q \xrightarrow{-\tau} q' \in \Delta$$

where $\tau \in T$. We draw them as $\textcircled{q} \xrightarrow{-\tau} \textcircled{q'}$. In contrast to untyped else rules, a typed else rule cannot be expanded with all letters $a \in \Sigma$, but only with those that can be given the type τ.

6 Compiler to Automata

We extended on the compilation chain for regular XPath queries to automata from [16]. As a running example, we consider the following query:

Q_2 : `h:body[@lang != '']`

Query Q_2 selects a node if it has a child named `body` in namespace `h`, that has the attribute node named `lang` containing a nonempty text.

Parser. Our parser for XPath expressions computes a parse tree following the grammar of XPath 3.1 from the W3C. In addition, it returns for any forward regular XPath expression a logical formula in the language FXP [6]. For the XPath example Q_2, we obtain the following FXP formula:

$$\begin{array}{l} child(lab_{elem:type} \wedge lab_{h:namespace} \wedge lab_{body:name} \wedge lab_{x:var} \wedge \\ \quad child(lab_{att:type} \wedge cand_{default:namespace} \wedge lab_{lang:name} \wedge string \neq'')) \end{array}$$

Our previous parser needed considerable improvement in order to be able to cover the large variety of queries from the corpus of Lick and Schmitz [29].

Nested Regular Expression. We next compile Fxp formulas to nested regular expressions, which extend on standard regular expressions from words to nested words. Again, considerable work was needed to enable a sufficiently large coverage. For the query Q_2 our compiler yields the nested regular expression:

$$\begin{array}{l} \langle (elem{:}type._) + doc{:}type)._{-}* . \\ \quad \langle elem{:}type.h{:}namespace.body{:}name.x{:}var. \\ \quad\quad \langle att.type.default{:}namespace.lang{:}name._{-}(\_{:}char.(\_{:}char)^* \rangle . * \rangle . * \rangle .* \end{array}$$

Note that the test for a nonempty string got translated by the regular expression $char.(\_{:}char)^*$. It should also be noticed that this expression matches some x-annotated nested words, that are not x-annotated examples, i.e. not belonging to the language $\mathcal{L}(xml\&one^x)$ of the schema. This is since the nested subwords matching expression $*$ are completely unconstrained.

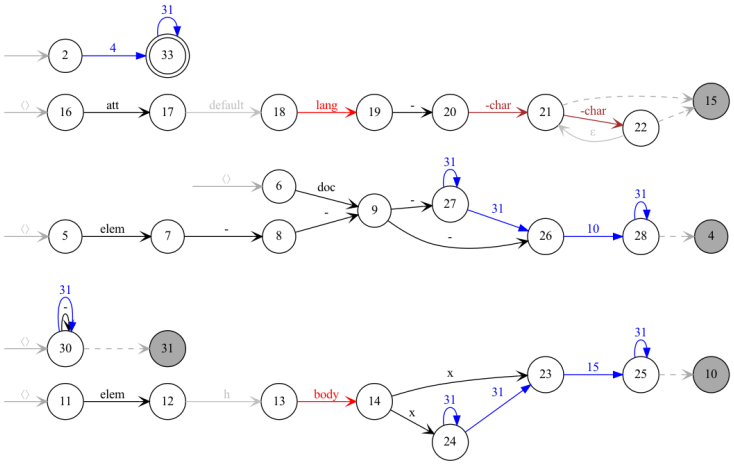

Fig. 2: A nondeterministic symbolic SHA for XPATH query Q_2.

Symbolic SHAs. The compiler then converts nested regular expressions into symbolic SHAs. This is done by extending a usual compiler from regular expressions to NFAs. The interaction of recursion and nesting leads to some nasty issues, that are discussed and resolved in [16]. For developping the present benchmark, we needed to add a treatment of typed wildcards such as _:*char*. This is done by introducing typed else rules. For the query Q_2 we obtain the nondeterministic symbolic SHA in Fig. 2. Similarly to the nested regular expression, this symbolic SHA recognizes some annotated nested words, that are not x-annotated examples, i.e., that do not belong to the language of the schema $L(xml\&one^x)$.

Determinization. The usual determinization algorithms for NFAs and tree automata can be lifted to a determinization algorithm for symbolic SHAs. When applied to query Q_2 however, we obtain a symbolic SHA with 25 states and 183 transition rules, which is much larger than one might expect. It is given in Fig. 5 of the appendix. Even worse, in some cases, the determinization algorithm does not finish after some hours.

Determinizing the Schema Product. Determinization applied to the product of the queries' automaton and the schema $xml\&one^x$ permits to compute deterministic automata for all queries of our benchmark within a timeout of 100 seconds. The result for Q_2 is a symbolic dSHA with 53 states and 110 transition rules given in Fig. 6 of the appendix. The overall size is smaller, and the automaton is much easier to understand, but the number of states increased.

Schema-Based Determinization. Schema-based determinization as proposed in [17] improves the situation further. For query Q_2 it yields the symbolic SHA in Fig. 3 with has only 22 states and 45 transitions. The size is roughly devided by 2 compared to determinizing the schema-product with the query automaton.

Minimization. We then minimize the symbolic dSHA from Fig. 3. This often reduces the size and the number of states in an important manner and often makes it easy to see how the automaton is functioning. Exceptionally in the case

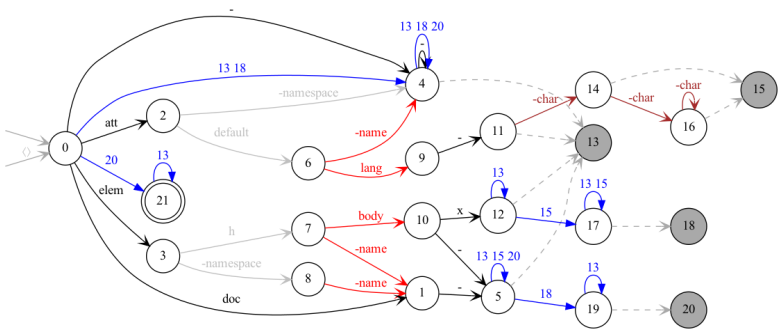

Fig. 3: The schema-based determinization of the symbolic SHA for query Q_2.

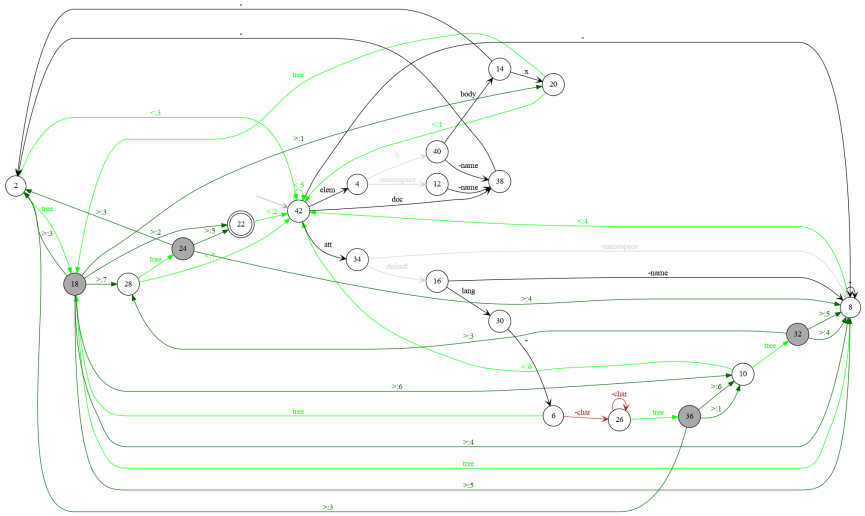

Fig. 4: The corresponding symbolic dNWA for query Q_2.

of Q_2, no states are fusioned when minimizing the symbolic dSHA obtained by schema-based determinization.

It should be noticed that minimizing the determinization of the schema product usually yields a different result then minimizing the schema-based determinization. This is since both automata may recognize different languages. Some nested words outside the schema may be accepted after schema-based determinization, but not by the schema product.

Symbolic NWAs. The compiler finally maps symbolic SHAs to symbolic NWAs in linear time, while preserving determinism. For instance the minimal symbolic dSHA in Fig. 3 is converted to the symbolic dNWA in Fig. 4.

7 Testing on Samples

For testing the automata, we created for each of the XPath queries an *XML* document, on which the query has a nonempty answer set. By using Saxon XSLT, we hen computed the answer set by, and produced for each of the answers a positive x-annotated examples. Negative x-annotated examples are obtained from the answers of the other queries on the same document. In this way, we obtained a sample with positive and negative x-annotated examples for each of the queries. The test samples can be provided by the authors in case of interest.

By testing the automaton on these samples, we could fix various problems. Currently, no test failures are remaining, except for the query 13896 below that we removed from the corpus for the current version. The problem here is raised by the blank symbol in the attribute value `'evans citation'`.

```
//HEADER//IDNO[@TYPE='evans citation']
```

8 Benchmark of Query Automata

We first compiled all of our 79 XPath queries symbolic Sha using the compilation chain described in Section 6. These Shas were also cleaned using the schema $xml\&one^x$ in Fig. 1. The statistics of the symbolic Shas are shown in the column A of Table 2. For each query, we find two numbers $size(\#states)$, where $size$ is the overall size of the automaton and $\#states$ the number of its states.

We note that 37% of the symbolic Shas in column A have more than 100 states. The biggest is for query 06176 with 630 states and an overall size of 1391. The reason is that this query is selecting a union of 20 subqueries, all with descendant-or-self axis. Foe each subquery, we have 4 constructs of respective state sizes: 2, 6, 10 and 13, making a subtotal of $31*20 = 620$. With an additional 8 states for one subquery that select all descendants with an attribute named *id* and another 2 for reading any tree, we end up with our total 630 states.

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

query id	A	det(A)	$B =$ det($A \times S$)	$C =$ $det_S(A)$	$B' =$ $mini(B)$	$C' =$ $mini(C)$	$nwa(C')$
18330	99 (41)	465 (43)	145 (44)	74 (22)	128 (39)	61 (18)	73 (18)
17914	179 (75)	2740 (141)	265 (69)	150 (44)	152 (43)	82 (24)	98 (24)
10745	187 (76)	939 (68)	275 (72)	141 (38)	218 (57)	130 (34)	150 (34)
02091	100 (42)	555 (45)	182 (57)	81 (24)	146 (44)	61 (17)	75 (17)
00744	109 (46)	335 (37)	169 (54)	80 (24)	128 (41)	54 (15)	64 (15)
12060	64 (25)	162 (22)	139 (44)	56 (16)	121 (39)	44 (12)	54 (12)
02762	121 (50)	564 (53)	222 (63)	97 (28)	123 (39)	46 (12)	56 (12)
06027	115 (48)	1101 (79)	184 (57)	82 (24)	123 (39)	46 (12)	56 (12)

02909	96 (38)	311 (36)	213 (62)	100 (27)	167 (49)	91 (24)	105 (24)
06415	139 (58)	1793 (93)	300 (74)	135 (36)	229 (55)	101 (25)	123 (25)
03257	130 (53)	1310 (92)	445 (85)	224 (46)	210 (49)	87 (20)	105 (20)
05122	83 (33)	292 (33)	221 (55)	92 (23)	161 (44)	63 (16)	77 (16)
09138	269 (117)		323 (97)	164 (49)	133 (40)	56 (13)	66 (13)
05460	232 (98)	3468 (174)	509 (127)	269 (77)	156 (44)	62 (16)	76 (16)
12404	84 (33)	258 (31)	170 (52)	77 (22)	143 (44)	68 (19)	82 (19)
10337	92 (36)	291 (34)	197 (58)	92 (25)	159 (47)	83 (22)	97 (22)
06639	123 (50)	516 (49)	237 (65)	106 (30)	154 (44)	60 (16)	74 (16)
14340	79 (33)	231 (29)	126 (40)	58 (18)	110 (36)	45 (14)	55 (14)
13804	70 (29)	155 (21)	128 (41)	63 (20)	124 (40)	60 (19)	70 (19)
02194	81 (33)	253 (31)	135 (42)	66 (20)	119 (38)	53 (16)	63 (16)
06726	149 (64)	2806 (149)	176 (53)	97 (30)	121 (38)	55 (16)	65 (16)
13640	100 (41)	364 (40)	165 (50)	86 (26)	140 (43)	76 (23)	90 (23)
05735	111 (45)	412 (44)	201 (58)	106 (30)	161 (47)	96 (27)	110 (27)
15766	144 (58)	669 (60)	300 (77)	155 (41)	219 (57)	135 (35)	151 (35)
15539	217 (88)	1709 (121)	402 (98)	213 (58)	228 (57)	144 (38)	164 (38)
15809	197 (84)	3795 (188)	230 (67)	129 (39)	145 (43)	82 (24)	96 (24)
15524	125 (50)	471 (49)	245 (68)	130 (35)	185 (52)	120 (32)	134 (32)
06512	135 (56)	583 (58)	218 (60)	117 (35)	152 (43)	77 (23)	91 (23)
06176	1391 (630)		1661 (448)	1203 (386)	176 (43)	113 (23)	127 (23)
12539	179 (76)	3479 (174)	243 (69)	138 (40)	166 (48)	101 (28)	115 (28)
11780	205 (88)	3832 (190)	254 (71)	143 (41)	164 (47)	99 (27)	113 (27)
11478	101 (41)	365 (40)	166 (50)	87 (26)	141 (43)	77 (23)	91 (23)
11227	153 (62)	583 (53)	334 (81)	163 (42)	244 (59)	144 (37)	166 (37)
05684	1348 (616)		1068 (284)	719 (226)	193 (39)	124 (16)	134 (16)
06947	744 (342)		828 (232)	444 (129)	151 (41)	71 (14)	83 (14)
06794	270 (121)		354 (102)	178 (51)	144 (42)	64 (15)	76 (15)
06169	346 (155)		427 (121)	219 (62)	147 (41)	67 (14)	79 (14)
06924	598 (274)		682 (192)	362 (105)	147 (41)	67 (14)	79 (14)
11958	109 (44)	348 (35)	213 (57)	90 (24)	178 (48)	76 (20)	94 (20)
01705	772 (350)		1308 (279)	746 (172)	221 (48)	113 (19)	129 (19)
02086	809 (367)		1366 (291)	781 (180)	223 (48)	115 (19)	131 (19)
02000	642 (291)		723 (201)	387 (110)	163 (41)	83 (14)	95 (14)
02697	383 (172)		464 (131)	240 (68)	149 (41)	69 (14)	81 (14)
14183	110 (44)	362 (36)	217 (58)	94 (25)	182 (49)	80 (21)	98 (21)
07106	457 (206)		538 (151)	282 (80)	153 (41)	73 (14)	85 (14)
05824	62 (25)	150 (21)	130 (42)	50 (15)	112 (37)	38 (11)	48 (11)

11368	102 (41)	458 (44)	247 (62)	104 (28)	191 (49)	78 (20)	96 (20)
15848	124 (49)	303 (35)	221 (63)	103 (27)	179 (51)	100 (26)	114 (26)
15462	127 (50)	325 (37)	237 (67)	112 (29)	191 (54)	109 (28)	123 (28)
04267	87 (34)	146 (20)	137 (43)	54 (15)	131 (41)	51 (14)	63 (14)
07113	695 (296)		2409 (456)	1527 (302)	311 (73)	229 (48)	241 (48)
03864	272 (121)		353 (101)	177 (50)	143 (41)	63 (14)	75 (14)
15484	181 (71)	657 (62)	394 (96)	189 (47)	277 (68)	174 (42)	194 (42)
15461	146 (58)	628 (54)	651 (109)	283 (51)	241 (59)	140 (33)	160 (33)
11160	309 (138)		390 (111)	198 (56)	145 (41)	65 (14)	77 (14)
06856	306 (138)		390 (112)	198 (57)	139 (41)	59 (14)	71 (14)
06458	827 (376)		908 (251)	492 (140)	173 (41)	93 (14)	105 (14)
13710	420 (189)		501 (141)	261 (74)	151 (41)	71 (14)	83 (14)
06808	525 (240)		609 (172)	321 (93)	145 (41)	65 (14)	77 (14)
04338	470 (206)		1066 (207)	563 (135)	213 (51)	95 (22)	115 (22)
04358	1006 (444)		3580 (559)	2021 (433)	757 (99)	345 (58)	401 (58)
13632	132 (58)	339 (33)	248 (66)	113 (33)	128 (36)	47 (9)	55 (9)
01847	559 (252)		1013 (223)	543 (137)	194 (48)	92 (19)	108 (19)
05219	698 (315)		1192 (260)	651 (164)	196 (48)	94 (19)	110 (19)
05226	920 (417)		1558 (338)	867 (218)	208 (48)	106 (19)	122 (19)
03325	753 (342)		834 (231)	450 (128)	169 (41)	89 (14)	101 (14)
03410	938 (427)		1019 (281)	555 (158)	179 (41)	99 (14)	111 (14)
03407	716 (325)		797 (221)	429 (122)	167 (41)	87 (14)	99 (14)
04245	901 (410)		982 (271)	534 (152)	177 (41)	97 (14)	109 (14)
04953	938 (427)		1019 (281)	555 (158)	179 (41)	99 (14)	111 (14)
05463	204 (86)	1180 (70)	332 (77)	152 (38)	180 (48)	78 (20)	96 (20)
12960	167 (68)	1340 (81)	421 (88)	190 (46)	317 (64)	146 (33)	176 (33)
12961	166 (68)	1318 (80)	417 (87)	186 (45)	313 (63)	142 (32)	172 (32)
09123	164 (64)	705 (59)	358 (90)	175 (43)	265 (66)	164 (40)	186 (40)
12514	182 (77)	2734 (112)	320 (77)	146 (38)	247 (57)	114 (28)	140 (28)
12964	128 (52)	560 (48)	277 (67)	120 (31)	219 (53)	96 (24)	118 (24)
08632	128 (52)	629 (51)	277 (67)	120 (31)	219 (53)	96 (24)	118 (24)
12962	129 (52)	576 (49)	281 (68)	124 (32)	223 (54)	100 (25)	122 (25)

The second column for $\det(A)$ contains the statistics for the determinization of A. No schema is used there. We use a timeout of 100 seconds. Whenever this is not enough, the cell in the table is left blank. Indeed, the determinization fails with this timeout for 37% of the queries of our corpus. Roughly, the determinization fails for all symbolic SHAs with more than 100 states. For instance,

for query 11780 the symbolic SHA A has has size 205(88), while the symbolic dSHA $\det(A)$ has size 3832(190).

The third column contains the determinization $B = \det(A \times S)$ of the product of the SHA A and the schema $S = xml\&one^x$. Even though $A \times S$ is always larger than A, we were able to always determinize $A \times S$ within the timeout, in contrast to A. The largest dSHA B obtained is for for query 04358: it has size 3580(559). This show that B may still be quite big, but often a big improvement in size over $\det(A)$.

The fourth column reports on $C = det_S(A)$ obtained by schema-based determinization with schema $S = xml\&one^x$. Again, the computation succeeds in all cases within the timeout of 100 seconds. The size of C for query 04358 is 2021(433), which improves in size over B.

In the next two columns, we respectively minimize the determinized SHA*s* B and C, using a naïve minimization algorithm. All automata can be minimized within the timeout of 100 seconds. We note that $C' = mini(C)$ is always smaller than $B' = mini(B)$, showing that schema-based determinization yields smaller minimal automata than determinizing the schema-product. The maximal number of states of the minimal symbolic dSHAs $C' = mini(C)$ is 58 for query 04358. In average the number of states decreases by 55%.

In the last column, we compiled the minimized dSHA*s* of C' to the dNWA $nwa(C')$. It has the same number of states than C' for all queries and a minor increase is the number of transitions. All these results, including the automata of the intermediate steps, generated during the whole compilation chain are available at the following url: `https://gitlab.inria.fr/aalserha/xpath-benchmark`.

9 Conclusion

We provide a benchmark of deterministic automata for regular XPATH queries obtained with an algorithm for schema-based determinization of symbolic SHAs 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[29]. While 37% of symbolic SHAs obtained from these FXP queries cannot be determinized in less then 100 seconds by starndard determinization; schema-based determinization succeeds for 100% of them. Furthermore, all symbolic dSHAs obtained by schema-based determinization are sufficiently small so that they can be minimized with the naïve quadratic algorithm. This leads us to a collection of minimal symbolic dSHAs with an average number of states of 22, and 71 as the average number of transitions.

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).
22. R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.
23. Kay, M. The saxon XSLT and XQuery processor, 2004. https://www.saxonica.com.
24. Labath P. and Niehren J.. A Uniform Programming Language for Implementing XML Standards. In SOFSEM 2015.
25. Gauwin O.. Streaming Tree Automata and XPath. PhD thesis, Université Lille 1, 2009.
26. Genevès P. and Layaïda N. A System for the Static Analysis of XPath. ACM Trans. Inf. Syst., October 2006.
27. Gottlob G., Koch C. and Pichler R.. The complexity of XPath query evaluation. In 22nd ACM SIGMOD-SIGACT-SIGART, 2003.
28. Sakho M. Certain Query Answering on Hyperstreams. Phd Thesis. Université de Lille; Inria, 2020

29. Lick, A., Schmitz S.: XPath Benchmark `https://archive.softwareheritage.org/browse/directory/1ea68cf5bb3f9f3f2fe8c7995f1802ebadf17fb5/`. Last visited April 13th 2022.

Table 3: The 79 largest forward navigational queries of the XPath corpus of Lick and Schmitz without dublicates 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 | @valign | @width | @xml:id | @xml:lang
06947	anchor | areaset | audiodata | audioobject | beginpage | constraint | indexterm | itermset | keywordset | msg | doc:anchor | doc:areaset | doc:audiodata | doc:audioobject | doc:beginpage | doc:constraint | doc:indexterm | doc:itermset | 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

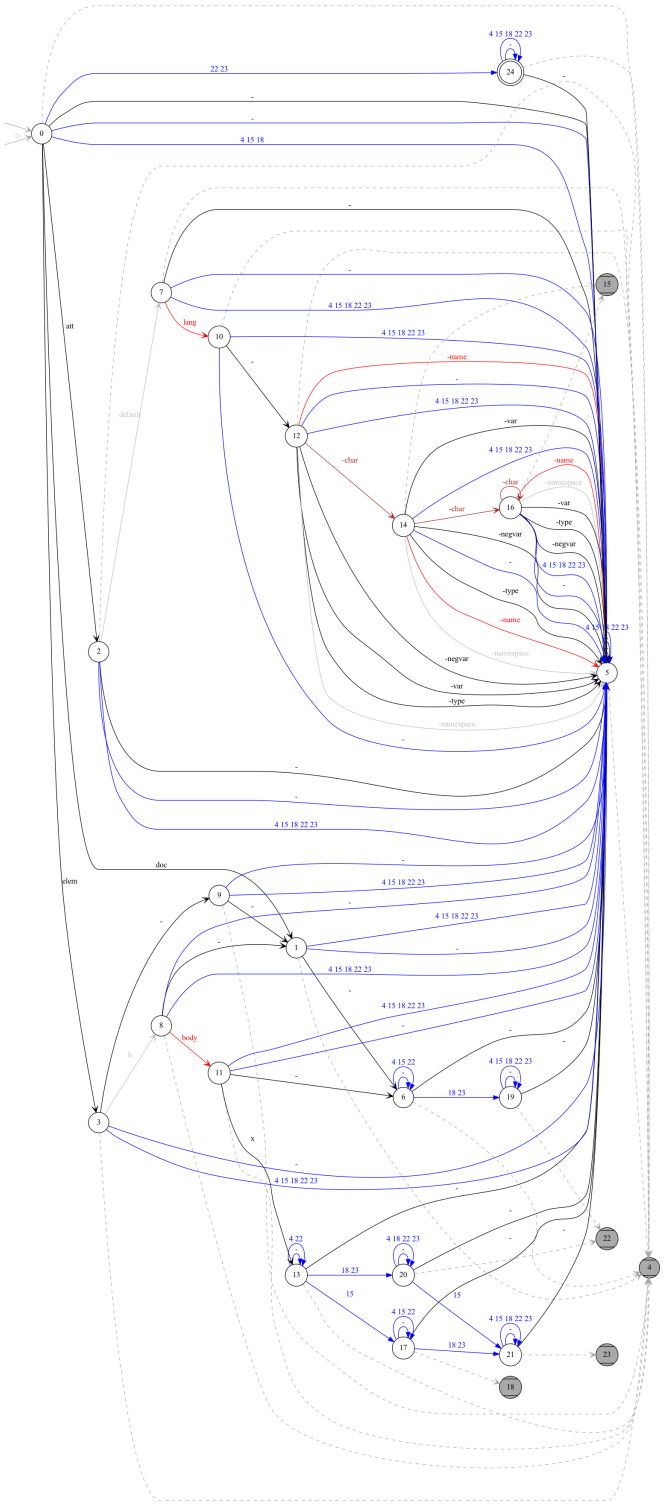

Fig. 5: The determinization of the symbolic SHA for query Q_2.

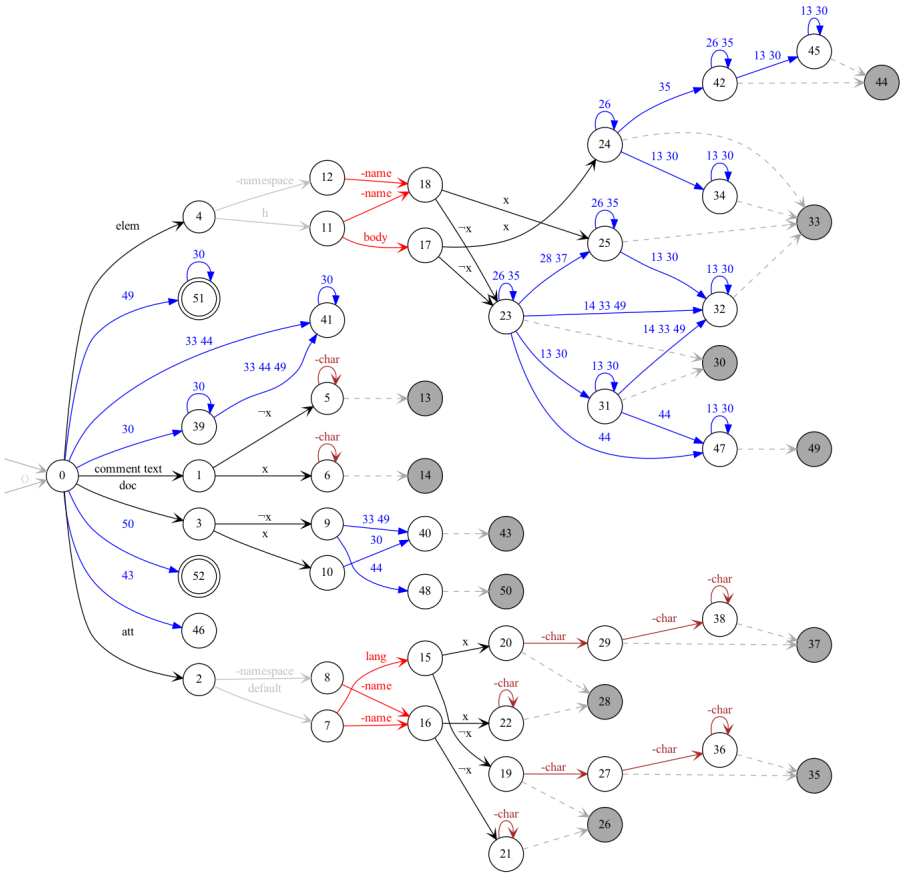

Fig. 6: The determinization of the product of the symbolic SHA for query Q_2 with the schema $xml\&one^x$.