Schema-Based Automata Determinization

Joachim Niehren, Momar Sakho, Antonio Al Serhali

Schema-Based Automata Determinization

Joachim Niehren
Inria, France Université de Lille
joachim.niehren@inria.fr

Momar Sakho
Inria, France Université de Lille
momar.sakho@inria.fr

Antonio Al Serhali
Inria, France Université de Lille
antonio.al-serhali@inria.fr

We propose an algorithm for schema-based determinization of finite automata on words and of stepwise hedge automata on nested words. The idea is to integrate schema-based cleaning directly into automata determinization. We prove the correctness of our new algorithm and show that it is always more efficient than standard determinization followed by schema-based cleaning. Our implementation permits to obtain a small deterministic automaton for an example of an XPath query, where standard determinization yields a huge stepwise hedge automaton for which schema-based cleaning runs out of memory.

1 Introduction

Schemas and queries for nested words are definitions of languages of words, trees, unranked trees, and hedges. They can model and query structured text documents such as XML and JSON documents.

Regular queries and schemas can be defined by deterministic nested word automaton (NWA) [24, 8, 2, 27]. An example for a regular schema is the XML data model and examples for regular queries are the forward navigational XPath queries on *XML* documents. Path queries can be seen as logical queries with a single free variable x. Such queries define languages of nested words containing exactly one occurrence of x. This property in turn, can be expressed by a schema one^x over the alphabet extended by the query's variable x.

Stepwise hedge automata (SHAs) provide alternative definitions for regular schemas and queries on nested words [26]. SHAs naturally subsume finite automata on words (NFAs) as well as stepwise tree automata for unranked trees [10]. SHAs have the same expressive power as NWA*s* modulo P-time transformations. Furthermore, SHAs come with a notion of determinism that generalizes on the left-to-right determinism of NFAs and on the bottom-up determinism of tree automata. An algorithm for determinizing SHAs can be based on the usual subset construction in contrast to NWA*s*. This becomes possible since SHAs do not support any form of top-down determinism, which for NWA*s* can be ruled out by imposing the weak single-entry property.

There exist compilers from forward navigational XPath queries to NWA*s* [4, 25, 12] and to SHAs [26]. The resulting automata can be determinized in, at most, exponential time. Deterministic automata are crucial for algorithmic tasks such as universality or inclusion checking. Both problems can be decided in P-time for the classes of deterministic SHAs or NWA*s*, but is DEXPTIME complete in the nondeterministic cases. For instance, universality checking is relevant for the earliest query answering of XPath queries on XML streams [17]. It is also relevant for the efficient enumeration of the answer sets of regular queries [28]. The need for deterministic automata in these applications motivates the question of how to determinize automata for nested words in practice.

Debarbieux et al. [12] noticed that the usual determinization algorithm for NWA*s* often behaves badly when applied to NWA*s* obtained from XPath queries as simple as $//a/b$. As recalled in the survey of Okhotin and Salomaa [27], this determinization algorithm was first invented by von Braunmühl and Verbeek in the eighties in the journal version of [8] and then rediscovered various times later on, for

Submitted to:
Gandalf 2022

instance in [3]. Niehren and Sakho [26, 7] observed recently that the situation is different for the determinization of SHAs: It works out nicely for the SHA of $//a/b$ and also for all other SHAs obtained by compilation from forward navigational XPath queries in the XPathMark benchmark [16]. Even more surprisingly, the same good behaviour was observed for the determinization algorithm for NWA when restricted to NWA*s* with the weak-single entry property. This property failed for the NWA*s* considered by Derbarbieux, but can be established in quadratic time by compiling NWA*s* to SHAs forth and back.

We then observed, unfortunately, that the determinization algorithm for SHAs may still behave badly when applied to some forward navigational XPath queries, which arise in practice but not in the XPathMark benchmark [16]. A typical killer example is the following XPath query:

```
(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 nodes may have any XML type: element, attribute, comment, or text. The nondeterministic SHA has 145 states and an overall size of 348. Its determinization however leads to an automaton with 10.005 states and an overall size of 1.634.122. This size is way too big for many algorithms requiring deterministic automata.

So one natural question is whether and how the size of deterministic automata can be reduced. One idea is to apply schema-based cleaning [26] which keeps only those states and transition rules of the automaton, that are needed to recognize some nested word satisfying the schema. For automata for XPath queries, we can use a schema one^x stating that a single node is selected by any solution, another schema defining the XML data model. When chosing the intersection of both of them as the schema, the schema-based cleaning of the deterministic SHA for `QN7` indeed has only 74 states and 203 transitions. When applying SHA minimization [26] afterwards, the size of the automaton goes down to 27 states and 71 transition rules.

So, the schema seems to play a crucial role. Our implementation of schema-based cleaning, however, runs out of memory for larger automata, say with more than 1000 states. Therefore, we cannot use it to reduce the sizes of the deterministic SHA obtained from `QN7` as announced above. The same holds for our implementation of SHA minimization.

The question of how to produce small deterministic automaton for general queries that are as simple as `QN7` remains thus open. Given that schemas are relevant, one approach could be to determinize the product of the automaton for the query with the schema. At a first glance, this may look questionable, given that the schema-product is usually bigger than the original automaton. So why could determinization become more efficient? But in the case of `QN7`, the determinization of the schema-product yields a deterministic automata with only 92 states and 325 transition rules, and can be computed efficiently. That looks promising. But we also notice that this is slightly bigger than what we announced for schema-based cleaning. The observations made at that example motivate three general questions.

1. Why are schemas so important for automata determinization?
2. Can this be shown by complexity result for the determinization of the schema-product?
3. Is there an efficient way to compute the schema-based cleaning of the determinization of an automaton? Clearly, schema-less determinization needs to be avoided.

Our main result is a novel algorithm for schema-based determinization of NFAs and SHAs, that integrates schema-based cleaning directly into the usual determinization algorithm, rather than applying it a posteriori. This algorithm answers question 3 positively. Its idea is to keep only those subsets of states of the automaton during the determinization, that can be aligned to some state of the schema. In our Theorem 2, we prove that schema-based determinization always produces the same deterministic automaton than schema-free determinization followed by schema-based cleaning. By schema-based determinization we

obtained the schema-based cleaning of the determinization of QN7 in less than three seconds. In contrast, the schema-based cleaning of the determinization does not terminate after a few hours. In the general case, the worst case complexity of schema-based determinization is lower than of determinization followed by schema-based cleaning. We also provide a more precise complexity upper bound in Proposition 12, showing that the maximal time for computing the schema-based determinization of a SHA is roughly the square of the number of its states. For NFAs it is the product of the number of states and the size of the alphabet.

Schema-based determinization also helps to analyze the complexity of the determinization of the schema-product. Let A be a nondeterministic automaton and $det(A)$ its determinization. Let schema S be a deterministic automata and $A \times S$ the schema-product. We first note that $det(A \times S) = det(A) \times S$ since S is deterministic.[1] We second notice that $det(A) \times S$ may be way smaller than $\det(A)$. The reason is that for the many states $Q = \{q_1 \dots q_n\}$ of $det(A)$ there may not exist any state s of S such that $(Q,s) \in det(A) \times S$, because this requires all states q_i can be aligned to s, i.e. that (q_i,s) in $A \times S$ for all $1 \leq i \leq n$. This is why the schema is so relevant for determinization, answering question 1. Third, it is not difficult to see that the schema-based determinization $det_S(A)$ is always smaller than $\det(A) \times S$. Furthermore, $\det(A) \times S$ is equal to $det_S(A) \times S$, so that $det(A \times S) = det_S(A) \times S$. Hence any size bound for the schema-based determinization $det_S(A)$ implies a size bound for the determinization of the schema-product. Furthermore, in our experiments $det_S(A) \times S$ is only by a factor of 2 bigger than $det_S(A)$. So the size of the determinization of the schema-product is closely tied to the size of the schema-based determinization. This is the answer to question 2.

Outline. In Section 2, we recall the definition NFAs and discuss how to use them as schemas and queries on words. In Section 3, we recall schema-based cleaning for NFAs. In Section 4, we contribute our schema-based determinization algorithm in the case of NFAs and show its correctness. In Section 5, we recall the notion of SHAs for defining languages of nested words. In Section 6, we lift schema-based determinization to SHAs. Full proofs are given in the Appendix. It also contains a section on further related work.

2 Finite Automata on Words, Schemas, and Queries

In this section, we discuss hwo to use NFAs for defining schemas and queries on words.

Let $\mathbb{N}$ be the set of natural numbers including 0. The set of words over a finite alphabet Σ is $\Sigma^* = \bigcup_{n \in \mathbb{N}}^{\infty} \Sigma^n$. A word $(a_1, \dots, a_n) \in \Sigma^n$ is written as $a_1 \dots a_n$. We denote by ε the empty word, i.e., the unique element of Σ^0 and by $w_1 \cdot w_2 \in \Sigma^*$ the concatenation of two words $w_1, w_2 \in \Sigma^*$. For example, if $\Sigma = \{a,b\}$ then $aa \cdot bb = aabb = a \cdot a \cdot b \cdot b$.

Definition 1. *A* NFA *is a tuple $A = (\Sigma, \mathscr{Q}, \Delta, I, F)$ such that $\mathscr{Q}$ is a finite set of states, the alphabet Σ is a finite set, $I, F \subseteq \mathscr{Q}$ are subsets of initial and final states, and $\Delta \subseteq \mathscr{Q} \times \Sigma \times \mathscr{Q}$ is the set of transition rules.*

The size of a NFA is $|A| = |\mathscr{Q}| + |\Delta|$. A transition rule $(q,a,q') \in \Delta$ is denoted by $q \xrightarrow{a} q' \in \Delta$. We define transitions $q \xrightarrow{w} q'$ wrt Δ for arbitrary words $w \in \Sigma^*$ by the following inference rules:

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

The language of words recognized by a NFA then is $\mathscr{L}(A) = \{w \in \Sigma^* \mid q \xrightarrow{w} q' \text{ wrt } \Delta,\ q \in I,\ q' \in F\}$.

[1] If $\{(q_1,s_1) \dots (q_n,s_n)\} \in det(A \times S)$ then there exists a tree that can go into all states $q_1 \dots q_n$ with A and into all states $s_1, \dots s_n$ with S. Since S is deterministic, we have $s_1 = \dots s_n$. So there exists a tree going into $\{q_1, \dots, q_n\}$ with $det(A)$ and also into all s_i. So $(\{q_1, \dots, q_n\}, s_i)$ is a state of $det(A) \times S$.

$$\frac{I^A \neq \emptyset}{I^A \in I^{det(A)} \qquad I^A \in \mathscr{Q}^{\det(A)}} \qquad \frac{Q \in \mathscr{Q}^{\det(A)} \quad Q \cap F^A \neq \emptyset}{Q \in F^{\det(A)}}$$

$$\frac{Q \in \mathscr{Q}^{\det(A)} \qquad Q' = \{q' \in \mathscr{Q}^A \mid q \xrightarrow{a} q' \in \Delta^A,\ q \in Q\} \neq \emptyset}{Q \xrightarrow{a} Q' \in \Delta^{det(A)} \qquad Q' \in \mathscr{Q}^{\det(A)}}$$

$$\det(A) = (\Sigma, \mathscr{Q}^{\det(A)}, \Delta^{\det(A)}, I^{\det(A)}, F^{\det(A)})$$

Figure 1: The accessible determinization $\det(A)$ of NFA A.

```
1  fun det(A) =
2    let Store = hashset.new(∅)
3    let Agenda = list.new() and Rules = hashset.new(∅)
4    if init^A ≠ ∅ then Agenda.add(init^A)
5    while Agenda.notEmpty() do
6      let Q = Agenda.pop()
7      let h be an empty hash table with keys from Σ.
8          // the values will be nonempty hash subsets of 2^A
9      for q -a-> q' ∈ Δ^A such that q ∈ Q do
10         if h.get(a) = undef then h.add(a, hashset.new(∅))
11         (h.get(a)).add(q')
12     for (a,Q') in h.tolist() do Rules.add(Q -a-> Q')
13         if not Store.member(Q') then Store.add(Q') Agenda.push(Q')
14   let init^det(A) = {Q | Q ∈ Store, Q ∩ init^A ≠ ∅} and F^det(A) = {Q | Q ∈ Store, Q ∩ F^A ≠ ∅}
15   return (Σ, Store.toSet(), Rules.toSet(), init^det(A), F^det(A))
```

Figure 2: A program computing the accessible determinization of an NFA A from Fig. 1.

A NFA A is called *deterministic* or equivalently a DFA, if it has at most one initial state, and for every pair $(q,a) \in \mathscr{Q} \times \Sigma$ there is at most one state $q' \in \mathscr{Q}^A$ such that $q \xrightarrow{a} q' \in \Delta^A$. Any NFA A can be converted into a DFA that recognizes the same language by the usual subset construction. The accessible determinization $\det(A)$ of $A = (\Sigma, \mathscr{Q}^A, \Delta^A, I^A, F^A)$ is defined by the inference rules in Fig. 1. It works like the usual subset construction, except that only accessible subsets are created. It is well known that $\mathscr{L}(A) = \mathscr{L}(\det(A))$. Since only accessible subsets of states are added, we have $\mathscr{Q}^{\det(A)} \subseteq 2^{\mathscr{Q}^A}$. Therefore, the accessible determinization may even reduce the size of the automaton and often avoid the exponential worst case where $\mathscr{Q}^{\det(A)} = 2^{\mathscr{Q}^A}$.

Proposition 2. *The accessible determinization* $\det(A)$ *of a* NFA A *can be computed in expected amortized time* $O(|\mathscr{Q}^{\det(A)}|\ |\Delta^A| + |A|)$.

Proof. An algorithm computing the fixed point of the inference rules for accessible determinization in Fig. 1 is presented in Fig. 2. It uses dynamic perfect hashing [13] for implementing hash sets, so that set inserting and membership can be done in randomized amortized time $O(1)$. The algorithm has a hash set *Store* to save all discovered states $\mathscr{Q}^{det(A)}$ and a hash set *Rules* to collect all transition rules. Furthermore, it has a stack *Agenda* to process all new states $Q \in \mathscr{Q}^{det(A)}$. For each Q popped from the stack *Agenda*, the algorithm uses a hash table h to compute all pairs (a, Q') such that $Q \xrightarrow{a} Q' \in \Delta^{\det(A)}$ and $Q' \neq \emptyset$. This is done by iterating of Δ^A so in time $O(|\Delta^A|)$. By iterating over the hash table h, all transitions $Q \xrightarrow{a} Q'$ will be added to the set *Rules* and Q' will be added to the stack *Agenda* and to the hash set *Store* if it wasn't there yet. The overall number of elements in the *Agenda* is $|\mathscr{Q}^{det(A)}|$. For each

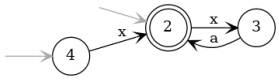


Figure 3: The NFA A_0 for the regular expression $(x+\varepsilon).(x.a)^*$

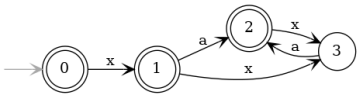


Figure 4: The accessible determinization $\det(A_0)$ up to the renaming of states $[\{2,4\}/0,\{2,3\}/1,\{2\}/2,\{3\}/3]$.

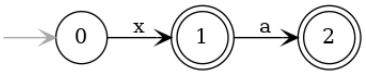


Figure 5: The schema-based cleaning of $\det(A_0)$ with schema $words\text{-}one^x_\Sigma$.

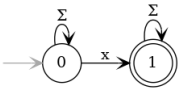
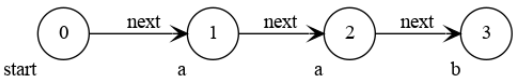


Figure 6: Schema $words\text{-}one^x_\Sigma$ with alphabet $\Sigma \uplus \{x\}$.

Q, the computation of all Q' is in time $O(|\Delta^A|)$. The preprocessing of A requires time $O(|A|)$. Thus, the total time of the algorithm is in $O(|\mathscr{Q}^{\det(A)}|\ |\Delta^A| + |A|)$. □

As a running example, we consider the NFA A_0 for the regular expression $(x+\varepsilon).(x.a)^*$ that is drawn as a labeled digraph in Fig. 3: the nodes of the graph are the states and the labeled edges represent the transitions rules. The initial states are indicated by an ingoing arrow and the final state are doubly circled. The graph of the DFA $\det(A_0)$ obtained by accessible determinization is shown in Fig. 4. It is given up to a renaming of the states that is given in the caption. Note that only 4 out of the $2^3 = 8$ subsets are accessible, so the size increases only by a single state and two transitions rules in this example.

A *regular schema* over Σ is a DFA with the alphabet Σ. We next show how to use automata to define regular queries on words. For this, any word is seen as a labeled digraph. The labeled digraph of the word *aab*, for instance, is drawn to the right. The



set of nodes of the graph is the set of positions of the word $pos(w) = \{0,\ldots,n\}$ where n is the length of w. Position 0 is labeled by *start*, while all other positions are labeled by a single letter in Σ. A monadic query function on words with alphabet Σ is a total function $\mathbf{Q}$ that maps some words $w \in \Sigma^*$ to a subset of position $\mathbf{Q}(w) \subseteq pos(w)$. We say that a position $\pi \in pos(w)$ is selected by $\mathbf{Q}$ if $w \in dom(\mathbf{Q})$ and $\pi \in \mathbf{Q}(w)$.

Let us fix a single variable x. Given a position π of a word $w \in \Sigma^*$ let $w * [\pi/x]$ be the word obtained from w by inserting x after position π. We note that all words of the form $w * [\pi/x]$ contain a single occurrence of x. Such words are also called V-structures where $V = \{x\}$ (see e.g [29]).

The set of all V-structures can be defined by the schema $words\text{-}one^x_\Sigma$ over $\Sigma \uplus \{x\}$ in Fig. 6. It is natural to identify any total monadic query function $\mathbf{Q}$ with the language of V-structures $L_\mathbf{Q} = \{w * [\pi/x] \mid w \in \Sigma^*, \pi \in \mathbf{Q}(w)\}$. This view permits us to define a subclass of total monadic query functions by automata.

A *(monadic) query automaton over* Σ is a NFA A with alphabet $\Sigma \uplus \{x\}$. It defines the unique total monadic query function $\mathbf{Q}$ such that $L_\mathbf{Q} = \mathscr{L}(A) \cap \mathscr{L}(words\text{-}one^x_\Sigma)$. A position π of a word $w \in \Sigma^*$ is thus selected by the query $\mathbf{Q}$ on w if and only if the V-structure $w * [\pi/x]$ is recognized by A, i.e.:

$$\pi \in \mathbf{Q}(w) \Leftrightarrow w * [\pi/x] \in \mathscr{L}(A)$$

A query function is called regular if it can be defined by some NFA. It is well-known from the work of Büchi in the sixties [9] that the same class of regular query functions can be defined equivalently by monadic second-order logic.

$$\frac{q \in I^A \quad s \in I^S}{(q,s) \in I^{A\times S} \qquad (q,s) \in \mathscr{Q}^{A\times S}} \qquad \frac{q \in F^A \quad s \in F^S \quad (q,s) \in \mathscr{Q}^{A\times S}}{(q,s) \in F^{A\times S}}$$

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad (q_1,s_1) \in \mathscr{Q}^{A\times S}}{(q_1,s_1) \xrightarrow{a} (q_2,s_2) \in \Delta^{A\times S} \quad (q_2,s_2) \in \mathscr{Q}^{A\times S}}$$

Figure 7: Accessible product $A \times S = (\Sigma, \mathscr{Q}^{A\times S}, I^{A\times S}, F^{A\times S}, \Delta^{A\times S})$.

We note that only the words satisfying the schema *words-one*$^x_\Sigma$ (the V-structures) are relevant for the query function $\mathbf{Q}$ of a query automaton A. The query automaton A_0 in Fig. 3 for instance, defines the query function that selects the start position of the words ε and a and no other positions elsewhere. This is since the subset of V-structures recognized by A_0 is $x + x.a$. Note that the words ε and xxa do also belong to $\mathscr{L}(A_0)$, but are not V-structures, and thus are irrelevant for the query function $\mathbf{Q}$.

3 Schema-Based Cleaning

Schema-based cleaning was introduced only recently [26] in order to reduce the size of automata on nested words. The idea is to remove all rules and states from an automaton that are not used to recognize any word satisfying the schema. Schema-based cleaning can be based on the accessible states of the product of the automaton with the schema. While this product may be larger than the automaton, the schema-based cleaning will always be smaller.

For illustration, the schema-based cleaning of NFA $\det(A_0)$ in Fig. 4 with respect to schema *words-one*$^x_\Sigma$ is given in Fig. 5. The only words recognized by both $\det(A_0)$ and *words-one*$^x_\Sigma$ are x and xa. For recognizing these two words, the automaton $\det(A_0)$ does not need states 2 and 3, so they can be removed with all their transitions rules. Thereby, the word xxa violating the schema is no more recognized after schema-based cleaning, while it was recognized by $\det(A_0)$. Furthermore, note that the state 0 needs no more to be final after schema-based cleaning. Therefore the word ε, which is recognized by the automaton but not by the schema, is no more recognized after schema-based cleaning. So schema-based cleaning may change the language of the automaton but only outside of the schema.

Interestingly, the NFA A_0 in Fig. 3 is schema-clean for schema *words-one*$^x_\Sigma$ too, even though it is not perfect, in that recognizes the words ε and xxa which are rejected by the schema. The reason is that for recognizing the words x and xa, which both satisfy the schema, all 3 states and all 4 transition rules of A_0 are needed. In contrast, we already noticed that the accessible determinization $\det(A_0)$ in Fig. 4 is not schema-clean for schema *words-one*$^x_\Sigma$. This illustrates that accessible determinization does not always preserve schema-cleanliness. In other words, schema-based cleaning may have a stronger cleaning effect after determinization than before.

The schema-based cleaning of an automaton can be defined based on the accessible product of the automaton with the schema. The accessible product $A \times S$ of two NFAs A and S with alphabet Σ is defined in Fig. 7. This is the usual product, except that only accessible states are admitted. Clearly, $\mathscr{L}(A \times S) = \mathscr{L}(A) \cap \mathscr{L}(S)$. Let $\Pi_A(A \times S)$ be obtained from the accessible product by projecting away the second component, as formally defined in Fig. 19. The schema-based cleaning of A with respect to schema S is this projection.

Definition 3. $scl_S(A) = \Pi_A(A \times S)$.

The fact that $A \times S$ is restricted to accessible states matches our intuition that all states of $scl_S(A)$ can be used to read some word in $\mathscr{L}(A)$ that satisfies schema S. This can be proven formally under the

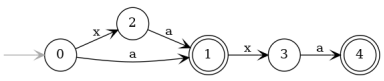


Figure 8: A DFA that is schema-clean but not perfect for $words\text{-}one^x_\Sigma$.

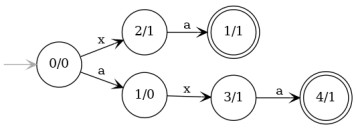


Figure 9: The accessible product with $words\text{-}one^x_\Sigma$ is schema-clean and perfect for $words\text{-}one^x_\Sigma$.

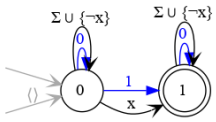


Figure 10: The dSHA one^x_Σ with alphabet $\Sigma \uplus \{x, \neg x\}$.

condition that all states of $A \times S$ are also co-accessible. Clearly, $scl_S(A)$ is obtained from A by removing states, initial states, final states, and transitions rules. So it is smaller or equal in size $|scl_S(A)| \leq |A|$ and language $\mathscr{L}(scl_S(A)) \subseteq \mathscr{L}(A)$. Still, schema-based cleaning preserves the part of the language within the schema:

Proposition 4. $\mathscr{L}(A) \cap \mathscr{L}(S) = \mathscr{L}(scl_S(A)) \cap \mathscr{L}(S)$.

Schema-clean deterministic automata may still not be perfect, in that schema-clean DFAs may recognize some words outside the schema. This happens if some state of a DFA is reached both, by a word satisfying the schema and another word that does not satisfy the schema. An example for a DFA that is schema-clean but not perfect for $words\text{-}one^x_\Sigma$ is given in Fig. 8. It is not perfect since it accepts the non V-structure $xaxa$. The problem is that state 1 can be reached by the words a and xa, so one cannot infer from being in state 1 whether some x was read or not. If one wants to avoid this, one can use the accessible product of the DFA with the schema instead. In the example, this yields the DFA in Fig. 9 that is schema-clean and perfect for $words\text{-}one^x_\Sigma$.

Proposition 5. *For any* DFA*s A and S with alphabet Σ the accessible product $A \times S$ and the schema-based cleaning $scl_S(A)$ can be computed in expected amortized time $O(|\mathscr{Q}^{A\times S}||\Sigma| + |A| + |S|)$.*

Proof. It is sufficient to show that the accessible product $A \times S$ can be computed in this time. An algorithm to compute the fixed points of the inference rules for the accessible product $A \times S$ in Fig. 7 can be organized such that only accessible states are considered (similarly to semi-naive datalog evaluation). This algorithm is presented in Fig. 11. It dynamically generates the set of rules *Rules* by using perfect dynamic hashing [13]. Testing set membership is in time $O(1)$ and the addition of elements to the set is in expected amortized time $O(1)$. The algorithm uses a stack, *Agenda*, to memoize all new pairs $(q_1, s_1) \in \mathscr{Q}^{A\times S}$ that need to be processed, and a hash set *Store* that saves all processed states $\mathscr{Q}^{A\times S}$. We aim not to push the same pair more than once in the *Agenda*. For this, membership to the *Store* is checked before an element is pushed to the *Agenda*. For each pair popped from the stack *Agenda*, the algorithm does the following: for each letter $a \in \Sigma$ it computes the sets $Q = \{q_2 \mid q_1 \xrightarrow{a} q_2 \in \Delta^A\}$ and $R = \{s_2 \mid s_1 \xrightarrow{a} s_2 \in \Delta^S\}$ and then adds the subset of states of $Q \times R$ that were not stored in the hash set *Store* to the agenda. Since A and S are deterministic, there is at most one such pair, so the time for treating one pair on the agenda is in expected amortized time $O(|\Sigma|)$. The overall number of elements in the agenda will be $|\mathscr{Q}^{A\times S}|$. Note that Q and R can be computed in $O(1)$ after preprocessing A and S in time $O(|A| + |S|)$. Therefore, we will have a total time of the algorithm in $O(|\mathscr{Q}^{A\times S}||\Sigma| + |A| + |S|)$. □

4 Schema-Based Determinization

Schema-based cleaning after determinization becomes impossible in practice if the automaton obtained by determinization is too big. We therefore show next how to integrate schema-based cleaning into automata determinization directly.

```
1  fun A × S =
2    let Store = hashset.new(∅)
3    let Agenda = list.new() and Rules = hashset.new(∅)
4    if init^A = {q0} and init^S = {s0} then Agenda.add((q0,s0))
5    while Agenda.notEmpty() do
6      let (q1,s1) = Agenda.pop()
7      for a ∈ Σ do
8        let Q = {q2 | q1 -a-> q2 ∈ Δ^A}  R = {s2 | s1 -a-> s2 ∈ Δ^S}
9        for q2 ∈ Q and s2 ∈ R do
10         Rules.add((q1,s1) -a-> (q2,s2))
11         if not Store.member((q2,s2))
12           then Store.add((q2,s2))  Agenda.push((q2,s2))
13   let init^{A×S} = {(q0,s0) | (q0,s0) ∈ Store} and F^{A×S} = {(q,s) | (q,s) ∈ Store, q ∈ F^A, s ∈ F^S}
14   return (Σ, Store.toSet(), Rules.toSet(), init^{A×S}, F^{A×S})
```

Figure 11: An algorithm computing the accessible product of DFAs A and S.

$$\frac{Q \in I^{det(A)} \quad I^S = \{s\}}{Q \in I^{det_S(A)} \quad Q \sim s} \qquad \frac{Q \sim s}{Q \in \mathscr{Q}^{det_S(A)}} \qquad \frac{Q \in F^{\det(A)} \qquad s \in F^S \qquad Q \sim s}{Q \in F^{det_S(A)}}$$

$$\frac{Q \xrightarrow{a} Q' \in \Delta^{det(A)} \qquad Q \sim s \qquad s \xrightarrow{a} s' \in \Delta^S}{Q \xrightarrow{a} Q' \in \Delta^{det_S(A)} \qquad Q' \sim s'}$$

Figure 12: Schema-based determ. $det_S(A) = (\Sigma, \mathscr{Q}^{det_S(A)}, \Delta^{det_S(A)}, I^{det_S(A)}, F^{det_S(A)})$.

The schema-based determinization of A with respect to schema S extends on accessible determinization $\det(A)$. The idea is to run the schema S in parallel with $\det(A)$, in order to keep only those state $Q \in \mathscr{Q}^{\det(A)}$ that can be aligned to some state $s \in \mathscr{Q}^S$. In this case we write $Q \sim s$.

The schema-determinization $det_S(A)$ is defined in Fig. 12. The automaton $det_S(A)$ permits to go from any subset $Q \in \mathscr{Q}^{\det(A)}$ and letter $a \in \Sigma$ to the set of states $Q' = a^{\Delta^{\det(A)}}(Q)$, under the condition that there exists schema states $s, s' \in \mathscr{Q}^S$ such that $Q \sim s$ and $s \xrightarrow{a} s'$. In this case $Q' \sim s'$ is inferred.

Theorem 1 (Correctness)**.** *$det_S(A) = scl_S(\det(A))$ for any* NFA *A and* DFA *S with the same alphabet.*

The theorem states that schema-based determinization yields the same result as accessible determinization followed by schema-based cleaning.

For preparing the correctness proof we first collapse the two systems of inference rules for accessible products and projection into a single rule system. This yields the rule systems for schema-based cleaning in Fig. 20.

The rules there define the automaton $\widehat{scl}_S(A)$, that we annotate with a hat, in order to distinguish it from the previous automaton $scl_S(A)$. The rules also infer judgements $(q,s) \in \mathscr{Q}^{A \widehat{\times} S}$ that we distinguish by a hat from the previous judgments $(q,s) \in \mathscr{Q}^{A \times S}$ of the accessible product. The next proposition shows that the system of collapsed inference rules indeed redefines the schema-based cleaning.

Proposition 6. *For any two* NFA*s A and S with the same alphabet:*

$$scl_S(A) = \widehat{scl}_S(A) \quad \text{and} \quad \mathscr{Q}^{A \times S} = \mathscr{Q}^{A \widehat{\times} S}$$

Proof of Correctness Theorem 1. Instantiating the system of collapsed rules for schema-based cleaning from Fig. 20 with $\det(A)$ for A yields the rule system in Fig. 21. We can identify the instantiated collapsed

```
1  fun detS(A,S) =
2    let Store = hashset.new(∅)
3    let Agenda = list.new() and Rules = hashset.new(∅)
4    if init^A ≠ ∅ and init^S = {s0} then Agenda.add(init^A ~ s0)
5    while Agenda.notEmpty() do
6      let (Q1 ~ s1) = Agenda.pop()
7      for a ∈ Σ do
8        let P = {Q2 | Q1 -a-> Q2 ∈ Δ^det(A)} and R = {s2 | s1 -a-> s2 ∈ Δ^S}
9        for Q2 ∈ P and s2 ∈ R do Rules.add(Q1 -a-> Q2)
10           if not Store.member(Q2 ~ s2)
11            then Store.add(Q2 ~ s2) Agenda.push(Q2 ~ s2)
12   let init^{det_S(A)} = {Q | Q ~ s ∈ Store, Q ∩ init^A ≠ ∅} and F^{det_S(A)} = {Q | Q ~ s ∈ Store, Q ∩ F^A ≠ ∅}
13   return (Σ, Store.toSet(), Rules.toSet(), init^{det_S(A)}, F^{det_S(A)})
```

Figure 13: An algorithm for schema-based determinization $det_S(A)$ of an NFA A and a DFA schema S.

system for $\widehat{scl}_S(det(A))$ with that for $det_S(A)$ in Fig. 12, by identifying the judgements $(Q,s) \in \mathcal{Q}^{det(A)\widehat{\times}S}$ with judgments $Q \sim s$. After renaming the predicates, the inference rules for the corresponding judgments are the same. Hence $\widehat{scl}_S(det(A)) = det_S(A)$, so that Proposition 6 implies $scl_S(det(A)) = det_S(A)$. □

Proposition 7. *The schema-based determinization $det_S(A)$ for a* NFA *A and a* DFA *S over Σ can be computed in expected amortized time $O(|\mathcal{Q}^{\det(A)\times S}||\Sigma| + |\mathcal{Q}^{det_S(A)}||\Delta^A| + |A| + |S|)$.*

Proof. An algorithm computing the fixed points of the inference rules of schema-based determinization from Fig. 12 is given in Fig. 13. It refines the algorithm computing the accessible product with on-the-fly determinization and projection.

On the stack *Agenda*, the algorithm stores alignments $Q \sim s$ such that $(Q,s) \in \mathcal{Q}^{\det(A)\times S}$ that were not considered before. Transition rules of $det_S(A)$ are collected in hash set *Rules*, using the dynamic perfect hashing aforementioned. The alignments $Q_1 \sim s_1$ popped from the agenda are processed as follows: For any letter $a \in \Sigma$, the sets $R = \{Q_2 \mid Q_1 \xrightarrow{a} Q_2 \in \Delta^{\det(A)}\}$ and $P = \{s_2 \mid s_1 \xrightarrow{a} s_2 \in \Delta^S\}$ are computed. One then pushes all new pairs $Q_2 \sim s_2$ with $Q_2 \in P$ and $s_2 \in R$ into the agenda, and adds $Q_1 \xrightarrow{a} Q_2$ to the set *Rules*. Since S and $\det(A)$ are deterministic there is at most one pair $(Q,s) \in P \times R$ for Q_1 and s_1. So the time for treating one pair on the agenda is in $O(|\Sigma|)$ plus the time for building the needed transition rules of $\det(A)$ from Δ^A on the fly. The time for the on the fly computation of transition rules of $\det(A)$ is in time $O(|\mathcal{Q}^{det_S(A)}||\Delta^A|)$. The overall number of pairs on the agenda is at most $|\mathcal{Q}^{\det(A)\times S}|$ so the main while loop of the algorithm requires time in $O(|\mathcal{Q}^{\det(A)\times S}||\Sigma|)$ apart from on the fly determinization. This will give us an overall complexity for the algorithm in $O(|\mathcal{Q}^{\det(A)\times S}||\Sigma| + |\mathcal{Q}^{det_S(A)}||\Delta^A| + |A| + |S|)$, with consideration of the preprocessing time of A and S. □ □

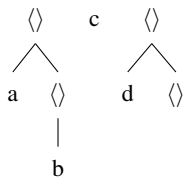
By Proposition 2, computing $\det(A)$ requires time $O(|\mathcal{Q}^{det(A)}|\ |\Delta^A| + |A|)$. Therefore, with Proposition 5, the accessible product $\det(A) \times S$ can be computed from A and S in time $O(|\mathcal{Q}^{\det(A)\times S}||\Sigma| + |\mathcal{Q}^{det(A)}|\ |\Delta^A| + |A| + |S|)$. Since $\mathcal{Q}^{det_S(A)} \subseteq \mathcal{Q}^{\det(A)}$ the proposition shows that schema-based determinization is at most as efficient in the worst case as accessible determinization followed by schema-based cleaning. If $|\mathcal{Q}^{\det(A)\times S}||\Sigma| < |\mathcal{Q}^{\det(A)}||\Delta^A|$ then it is more efficient, since schema-based determinization avoids the computation of $\det(A)$ all over. Instead, it only computes the accessible product $\det(A) \times S$, which may be considerably smaller, since exponentially many states of $\det(A)$ may not be aligned to any state of S. Sometimes, however, the accessible product may be bigger. In this case, schema-based determinization may be more costly than pure accessible determinization, not followed by schema-based cleaning.

5 Stepwise Hedge Automata for Nested Words

We next recall SHAs [26] for defining languages of nested words, regular schemas and queries. Nested words generalize on words by adding parenthesis that must be well-nested. While containing words natively, they also generalize on unranked trees, and hedges. We restrict ourselves to nested words with a single pair of opening and closing parenthesis $\langle$ and $\rangle$. Nested words over a finite alphabet Σ of internal letters have the following abstract syntax.

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

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. Seen as a graph, the inner nodes are labeled by the tree constructor $\langle\rangle$ and the leafs by symbols in Σ 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.



A nested word of type *tree* has the form $\langle h \rangle$. Note that dangling parentheses are ruled out and that labeled parentheses can be simulated by using internal letters. *XML* documents are labeled unranked trees, for instance: $\langle a\ name = \text{“}uff\text{”} \rangle \langle b \rangle isgaga \langle d/ \rangle \langle /b \rangle \langle c/ \rangle \langle /a \rangle$. Labeled unranked trees satisfying the *XML* data model can be represented as nested words over an alphabet that contains the *XML* node-types $(elem, attr, text, \ldots)$, the *XML* names of the document $(a, \ldots, d, name)$, and the characters of the data values, say UTF8. For the above example, we get the nested word $\langle elem \cdot a \cdot \langle attr \cdot name \cdot u \cdot f \cdot f \rangle \langle elem \cdot b \cdot \langle text \cdot i \cdot s \cdot g \cdot a \cdot g \cdot a \rangle \langle elem \cdot d \rangle \rangle \langle elem \cdot c \rangle \rangle$

Definition 8. *A* SHA *is a tuple* $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ *where* $\Delta = (\Delta', @^\Delta, \langle\rangle^\Delta)$ *such that* $(\Sigma, \mathcal{Q}, \Delta', I, F)$ *is a* NFA, $\langle\rangle^\Delta \subseteq \mathcal{Q}$ *is a set of tree initial states and* $@^\Delta \subseteq \mathcal{Q}^3$ *a set of apply rules.*

SHAs can be drawn as graphs while extending on the graphs of NFAs. A tree initial state $q \in \langle\rangle^\Delta$ is drawn as a node $\xrightarrow{\langle\rangle} (q)$ with an incoming tree arrow. An applyrule $(q_1, q, q_2) \in @^\Delta$ is drawn as a blue edge $(q_1) \xrightarrow{q} (q_2)$ that is labeled by a state $q \in \mathcal{Q}$ rather than a letter $a \in \Sigma$. It states that a nested word in state q_1 can be extended by a tree in state q and become a nested word in state q_2.

For instance, the SHA one^x_Σ is drawn graphically in Fig. 10. It accepts all nested words over $\Sigma \uplus \{x, \neg x\}$ that contain exactly one occurrence of letter x. Compared to the NFA $words\text{-}one^x_{\Sigma \uplus \{\neg x\}}$ from Fig. 6, the SHA one^x_Σ contains three additional apply rules $(0,0,0)$, $(0,1,1)$, $(1,0,1) \in @^{\Delta^{one^x_\Sigma}}$ for reading the states assigned to subtrees. The state 0 is chosen as the single tree initial state.

Transitions for NFAs on words can be lifted to transitions for SHAs of the form $q \xrightarrow{w} q'$ wrt Δ where $w \in \mathcal{N}_\Sigma$ and $q, q' \in \mathcal{Q}$. For this, we add the following inference rule to the previous inference rules for NFAs:

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

The rule says that a tree $\langle w \rangle$ can transit from a state q_1 to a state q_2 if there is an apply rule $(q_1, q, q_2) \in @^\Delta$ so that w can transit from some tree initial state $q' \in \langle\rangle^\Delta$ to q. Otherwise, the language $\mathcal{L}(A)$ of nested words accepted by a SHA A is defined as in the case of NFAs.

Definition 9. *A* SHA $(\Sigma, \mathcal{Q}, \Delta, I, F)$ *is* deterministic *or equivalently a d*SHA*, if it satisfies the following conditions:*

- *I and $\langle\rangle^\Delta$ both contain at most one element,*
- *a^Δ is a partial function from $\mathcal{Q}$ to $\mathcal{Q}$ for all $a \in \Sigma$, and*

$$\frac{\langle\!\rangle^{\Delta^A} \neq \emptyset}{\langle\!\rangle^{\Delta^A} \in \mathcal{Q}^{\det(A)}} \qquad \frac{\begin{array}{c} Q_1 \in \mathcal{Q}^{\det(A)} \qquad Q_2 \in \mathcal{Q}^{\det(A)} \\ Q' = \{q' \in \mathcal{Q}^A \mid q_1 @ q_2 \to q' \in \Delta^A,\ q_1 \in Q_1, q_2 \in Q_2\} \neq \emptyset \end{array}}{Q_1 @ Q_2 \to Q' \in \Delta^{det(A)} \qquad Q' \in \mathcal{Q}^{\det(A)}}$$

Figure 14: Accessible determinization $\det(A)$ lifted from NFAs to SHAs.

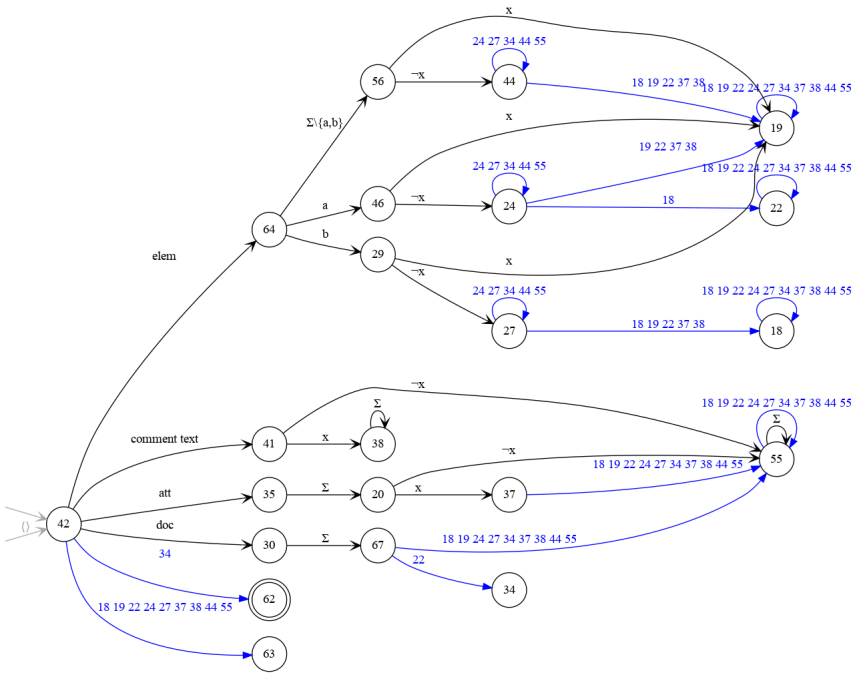


Figure 15: A one^x_Σ-cleaned minimal dSHA for the XPATH query `QN7`.

- $@^\Delta$ *is a partial function from* $\mathcal{Q} \times \mathcal{Q}$ *to* $\mathcal{Q}$.

Note that if A is a dSHA and $\Delta = (\Delta', @^\Delta, \langle\!\rangle^\Delta)$ then $A' = (\Sigma, \mathcal{Q}, \Delta', I, F)$ is a DFA. Conversely any DFA A' defines a dSHA with $@^\Delta = \emptyset$ and $I = \emptyset$. For instance, the SHA one^x_Σ in Fig. 10 contains the DFA $words\text{-}one^x_{\Sigma \uplus \{\neg x\}}$ from Fig. 6 with Σ instantiated by $\Sigma \uplus \{x\}$.

A schema for nested words over Σ is a dSHA over Σ. Note that schemas for nested words generalize over schemas of words, since dSHAs generalize on DFAs. The rules for the accessible determinization $\det(A)$ of a SHA A in Fig. 14 extend on those for NFAs in Fig. 1. As for words, $\det(A)$ is always determinstic, recognizes the same language as A, and contains only accessible states. The complexity of accessible determinization in case of SHA go similarly to DFA, however, the apply rules will introduce quadratic factor in the number of states.

Proposition 10. *The accessible determinization of a* SHA *can be computed in expected amortized time* $O(|\mathcal{Q}^{\det(A)}|^2\ |\Delta^A| + |A|)$.

The notions of monadic query functions $\mathbf{Q}$ can be lifted from words to nested words, so that it selects nodes of the graph of a nested word. For this, we have to fix one of manner possible manners to define identifiers for these nodes. The set of nodes of a nested word w is denoted by $nod(w) \subseteq \mathbb{N}$.

For indicating the selection of node $\pi \in nod(w)$, we insert the variable x into the sequence of letters following the opening parenthesis of π. If we don't want to select π, we insert the letter $\neg x$ instead. For any nested word w with alphabet Σ, the nested word $w[\pi/x]$ obtained by insertion of x or $\neg x$ at a node $\pi \in nod(w)$ has alphabet $\Sigma \uplus \{x, \neg x\}$. As before, we define $L_{\mathbf{Q}} = \{w * [\pi/x] \mid w \in \mathcal{N}_\Sigma, \pi \in \mathbf{Q}(w)\}$.

$$\frac{q \in \langle\rangle^{\Delta^A} \quad s \in \langle\rangle^{\Delta^S}}{(q,s) \in \langle\rangle^{\Delta^{A\times S}} \quad (q,s) \in \mathscr{Q}^{A\times S}} \qquad \frac{\begin{array}{cc}(q_1,s_1) \in \mathscr{Q}^{A\times S} & q_1@q \to q_2 \in \Delta^A \\ (q,s) \in \mathscr{Q}^{A\times S} & s_1@s \to s_2 \in \Delta^S\end{array}}{(q_1,s_1)@(q,s) \to (q_2,s_2) \in \Delta^{A\times S} \quad (q_2,s_2) \in \mathscr{Q}^{A\times S}}$$

Figure 16: Lifting accessible products to SHAs.

$$\frac{\langle\rangle^{\Delta^S} = \{s\}}{\langle\rangle^{\Delta^A} \in \langle\rangle^{\Delta^{det_S(A)}} \quad \langle\rangle^{\Delta^A} \sim s} \qquad \frac{\begin{array}{ccc}s_1@s_2 \to s' \in \Delta^S & Q_1 \sim s_1 & Q_2 \sim s_2 \\ Q_1@Q_2 \to Q' \in \Delta^{det(A)} & & \end{array}}{Q_1@Q_2 \to Q' \in \Delta^{det_S(A)} \qquad Q' \sim s'}$$

Figure 17: Extension of schema-based determinization to SHAs.

The notion of a query automata can now be lifted from words to nested words straightforwardly: a query automaton for nested words over Σ is a SHA A with alphabet $\Sigma \cup \{x, \neg x\}$. It defines the unique total query $\mathbf{Q}$ such that $L_{\mathbf{Q}} = \mathscr{L}(A) \cap \mathscr{L}(one^x_\Sigma)$. A deterministic query automaton for the XPATH QN7 on *XML* documents is given in Fig. 15.

6 Schema-Based Determinization for SHAs

We can lift all previous algorithms from NFAs to SHAs while extending the system of inference rules. The additional rules concern tree initial states, that work in analogy to initial states, and also apply rules that works similarly as internal rules. The new inference rules for accessible products $A \times S$ are given in Fig. 16 and for projection $\Pi_A(A \times S)$ in Fig. 23. As before we define $scl_S(A) = \Pi_A(A \times S)$. The rules for schema-based determinization $det_S(A)$ are extended in Fig. 17. The next complexity upper bound, however, now become quadratic with fixed alphabet:

Proposition 11. *If A and S are dSHAs then the accessible product $A \times S$ and the schema-based cleaning $scl_S(A)$ can be computed in expected amortized time $O(|\mathscr{Q}^{A\times S}|^2 + |\mathscr{Q}^{A\times S}||\Sigma| + |A| + |S|)$.*

Theorem 2 (Correctness)**.** *$det_S(A) = scl_S(\det(A))$. for any SHA A and dSHA S with the same alphabet.*

The proof extends on that for NFAs (Theorem 1) in a direct manner.

Proposition 12. *The schema-based determinization $det_S(A)$ of a SHA A with respect to a dSHA S can be computed in expected amortized time $O(|\mathscr{Q}^{\det(A)\times S}|^2 + |\mathscr{Q}^{\det(A)\times S}|\ |\Sigma| + |\mathscr{Q}^{det_S(A)}|^2\ |\Delta^A| + |A| + |S|)$.*

This proposition follows the result in Proposition 7 with an additional quadratic factor in the size of states of the product $det(A) \times S$ and the states of the schema-based determinized automaton. This is always due to the apply rules of type $\mathscr{Q}^3$.

By Propositions 10 and 11, computing $scl_S(det(A))$ by schema-based cleaning after accessible determinization needs time in $O(|\mathscr{Q}^{\det(A)\times S}|^2 + |\mathscr{Q}^{\det(A)\times S}|\ |\Sigma| + |\mathscr{Q}^{\det(A)}|^2\ |\Delta^A| + |A| + |S|)$. This complexity bound is similar to that of schema-based determinization from Proposition 12. Since $\mathscr{Q}^{det_S(A)} \subseteq \mathscr{Q}^{\det(A)}$, Proposition 12 shows that the worst case time complexity of schema-based determinization is never worse than for schema-based cleaning after determinization.

7 Experiments

For studying the relevance of schemas in automata determinization experimentally we compare determinization, schema-based determinization and the determinization of the schema-product for a scalable collection of SHAs.

	A	det(A)	$det(A \times S)$	$det_S(A)$	scl_S(det(A))	mini(det($A \times S$))	mini($det_S(A)$)
Q1.1	124 (51)	582 (52)	441 (74)	211 (40)	211 (40)	153 (41)	67 (19)
Q1.2	150 (61)	1532 (106)	1240 (134)	627 (80)	627 (80)	155 (41)	69 (19)
Q1.3	176 (71)	4118 (230)	3529 (270)	1859 (172)	1859 (172)	157 (41)	71 (19)
Q1.4	202 (81)	10950 (514)	9824 (582)	5299 (384)		159 (41)	73 (19)
Q2.1	161 (65)	1348 (99)	526 (86)	278 (51)	278 (51)	155 (41)	69 (19)
Q2.2	194 (77)	3579 (211)	1468 (158)	822 (103)	822 (103)	157 (41)	71 (19)
Q2.3	227 (89)	9488 (467)	4148 (322)	2414 (223)		159 (41)	73 (19)
Q2.4	260 (101)	24691 (1047)	11470 (698)	6814 (499)		161 (41)	75 (19)
Q3.1	198 (79)	3246 (202)	611 (98)	345 (62)	345 (62)	157 (41)	71 (19)
Q3.2	238 (93)	8598 (444)	1696 (182)	1017 (126)		159 (41)	73 (19)
Q3.3	278 (107)	22422 (992)	4767 (374)	2969 (274)		161 (41)	75 (19)
Q3.4	318 (121)	57196 (2220)	13116 (814)	8329 (614)		163 (41)	77 (19)

Figure 18: Automata statistics: size(#states)

In order to obtain scalable SHAs whose determinizations are too big to be schema-cleaned or minimized, we start from the following XPATH queries:

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

where $n \in \{1,\ldots,3\}$ and $m \in \{1,\ldots,4\}$. Query `Qn.m` selects all elements of an *XML* document, that are named by either of `a0`, ..., `an` and have some descendant element named by either of `b1`, ..., `bm`. We compile those XPATH queries to SHAs based on the compiler from [26]. We note that these SHAs may also have typed else rules. As schema S we chose the product of one^x with a dSHA for the XML data model (see Fig. 28 of the appendix).

All the experiments are conducted on a Dell laptop with the following specs: Intel® Core™ i7-10875H CPU @ 2.30 GHz,16 cores, and 32 GB of RAM. The statistics on the sizes and number of states of the various automata computed for these queries are presented in Table 18.

The SHA A in the first column is the automaton obtained from query Qn.m by the compiler from [26]. The dSHA $det(A)$ in the second column is obtained from A by accessible determinization. We notice that all SHAs Qn.m could be determinized within our timeout of 1000 seconds. The column $scl_S(\text{det}(A))$ contains the schema-based cleaning of det(A) if it could be computed within the timeout of 1000 seconds. Otherwise the field is left blank, which is the case for 6 out of the 12 queries where the size of det(A) is too big.

In contrast, we can compute the determinization of the schema-product $det(A \times S)$ and the schema-based determinization $det_S(A)$ in all cases in less than 100 seconds. For instance, the highest computation time is for Q3.4 where $det_S(A_{Q3.4})$ took around 31 seconds, and $det(A_{Q3.4} \times S)$ needed 74 seconds.

This shows that the schema is indeed essential for determinization of these automata. Furthermore, the automata obtained by schema-based determinization are always smaller than those obtained by determinizing the schema product.

We notice that the schema-based determinization $det_S(A)$ is always equal to schema-based cleaning of the accessible determinization $scl_S(\det(A))$, if both could be computed within the timeout. This confirms Theorem 2 on the correctness of schema-based determinization. Note that this theorem is not directly applicable since we are using a slightly enriched SHA model in our experiments.

Finally, we minimized all $det(A \times S)$ and $det_S(A)$. It turns out, that all $mini(det(A \times S))$ have 41 states, while all $mini(det_S(A))$ have 19 states (see Fig. 29 and Fig. 30 of the appendix for Q3.4). Only the number of transition rules grows with n and m. We notice that they are neither equal nor equivalent, since they recognize different languages outside schema S. But in any case, schema-based determinization produces smaller dSHAs even after minimization.

Conclusion and Future Work. We presented an algorithm for schema-based determinization for NFAs and SHAs and proved it to produce the same results as determinization followed by schema-based cleaning but with lower worst case complexity. This complexity result also gave us a bound on the size of the standard determinization of the query-schema product. We started with an example of an SHA for a regular XPATH query to lay out the risen problems, and for which schema-based determinization produces a dSHA of size < 300, while schema-free determinization produces an dSHA of size > 1.5 million and provided a scalable experiment denoting the performance of each approach and the size of their resulting automata. In a subsequent paper [1], we successfully applied our schema-based determinization algorithm on a practical benchmark of forward navigational XPATH queries [22], that Lick and Schmitz extracted from practical XSLT and XQuery programs. The results of our benchmark were highly promising and raises hope of providing the needed deterministic tools for many algorithms such as query answering and enumeration. An open line would be to improve the complexity and find an algorithm relying only on the transition set and therefore, possibly get rid of the quadratic factor in states and alphabet.

References

[1] Antonio Al Serhali & Joachim Niehren (2022): *A Benchmark Collection of Deterministic Automata for XPath Queries*. Available at `https://hal.inria.fr/hal-03527888`. Technical report.

[2] Rajeev Alur (2007): *Marrying Words and Trees*. In: *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ACM-Press, pp. 233–242. Available at `http://dx.doi.org/10.1145/1265530.1265564`.

[3] Rajeev Alur & P. Madhusudan (2004): *Visibly pushdown languages*. In: *36th ACM Symposium on Theory of Computing*, ACM-Press, pp. 202–211. Available at `http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=1007390`.

[4] Rajeev Alur & P. Madhusudan (2009): *Adding nesting structure to words*. *Journal of the ACM* 56(3), pp. 1–43. Available at `http://doi.acm.org/10.1145/1516512.1516518`.

[5] Marcelo Arenas & Jorge Pérez (2011): *Querying semantic web data with SPARQL*. In: *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 305–316.

[6] Mikolaj Bojanczyk & Igor Walukiewicz (2008): *Forest algebras*. In Jörg Flum, Erich Grädel & Thomas Wilke, editors: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, *Texts in Logic and Games* 2, Amsterdam University Press, pp. 107–132.

[7] Iovka Boneva, Joachim Niehren & Momar Sakho (2020): *Nested Regular Expressions Can Be Compiled to Small Deterministic Nested Word Automata*. In Henning Fernau, editor: *Computer Science - Theory and Applications - 15th International Computer Science Symposium in Russia, CSR 2020, Yekaterinburg, Russia, June 29 - July 3, 2020, Proceedings*, *Lecture Notes in Computer Science* 12159, Springer, pp. 169–183, doi:10.1007/978-3-030-50026-9_12. Available at `https://doi.org/10.1007/978-3-030-50026-9_12`.

[8] Burchard von Braunmühl & Rutger Verbeek (1985): *Input Driven Languages are Recognized in log n Space*. In Marek Karplnski & Jan van Leeuwen, editors: *Topics in the Theory of Computation*, *North-Holland Mathematics Studies* 102, North-Holland, pp. 1 – 19, doi:https://doi.org/10.1016/S0304-0208(08)73072-X. Available at `http://www.sciencedirect.com/science/article/pii/S030402080873072X`.

[9] J. R. Büchi (1960): *On a Decision Method in a Restricted Second Order Arithmetic*. In Press, editor: *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pp. 1–11.

[10] Julien Carme, Joachim Niehren & Marc Tommasi (2004): *Querying Unranked Trees with Stepwise Tree Automata*. In: *19th International Conference on Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 3091, Springer Verlag, pp. 105–118. Available at `http://www.ps.uni-sb.de/Papers/abstracts/stepwise.html`.

[11] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison & Marc Tommasi (2007): *Tree Automata Techniques and Applications*. Available online since 1997: `http://tata.gforge.inria.fr`.

[12] Denis Debarbieux, Olivier Gauwin, Joachim Niehren, Tom Sebastian & Mohamed Zergaoui (2015): *Early nested word automata for XPath query answering on XML streams*. *Theor. Comput. Sci.* 578, pp. 100–125, doi:10.1016/j.tcs.2015.01.017. Available at `http://dx.doi.org/10.1016/j.tcs.2015.01.017`.

[13] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert & Robert Endre Tarjan (1994): *Dynamic Perfect Hashing: Upper and Lower Bounds*. *SIAM J. Comput.* 23(4), pp. 738–761, doi:10.1137/S0097539791194094. Available at `https://doi.org/10.1137/S0097539791194094`.

[14] Ronald Fagin, Benny Kimelfeld, Frederick Reiss & Stijn Vansummeren (2015): *Document Spanners: A Formal Approach to Information Extraction*. *J. ACM* 62(2), pp. 12:1–12:51, doi:10.1145/2699442. Available at `https://doi.org/10.1145/2699442`.

[15] Michael J. Fischer & Richard E. Ladner (1979): *Propositional Dynamic Logic of Regular Programs*. *J. Comput. Syst. Sci.* 18(2), pp. 194–211, doi:10.1016/0022-0000(79)90046-1. Available at `https://doi.org/10.1016/0022-0000(79)90046-1`.

[16] Massimo Franceschet: *XPathMark Performance Test*. `https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html`. Accessed: 2020-10-25.

[17] Olivier Gauwin, Joachim Niehren & Sophie Tison (2009): *Earliest Query Answering for Deterministic Nested Word Automata*. In: *17th International Symposium on Fundamentals of Computer Theory*, *Lecture Notes in Computer Science* 5699, Springer Verlag, pp. 121–132. Available at `http://hal.inria.fr/inria-00390236/en`.

[18] Georg Gottlob & Christoph Koch (2002): *Monadic Queries over Tree-Structured Data*. In: *17th Annual IEEE Symposium on Logic in Computer Science*, Copenhagen, pp. 189–202.

[19] Georg Gottlob, Christoph Koch & Reinhard Pichler (2003): *The complexity of XPath query evaluation*. In: *22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 179–190.

[20] Haruo Hosoya & Benjamin C. Pierce (2003): *XDuce: A statically typed XML processing language*. *ACM Trans. Internet Techn.* 3(2), pp. 117–148, doi:10.1145/767193.767195. Available at `https://doi.org/10.1145/767193.767195`.

[21] Leonid Libkin, Wim Martens & Domagoj Vrgoč (2013): *Querying Graph Databases with XPath*. In: *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, Association for Computing Machinery, New York, NY, USA, p. 129–140, doi:10.1145/2448496.2448513. Available at `https://doi.org/10.1145/2448496.2448513`.

[22] Anthony Lick (2019): *Logique de requêtes à la XPath : systèmes de preuve et pertinence pratique*. Theses, Université Paris-Saclay. Available at `https://tel.archives-ouvertes.fr/tel-02276423`.

[23] Wim Martens & Tina Trautner (2018): *Evaluation and Enumeration Problems for Regular Path Queries*. In Benny Kimelfeld & Yael Amsterdamer, editors: *21st International Conference on Database Theory (ICDT 2018)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 98, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 19:1–19:21, doi:10.4230/LIPIcs.ICDT.2018.19. Available at `http://drops.dagstuhl.de/opus/volltexte/2018/8594`.

[24] Kurt Mehlhorn (1980): *Pebbling Moutain Ranges and its Application of DCFL-Recognition*. In J. W. de Bakker & Jan van Leeuwen, editors: *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, *Lecture Notes in Computer Science* 85, Springer, pp. 422–435, doi:10.1007/3-540-10003-2_89. Available at `https://doi.org/10.1007/3-540-10003-2_89`.

[25] Barzan Mozafari, Kai Zeng & Carlo Zaniolo (2012): *High-performance complex event processing over XML streams*. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, Ariel Fuxman, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano & Ariel Fuxman, editors: *SIGMOD Conference*, ACM, pp. 253–264, doi:10.1145/2213836.2213866. Available at `http://dx.doi.org/10.1145/2213836.2213866`.

[26] Joachim Niehren & Momar Sakho (2021): *Determinization and Minimization of Automata for Nested Words Revisited*. *Algorithms*, doi:10.3390/a14030068. Available at `https://hal.inria.fr/hal-03134596`.

[27] Alexander Okhotin & Kai Salomaa (2014): *Complexity of input-driven pushdown automata*. *SIGACT News* 45(2), pp. 47–67, doi:10.1145/2636805.2636821. Available at `https://doi.org/10.1145/2636805.2636821`.

[28] Markus L. Schmid & Nicole Schweikardt (2021): *A Purely Regular Approach to Non-Regular Core Spanners*. In Ke Yi & Zhewei Wei, editors: *24th International Conference on Database Theory (ICDT 2021)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 186, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 4:1–4:19, doi:10.4230/LIPIcs.ICDT.2021.4. Available at `https://drops.dagstuhl.de/opus/volltexte/2021/13712`.

[29] H. Straubing (1994): *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Computer Science and Applied Series, Birkhäuser. Available at `https://books.google.fr/books?id=jLbH4LJbuDsC`.

[30] J. W. Thatcher (1967): *Characterizing derivation trees of context-free grammars through a generalization of automata theory*. *Journal of Computer and System Science* 1, pp. 317–322.

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

$$\frac{(q,s) \in I^{A\times S}}{q \in I^{\Pi_A(A\times S)}} \qquad \frac{(q,s) \in \mathcal{Q}^{A\times S}}{q \in \mathcal{Q}^{\Pi_A(A\times S)}} \qquad \frac{(q,s) \in F^{A\times S}}{q \in F^{\Pi_A(A\times S)}} \qquad \frac{(q_1,s_1) \xrightarrow{a} (q_2,s_2) \in \Delta^{A\times S}}{q_1 \xrightarrow{a} q_2 \in \Delta^{\Pi_A(A\times S)}}$$

Figure 19: Projection $\Pi_A(A \times S) = (\Sigma, \mathcal{Q}^{\Pi_A(A\times S)}, \Delta^{\Pi_A(A\times S)}, I^{\Pi_A(A\times S)}, F^{\Pi_A(A\times S)})$.

A Proofs for Section 1 (Introduction)

A.1 Further Related Work

Nested regular path queries. [21, 23] are formulas from propositional dynamic logic (PDL) [15]. While applicable to general data graphs, they can also be restricted to nested words. They extend on usual regular expression by adding nested filters that are closed under the logical operators. Filters may test for the existence of nodes answering a nested regular path query. When XML documents, nested regular path queries can be identified with regular forward XPath queries [5, 19]. t is folklore that nested regular path queries on data trees can be compiled to automata.

MSO. For ranked trees, one can first compile path queries to the monadic second-order (MSO) formulas, and from there to tree automata that recognize so-called V-structures [31, 18, 10]. V-structures are trees that are annotated with variables x satisfying $\mathit{words\text{-}one}^x_\Sigma$, saying where the any variable assignement has to assign a single node to variable x. In recent database terminology, languages of V-structures are called document spanners [14, 28]. But since the compilation of path queries to MSO formulas eagerly introduces quantifier alternations, that are to be eliminated by repeated automata determinization, this approach leads to a large size blowup.

Nested Regular Expressions. A more recent idea [7] is to compile nested regular path queries to nested regular expressions in a first step, i.e., regular expressions for nested words that were introduced earlier under the name *regular expression types* by Hosoya and Pierce [20]. Nested regular expressions support the usual operators of regular expressions and the nesting expressions $\langle e \rangle$ for defining languages of nested words. Furthermore, they support vertically recursive definitions based on μ-expressions $\mu x.e$, intersections $e \cap e'$, and complementation $\bar{e}$. It is then possible to compile nested regular expressions to SHAs or NWAs, by lifting the usual automata constructions from standard regular to nested regular expressions.

Forest Algebras. Nested words are very similar to forests, i.e., sequences of unranked trees. The transition relation of any SHA can be used to define a forest algebra [6].

Hedge Automata. Stepwise hedge automata (SHAs) improve on classical hedge automata [30, 11] in that they come with a satisfying notion of determinism.

B Proofs for Section 3 (Schema-Based Cleaning)

C Proofs for Section 4 (Schema-Based Determinization)

Proposition 6. *For any two* NFAs *A and S with the same alphabet:*

$$\mathit{scl}_S(A) = \widehat{\mathit{scl}}_S(A) \quad \text{and} \quad \mathcal{Q}^{A\times S} = \mathcal{Q}^{A\,\widehat{\times}\,S}$$

Proof. The two equations are shown by the following four lemmas. The judgements with a hat there are to be inferred by the collapsed system of inference rules in Fig. 20, while the other judgments are to be inferred with the rule system for accessible products in Fig. 7.

□

$$\frac{q \in I^A \quad s \in I^S}{q \in I^{\widehat{scl}_S(A)} \quad (q,s) \in \mathscr{Q}^{A\widehat{\times}S}} \qquad \frac{q \in F^A \quad s \in F^S \quad (q,s) \in \mathscr{Q}^{A\widehat{\times}S}}{q \in F^{\widehat{scl}_S(A)}}$$

$$\frac{(q,s) \in \mathscr{Q}^{A\widehat{\times}S}}{q \in \mathscr{Q}^{\widehat{scl}_S(A)}} \qquad \frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad (q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}}{q_1 \xrightarrow{a} q_2 \in \Delta^{\widehat{scl}_S(A)} \quad (q_2,s_2) \in \mathscr{Q}^{A\widehat{\times}S}}$$

$$\widehat{scl}_S(A) = (\Sigma, \mathscr{Q}^{\widehat{scl}_S(A)}, \Delta^{\widehat{scl}_S(A)}, I^{\widehat{scl}_S(A)}, F^{\widehat{scl}_S(A)})$$

Figure 20: A collapsed rule systems for schema-based cleaning $\widehat{scl}_S(A)$.

Lemma 13. *$q \in I^{\widehat{scl}_S(A)}$ iff $q \in I^{scl_S(A)}$.*

Proof. The rule systems of accessible product, projection, and the collapsed system can be used as following :

$$\frac{q \in I^A \quad s \in I^S}{q \in I^{\widehat{scl}_S(A)}} \qquad \frac{\dfrac{q \in I^A \quad s \in I^S}{(q,s) \in I^{A\times S}}}{q \in I^{scl_S(A)}}$$

□

Lemma 14. *$(q,s) \in \mathscr{Q}^{A\widehat{\times}S}$ iff $(q,s) \in \mathscr{Q}^{A\times S}$.*

Proof. We proof for all $n \geq 0$ that if $(q,s) \in \mathscr{Q}^{A\widehat{\times}S}$ has a proof tree of size n then there exists a proof tree for $(q,s) \in \mathscr{Q}^{A\times S}$. The proof is by induction on n.

In the case of the rules of the initial states, $(q,s) \in \mathscr{Q}^{A\widehat{\times}S}$ is inferred directly whenever $(q,s) \in \mathscr{Q}^{A\times S}$ and vice versa, using the following:

$$\frac{q \in I^A \quad s \in I^S}{(q,s) \in I^{A\times S} \quad (q,s) \in \mathscr{Q}^{A\times S}} \qquad \frac{q \in I^A \quad s \in I^S}{q \in I^{\widehat{scl}_S(A)} \quad (q,s) \in \mathscr{Q}^{A\widehat{\times}S}}$$

If $(q,s) \in \mathscr{Q}^{A\widehat{\times}S}$ is inferred by the internal rule of the collapsed rule system in Fig. 20. Then the proof tree has the following form for some proof tree T_1:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1}{(q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}}}{(q_2,s_2) \in \mathscr{Q}^{A\widehat{\times}S}}$$

This shows that there is a smaller proof tree T_1 for inferring $(q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}$. So by induction hypothesis applied to T_1, there exists a proof tree T_1' for inferring $(q_1,s_1) \in \mathscr{Q}^{A\times S}$ with the proof system of accessible products in Fig. 7:

$$\frac{T_1'}{(q_1,s_1) \in \mathscr{Q}^{A\times S}}$$

Therefore, we also have the following proof tree for $(q_2,s_2) \in \mathscr{Q}^{A\times S}$ with the internal rule for the accessible product:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathscr{Q}^{A\times S}}}{(q_2,s_2) \in \mathscr{Q}^{A\times S}}$$

For the inverse direction, if $(q,s) \in \mathscr{Q}^{A\times S}$ is inferred by the internal rule of the accessible product rule system in Fig. 7. Then the proof tree has the following form for some proof tree T_1:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1}{(q_1,s_1) \in \mathscr{Q}^{A\times S}}}{(q_2,s_2) \in \mathscr{Q}^{A\times S}}$$

This means that there is a smaller proof tree T_1 for inferring $(q_1,s_1) \in \mathscr{Q}^{A\times S}$. By induction hypothesis applied to T_1, there exists a proof tree T_1' for inferring $(q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}$ with the collapsed system in Fig. 20:

$$\frac{T_1'}{(q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}}$$

which leads to the following proof tree for $(q_2,s_2) \in \mathscr{Q}^{A\times S}$ with the internal rule for the collapsed system:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}}}{(q_2,s_2) \in \mathscr{Q}^{A\widehat{\times}S}}$$

□

Lemma 15. $q_1 \xrightarrow{a} q_2 \in \Delta^{\widehat{scl}_S(A)}$ *iff* $q_1 \xrightarrow{a} q_2 \in \Delta^{scl_S(A)}$.

Proof. We prove for all $n \geq 0$ that, if $q_1 \xrightarrow{a} q_2 \in \Delta^{\widehat{scl}_S(A)}$ has a proof tree of size n, then there exists a proof tree for $q_1 \xrightarrow{a} q_2 \in \Delta^{scl_S A}$ and vice versa. The proof is by induction on n.

If $q_1 \xrightarrow{a} q_2 \in \Delta^{\widehat{scl}_S(A)}$ is inferred by the internal rule of the collapsed system, the proof tree will have the following for some tree T_1:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1}{(q_1,s_1) \in \mathscr{Q}^{A\widehat{\times}S}}}{q_1 \xrightarrow{a} q_2 \in \Delta^{\widehat{scl}_S(A)}}$$

By Lemma 14 and the rule of internal rules of the accessible product rule system:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathscr{Q}^{A\times S}}}{(q_1,s_1) \xrightarrow{a} (q_2,s_2) \in \Delta^{A\times S}}$$

For the inverse direction, if $q_1 \xrightarrow{a} q_2 \in \Delta^{scl_S(A)}$ is inferred by the internal rule of the accessible product, the proof tree will have the following for some tree T_1:

$$\frac{\dfrac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1}{(q_1,s_1) \in \mathscr{Q}^{A\times S}}}{(q_1,s_1) \xrightarrow{a} (q_2,s_2) \in \Delta^{A\times S}}}{q_1 \xrightarrow{a} q_2 \in \Delta^{scl_S(A)}}$$

$$\frac{Q \in I^{det(A)} \quad s \in I^S}{Q \in I^{\widehat{scl}_S(det(A))} \quad (Q,s) \in \mathcal{Q}^{det(A)\widehat{\times}S}} \qquad \frac{Q \in F^{det(A)} \quad s \in F^S \quad (Q,s) \in \mathcal{Q}^{det(A)\times S}}{Q \in F^{\widehat{scl}_S(det(A))}}$$

$$\frac{(Q,s) \in \mathcal{Q}^{det(A)\widehat{\times}S}}{Q \in \mathcal{Q}^{\widehat{scl}_S(det(A))}} \qquad \frac{Q_1 \xrightarrow{a} Q_2 \in \Delta^{det(A)} \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad (Q_1,s_1) \in \mathcal{Q}^{det(A)\times S}}{Q_1 \xrightarrow{a} Q_2 \in \Delta^{\widehat{scl}_S(det(A))} \quad (Q_2,s_2) \in \mathcal{Q}^{det(A)\widehat{\times}S}}$$

$$\widehat{scl}_S(det(A)) = (\Sigma, \mathcal{Q}^{\widehat{scl}_S(det(A))}, \Delta^{\widehat{scl}_S(det(A))}, I^{\widehat{scl}_S(det(A))}, F^{\widehat{scl}_S(det(A))})$$

Figure 21: Instantiation of the collapsed rule system for schema-based cleaning from Fig. 20 with $\det(A)$.

By lemma 14 and the rule of internal rules of the collapsed system:

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathcal{Q}^{A\times S}}}{q_1 \xrightarrow{a} q_2 \in \Delta^{\widehat{scl}_S(A)}}$$

□

Lemma 16. *$q \in \mathcal{Q}^{\widehat{scl}_S(A)}$ iff $q \in \mathcal{Q}^{scl_S(A)}$ and $q \in F^{\widehat{scl}_S(A)}$ iff $q \in F^{scl_S(A)}$.*

Proof. We start proving $q \in \mathcal{Q}^{\widehat{scl}_S(A)}$ iff $q \in \mathcal{Q}^{scl_S(A)}$. By Lemma 14, and rules of construction of the accessible product, projection, and collapsed systems, this lemma holds for some proof trees T and T' as follows:

$$\frac{\dfrac{T}{(q,s) \in \mathcal{Q}^{A\times S}}}{q \in \mathcal{Q}^{scl_S(A)}} \qquad \frac{T'}{q \in \mathcal{Q}^{\widehat{scl}_S(A)}}$$

Finally, we show $q \in F^{\widehat{scl}_S(A)}$ iff $q \in F^{scl_S(A)}$. Using Lemma 14, there exists some proof trees T and T' that infers $(q,s) \in \mathcal{Q}^{A\times S}$ and $(q,s) \in \mathcal{Q}^{A\widehat{\times}S}$ in both ways and therefore having the following form of rules:

$$\frac{q \in F^A \quad s \in F^S \quad \dfrac{T}{(q,s) \in \mathcal{Q}^{A\widehat{\times}S}}}{q \in F^{\widehat{scl}_S(A)}} \qquad \frac{q \in F^A \quad s \in F^S \quad \dfrac{T'}{(q,s) \in \mathcal{Q}^{A\times S}}}{\dfrac{(q,s) \in F^{A\times S}}{q \in F^{scl_S(A)}}}$$

□

D Proofs for Section 5 (Stepwise Hedge Automata for Nested Words)

Proposition 10. *The accessible determinization of a* SHA *can be computed in expected amortized time* $O(|\mathcal{Q}^{\det(A)}|^2\ |\Delta^A| + |A|)$.

Proof. An algorithm for computing the fixed points of the inference rules of accessible determinization of a SHA is presented in Fig. 22. It extends on the case of NFAs with the same data structures. It uses dynamic perfect hashing for the hash sets. The additional treatment of apply rules, that dominates the complexity of the algorithm, works as follows: for each $Q \in \mathcal{Q}^{det(A)}$ in the *Agenda* and each state $Q1 \in \mathcal{Q}^{det(A)}$ in the *Store*, it computes the sets $Q' = \{q' \mid q@q_1 \to q', q_1 \in Q_1, q \in Q\}$ and $Q'' = \{q'' \mid q_1@q \to$

```
1  fun detSHA(A) =
2    let Store = hashset.new(∅)
3    let Agenda = list.new() and Rules = hashset.new(∅)
4    if init^A ≠ ∅ then Agenda.add(init^A)
5    while Agenda.notEmpty() do
6      let (Q) = Agenda.pop()
7      let h be an empty hash table with keys from Σ.
8        // the values will be nonempty hash subsets of 2^A
9      for q -a-> q' ∈ Δ^A such that q ∈ Q do
10       if h.get(a) = undef then h.add(a,hashset.new(∅))
11     (h.get(a)).add(q')
12     for (a,Q') in h.tolist() do Rules.add(Q -a-> Q')
13       if not Store.member(Q') then Store.add(Q') Agenda.push(Q')
14     for Q1 ∈ Store do
15       let Q' = {q' | q@q1 → q', q1 ∈ Q1, q ∈ Q}
16       if Q' ≠ ∅ then Rules.add(Q@Q1 → Q')
17                 if not Store.member(Q') then Store.add(Q') Agenda.push(Q')
18       let Q'' = {q'' | q1@q → q'', q1 ∈ Q1, q ∈ Q}
19       if Q'' ≠ ∅ then Rules.add(Q1@Q → Q'')
20   let init^det(A) = {Q | Q ∈ Store, Q ∩ init^A ≠ ∅} and F^det(A) = {Q | Q ∈ Store, Q ∩ F^A ≠ ∅}
21   return (Σ, Store.toSet(), Rules.toSet(), init^det(A), F^det(A))
```

Figure 22: An algorithm for accessible determinization of SHAs.

$$\frac{(q,s) \in \langle\rangle^{\Delta^{A\times S}}}{q \in \langle\rangle^{\Delta^{\Pi_A(A\times S)}}} \qquad \frac{(q_1,s_1)@(q,s) \to (q_2,s_2) \in \Delta^{A\times S}}{q_1@q \to q_2 \in \Delta^{\Pi_A(A\times S)}}$$

Figure 23: Lifting projections $\Pi_A(A \times S)$ to SHAs.

$q'', q_1 \in Q_1, q \in Q\}$ and puts all new non-empty sets in both the *Agenda* and the *Store*, while adding dynamically the generated apply rules in the hash set *Rules*. Again, the overall number of elements in the agenda will be $|\mathcal{Q}^{det(A)}|$, requiring time in $O(|\mathcal{Q}^{det(A)}|^2\ |\Delta^A|)$. With a precomputation time of A in $O(|A|)$, the total computation will be in $O(|\mathcal{Q}^{\det(A)}|^2\ |\Delta^A| + |A|)$. □

E Proofs for Section 6 (Schema-Based Determinization for SHAs)

Proposition 11. *If A and S are dSHAs then the accessible product $A \times S$ and the schema-based cleaning $scl_S(A)$ can be computed in expected amortized time $O(|\mathcal{Q}^{A\times S}|^2 + |\mathcal{Q}^{A\times S}||\Sigma| + |A| + |S|)$.*

The algorithm in Fig. 24 is obtained by lifting the algorithm for DFAs in Fig. 11 to SHAs. For the case of apply rules, we have to combine each pair $(q_1,s_1) \in \mathcal{Q}^{A\times S}$ in the stack *Agenda* with all $(q,s) \in \mathcal{Q}^{A\times S}$ in the hash set *Store*, in both directions. The time to treat these pairs is $O(|\mathcal{Q}^{A\times S}|^2)$, so quadratic in the worst case. As before, no state (q_1,s_1) will be processed twice, due to the set membership test before pushing a pair into the agenda.

Proposition 12. *The schema-based determinization $det_S(A)$ of a* SHA *A with respect to a d*SHA *S can be computed in expected amortized time $O(|\mathcal{Q}^{\det(A)\times S}|^2 + |\mathcal{Q}^{\det(A)\times S}|\ |\Sigma| + |\mathcal{Q}^{det_S(A)}|^2\ |\Delta^A| + |A| + |S|)$.*

Proof. Analogously to the case of NFAs on words. The algorithm in Fig. 25 computes the fixed point of the inference rules of schema-based determinization of SHAs. As for NFAs, it stores untreated alignments

```
1  fun A × S =
2    let Store = hashset.new(∅)
3    let Agenda = list.new() and Rules = hashset.new(∅)
4    if init^A = {q0} and init^S = {s0} then Agenda.add((q0,s0))
5    while Agenda.notEmpty() do
6      let (q1,s1) = Agenda.pop()
7      for a ∈ Σ do
8        let Q = {q2 | q1 -a-> q2 ∈ Δ^A} and R = {s2 | s1 -a-> s2 ∈ Δ^S}
9        for q2 ∈ Q and s2 ∈ R do Rules.add((q1,s1) -a-> (q2,s2))
10         if not Store.member((q2,s2))
11           then Store.add((q2,s2)) Agenda.push((q2,s2))
12     for (q,s) ∈ Store do
13       let Q' = {q2 | q1@q -> q2 ∈ Δ^A} and R' = {s2 | s1@s -> s2 ∈ Δ^S}
14       for q2 ∈ Q' and s2 ∈ R' do Rules.add((q1,s1) -(q,s)-> (q2,s2))
15         if not Store.member((q2,s2))
16           then Store.add((q2,s2)) Agenda.push((q2,s2))
17       let Q'' = {q2 | q@q1 -> q2 ∈ Δ^A} and R'' = {s2 | s@s1 -> s2 ∈ Δ^S}
18       for q2 ∈ Q'' and s2 ∈ R'' do Rules.add((q,s) -(q1,s1)-> (q2,s2))
19         if not Store.member((q2,s2))
20           then Store.add((q2,s2)) Agenda.push((q2,s2))
21   let init^{A×S} = {(q0,s0) | (q0,s0) ∈ Store} and F^{A×S} = {(q,s) | (q,s) ∈ Store, q ∈ F^A, s ∈ F^S}
22   return (Σ, Store.toSet(), Rules.toSet(), init^{A×S}, F^{A×S})
```

Figure 24: An algorithm computing the accessible product of dSHAs A and S.

on a stack *Agenda* and processed alignments in a hash set *Store*. It also collects transition rules in a hash set *Rules*. New alignments can now be produced by the the inference rule for apply transitions: for each alignment $Q_1 \sim s_1$ on the *Agenda* and $Q_2 \sim s_2$ in the *Store*, the algorithm computes the sets $\{s \mid s_1@s_2 \to s \in \Delta^S\}$ and $\{Q \mid Q_1@Q_2 \to Q \in \Delta^{det(A)}\}$ and pushes all pairs $Q \sim s$ outside the *Store* to the *Agenda*. There may be at most one such pair since S and $det(A)$ are deterministic. We also have to consider the symmetric case where $Q_1 \sim s_1$ on the store and $Q_2 \sim s_2$ on the *Agenda*. Thus, it is in time $O(|\mathscr{Q}^{\det(A)\times S}|^2)$ which is quadratic in the worst case. Added to the latter, the cost of computing the transition of $det(A)$ on the fly which is in worst case $O(|\mathscr{Q}^{det_S(A)}|^2\ |\Delta^A| + |A|)$. Therefore, having the whole algorithm running, including the time for computing the internal rules, in $O(|\mathscr{Q}^{\det(A)\times S}|^2 + |\mathscr{Q}^{\det(A)\times S}|\ |\Sigma| + |\mathscr{Q}^{det_S(A)}|^2\ |\Delta^A| + |A| + |S|)$. □

F Proof of Correctness Theorem 2

The proof extends on the proof of the case of words (Theorem 1) in a direct manner.

We first lift the collapsed rule system for NFAs from Fig. 20 to SHAs inFig. 26, and then show that collapsed rules also redefine the schema-based cleaning $\widehat{scl_S}(A) = scl_S(A)$ in the case of SHAs.

Proposition 17. *For any two* SHA*s A and S with the same alphabet:*

$$\Pi_A(A \times S) = \widehat{scl_S}(A) \quad \textit{and} \quad \mathscr{Q}^{A\times S} = \mathscr{Q}^{A\widehat{\times} S}$$

Proof. The two equations are shown either by new lemmas or an extension of the lemmas from the proof of Theorem 1, whereas all unchanged existing lemmas hold(Lemmas 13, 15 and 16).

□

```
1  fun detS(A,S) =
2    let Store = hashset.new(∅)
3    let Agenda = list.new() and Rules = hashset.new(∅)
4    if init^A ≠ ∅ and init^S = {s_0} then Agenda.add(init^A ∼ s_0)
5    while Agenda.notEmpty() do
6      let (Q_1 ∼ s_1) = Agenda.pop()
7      for a ∈ Σ do
8        let P = {Q_2 | Q_1 -a-> Q_2 ∈ Δ^det(A)} and R = {s_2 | s_1 -a-> s_2 ∈ Δ^S}
9        for Q_2 ∈ P and s_2 ∈ R do Rules.add(Q_1 -a-> Q_2)
10           if not Store.member(Q_2 ∼ s_2)
11             then Store.add(Q_2 ∼ s_2) Agenda.push(Q_2 ∼ s_2)
12     for (Q ∼ s) ∈ Store do
13       let P' = {Q_2 | Q_1@Q → Q_2 ∈ Δ^det(A)} and R' = {s_2 | s_1@s → s_2 ∈ Δ^S}
14       for Q_2 ∈ P' and s_2 ∈ R' do Rules.add(Q_1@Q → Q_2)
15           if not Store.member(Q_2 ∼ s_2)
16             then Store.add(Q_2 ∼ s_2) Agenda.push(Q_2 ∼ s_2)
17       let P'' = {Q_2 | Q@Q_1 → Q_2 ∈ Δ^det(A)} and R'' = {s_2 | s@s_1 → s_2 ∈ Δ^S}
18       for Q_2 ∈ P'' and s_2 ∈ R'' do Rules.add(Q@Q_1 → Q_2)
19           if not Store.member(Q_2 ∼ s_2)
20             then Store.add(Q_2 ∼ s_2) Agenda.push(Q_2 ∼ s_2)
21   let init^detS(A) = {Q | Q ∼ s ∈ Store, Q ∩ init^A ≠ ∅} and F^detS(A) = {Q | Q ∼ s ∈ Store, Q ∩ F^A ≠ ∅}
22   return (Σ, Store.toSet(), Rules.toSet(), init^detS(A), F^detS(A))
```

Figure 25: An algorithm for schema-based determinization of an SHA A and a dSHA schema S

$$\frac{q \in \langle\!\rangle^{\Delta^A} \quad s \in \langle\!\rangle^{\Delta^S}}{q \in \langle\!\rangle^{\Delta^{\widehat{scl}_S(A)}} \quad (q,s) \in \mathcal{Q}^{A\widehat{\times}S}}$$

$$\frac{q_1@q \to q_2 \in \Delta^A \quad s_1@s \to s_2 \in \Delta^S \quad (q_1,s_1) \in \mathcal{Q}^{A\widehat{\times}S} \quad (q,s) \in \mathcal{Q}^{A\widehat{\times}S}}{q_1@q \to q_2 \in \Delta^{\widehat{scl}_S(A)} \quad (q_2,s_2) \in \mathcal{Q}^{A\widehat{\times}S}}$$

Figure 26: Lifting the collapsed rule system from NFAs to SHAs.

Lemma 18. $q \in \langle\!\rangle^{\Delta^{\widehat{scl}_S(A)}}$ *iff* $\langle\!\rangle^{\Delta^{scl_S(A)}}$

Proof. The rule systems of accessible product, projection and the collapsed system can be used as following :

$$\frac{q \in \langle\!\rangle^{\Delta^A} \quad s \in \langle\!\rangle^{\Delta^S}}{q \in \langle\!\rangle^{\Delta^{\widehat{scl}_S(A)}}} \qquad \frac{\dfrac{q \in \langle\!\rangle^{\Delta^A} \quad s \in \langle\!\rangle^{\Delta^S}}{(q,s) \in \langle\!\rangle^{\Delta^{A\times S}}}}{q \in \langle\!\rangle^{\Delta^{scl_S(A)}}}$$

□

Lemma 19 (extends Lemma 14)**.** $(q,s) \in \mathcal{Q}^{A\widehat{\times}S}$ *iff* $(q,s) \in \mathcal{Q}^{A\times S}$.

All proofs for initial states rules and internal rules from the previous lemma hold and we extend it for tree initial rules and apply rules:

Proof. Similarly, we prove for all $n \geq 0$ that if $(q,s) \in \mathcal{Q}^{A\widehat{\times}S}$ has a proof tree of size n then there exists a proof trees for $(q,s) \in \mathcal{Q}^{A\times S}$. The proof is by induction on n.

In the case of tree initial rules, $(q,s) \in \mathcal{Q}^{A \widehat{\times} S}$ is inferred directly whenever $(q,s) \in \mathcal{Q}^{A \times S}$ and vice versa, using the following:

$$\frac{q \in \langle\rangle^{\Delta^A} \quad s \in \langle\rangle^{\Delta^S}}{(q,s) \in \langle\rangle^{\Delta^{A\times S}} \quad (q,s) \in \mathcal{Q}^{A\times S}} \qquad \frac{q \in \langle\rangle^{\Delta^A} \quad s \in \langle\rangle^{\Delta^S}}{q \in \langle\rangle^{\widehat{scl}_S(A)} \quad (q,s) \in \mathcal{Q}^{A\widehat{\times}S}}$$

In the same spirit, if $(q,s) \in \mathcal{Q}^{A \widehat{\times} S}$ is inferred by the apply rule of the same system, then the proof tree has the following form for some proof trees T_1 and T:

$$\frac{q_1 @ q \to q_2 \in \Delta^A \quad s_1 @ s \to s_2 \in \Delta^S \quad \dfrac{T_1}{(q_1,s_1) \in \mathcal{Q}^{A\widehat{\times}S}} \quad \dfrac{T}{(q,s) \in \mathcal{Q}^{A\widehat{\times}S}}}{(q_2,s_2) \in \mathcal{Q}^{A\widehat{\times}S}}$$

This means that there are smaller proof trees T_1 and T for inferring respectively $(q_1,s_1) \in \mathcal{Q}^{A\widehat{\times}S}$ and $(q,s) \in \mathcal{Q}^{A\widehat{\times}S}$. Correspondingly, by induction hypothesis applied to T_1 and T, there exists T_1', T' for inferring $(q_1,s_1) \in \mathcal{Q}^{A\times S}$ and $(q,s) \in \mathcal{Q}^{A\times S}$:

$$\frac{T_1'}{(q_1,s_1) \in \mathcal{Q}^{A\times S}} \qquad \frac{T'}{(q,s) \in \mathcal{Q}^{A\times S}}$$

Thus allowing the following proof tree for $(q_2,s_2) \in \mathcal{Q}^{A\times S}$ with the apply rule of the accessible product:

$$\frac{q_1 @ q \to q_2 \in \Delta^A \quad s_1 @ s \to s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathcal{Q}^{A\times S}} \quad \dfrac{T'}{(q,s) \in \mathcal{Q}^{A\times S}}}{(q_2,s_2) \in \mathcal{Q}^{A\times S}}$$

For the inverse direction of the apply rules, and using the induction hypothesis, we will be able to infer $(q_2,s_2) \in \mathcal{Q}^{A\widehat{\times}S}$ with some T_1' and T' and ending with the following proof tree:

$$\frac{q_1 @ q \to q_2 \in \Delta^A \quad s_1 @ s \to s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathcal{Q}^{A\widehat{\times}S}} \quad \dfrac{T'}{(q,s) \in \mathcal{Q}^{A\widehat{\times}S}}}{(q_2,s_2) \in \mathcal{Q}^{A\widehat{\times}S}}$$

□

Lemma 20. *$q_1 @ q \to q_2 \in \Delta^{\widehat{scl}_S(A)}$ iff $q_1 @ q \to q_2 \in \Delta^{scl_S(A)}$.*

Proof. Following the same logic in Lemma 15, this lemma holds by the following sequence of rules, for some proof trees T_1, T, T_1' and T' :

$$\frac{q_1 @ q \to q_2 \in \Delta^A \quad s_1 @ s \to s_2 \in \Delta^S \quad \dfrac{T_1}{(q_1,s_1) \in \mathcal{Q}^{A\widehat{\times}S}} \quad \dfrac{T}{(q,s) \in \mathcal{Q}^{A\widehat{\times}S}}}{q_1 @ q \to q_2 \in \Delta^{\widehat{scl}_S(A)}}$$

$$\frac{q_1 @ q \to q_2 \in \Delta^A \quad s_1 @ s \to s_2 \in \Delta^S \quad \dfrac{T_1'}{(q_1,s_1) \in \mathcal{Q}^{A\times S}} \quad \dfrac{T'}{(q,s) \in \mathcal{Q}^{A\times S}}}{(q_1,s_1) @ (q,s) \to (q_2,s_2) \in \Delta^{A\times S}}$$

$$\frac{Q \in \langle\rangle^{\Delta^{det(A)}} \quad s \in \langle\rangle^{\Delta^{S}}}{Q \in \langle\rangle^{\Delta^{\widehat{scl}_S(det(A))}} \quad (Q,s) \in \mathcal{Q}^{det(A)\widehat{\times}S}}$$

$$\frac{Q_1@Q \to Q_2 \in \Delta^{det(A)} \quad s_1@s \to s_2 \in \Delta^{S} \quad (Q_1,s_1) \in \mathcal{Q}^{det(A)\widehat{\times}S} \quad (Q,s) \in \mathcal{Q}^{det(A)\widehat{\times}S}}{Q_1@Q \to Q_2 \in \Delta^{\widehat{scl}_S(det(A))} \quad (Q_2,s_2) \in \mathcal{Q}^{det(A)\widehat{\times}S}}$$

Figure 27: Extending the instantiation of the alt. definition of schema-based cleaning: $\widehat{scl}_S(det(A)) = (\Sigma, \mathcal{Q}^{\widehat{scl}_S(det(A))}, \Delta^{\widehat{scl}_S(det(A))}, I^{\widehat{scl}_S(det(A))}, F^{\widehat{scl}_S(det(A))})$.

$$\frac{(q_1,s_1)@(q,s) \to (q_2,s_2) \in \Delta^{A\times S}}{q_1@q \to q_2 \in \Delta^{\widehat{scl}_S(A)}}$$

For the inverse direction, the sequence of rules, for some proof trees T_1, T, T_1' and T' will be:

$$\frac{q_1@q \to q_2 \in \Delta^{A} \quad s_1@s \to s_2 \in \Delta^{S} \quad \dfrac{T_1}{(q_1,s_1) \in \mathcal{Q}^{A\times S}} \quad \dfrac{T}{(q,s) \in \mathcal{Q}^{A\times S}}}{\dfrac{(q_1,s_1)@(q,s) \to (q_2,s_2) \in \Delta^{A\times S}}{q_1@q \to q_2 \in \Delta^{scl_S(A)}}}$$

$$\frac{q_1@q \to q_2 \in \Delta^{A} \quad s_1@s \to s_2 \in \Delta^{S} \quad \dfrac{T_1'}{(q_1,s_1) \in \mathcal{Q}^{A\widehat{\times}S}} \quad \dfrac{T'}{(q,s) \in \mathcal{Q}^{A\widehat{\times}S}}}{q_1@q \to q_2 \in \Delta^{\widehat{scl}_S(A)}}$$

□

Proof of Correctness of Theorem 2. For the proof of theorem for nested words with SHA, we extend the instantiation of the rule system for schema-based cleaning from Fig. 26 with $\det(A)$, yielding the rule system in Fig. 27. The whole instantiation holds with the previous instantiation for words and we can still identify the rule system for $\widehat{scl}_S(det(A))$ with the rule system $det_S(A)$. With the same identification of judgements and predicate renaming, the two systems are still exactly the same. Having $\widehat{scl}_S(det(A)) = det_S(A)$ implies, by Proposition 17 $scl_S(det(A)) = det_S(A)$. □

G Proofs for Section 7 (Experiments)

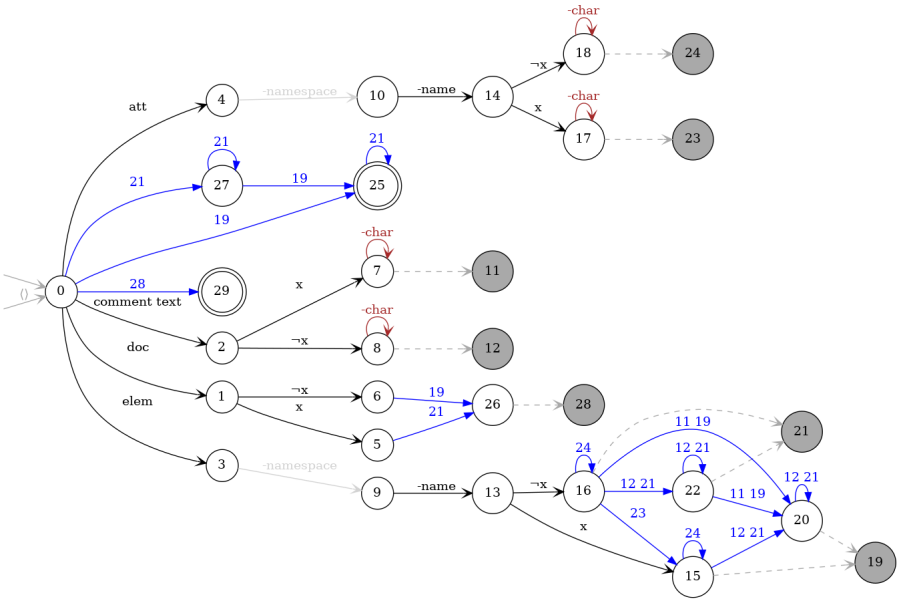


Figure 28: A schema for the intersection of *XML* data model with one^x.

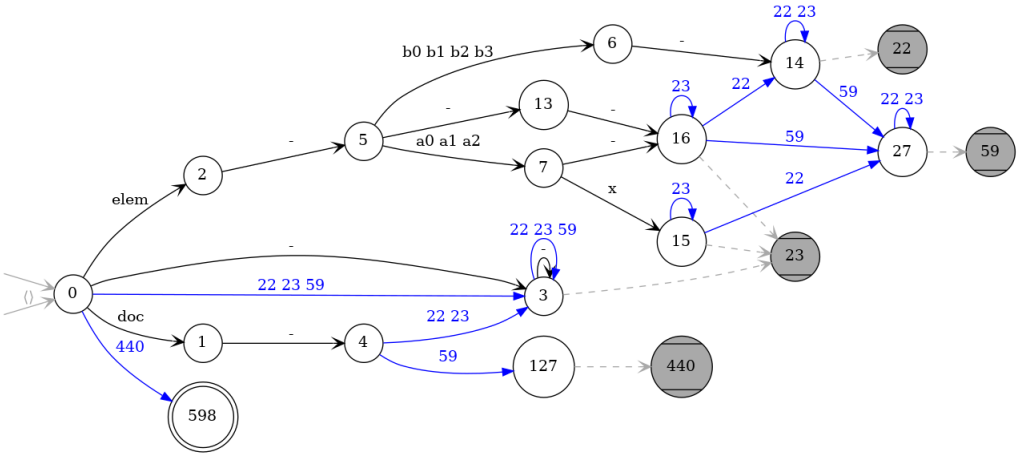


Figure 29: The automaton $mini(det_S(A))$ of the query Q3.4.

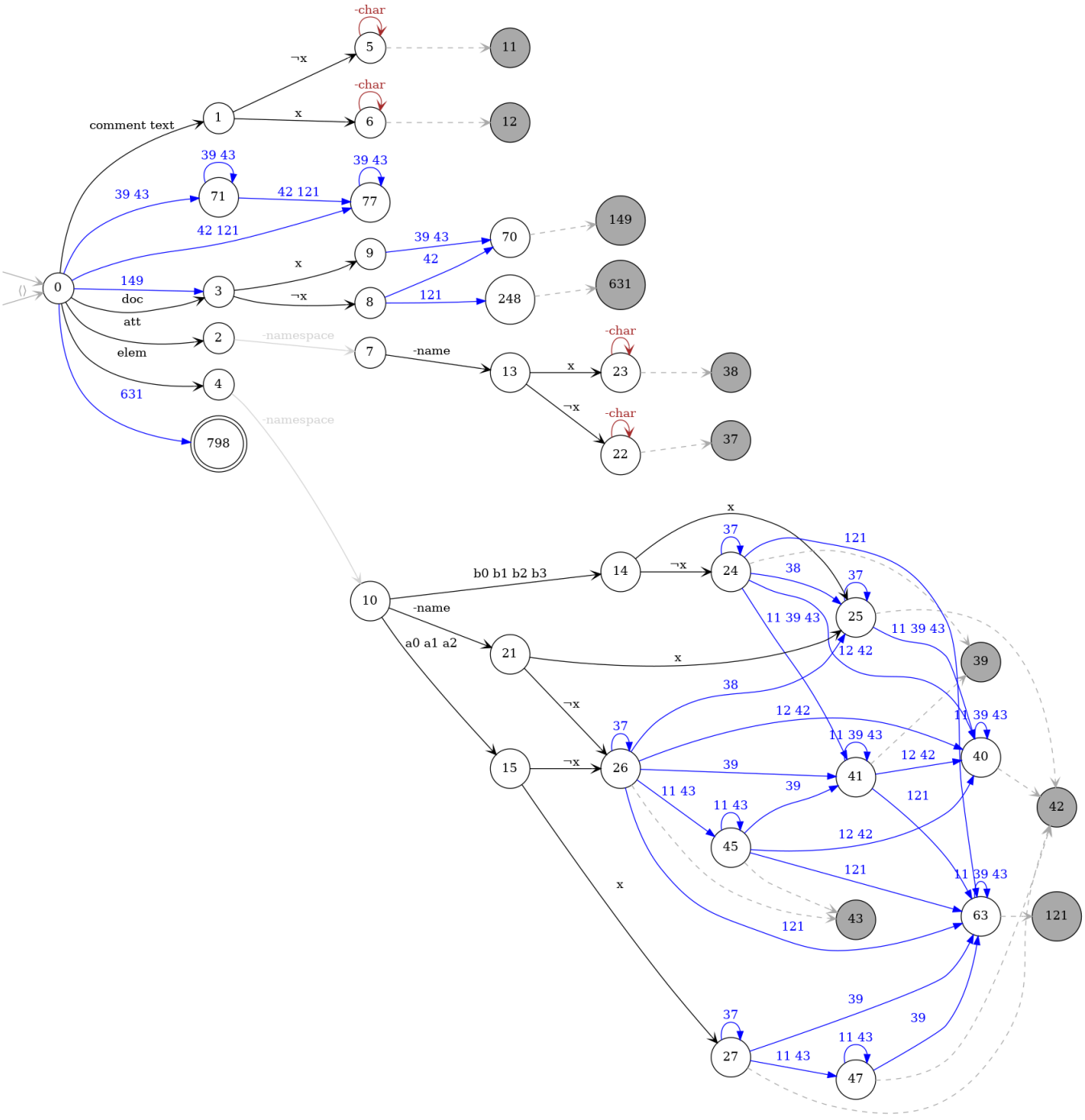


Figure 30: The automaton $mini(det(A \times S))$ of the query Q3.4.