Nested Regular Expressions can be Compiled to Small Deterministic Nested Word Automata

Iovka Boneva, Joachim Niehren, Momar Sakho

Nested Regular Expressions can be Compiled to Small Deterministic Nested Word Automata

Iovka Boneva[1], Joachim Niehren[2], and Momar Sakho[1,2]

[1] Université de Lille, [2] Inria Lille, France

Abstract. We study the problem of whether regular expressions for nested words can be compiled to small deterministic *nested word automata* (NWAs). In theory, we obtain a positive answer for small deterministic regular expressions for nested words. In practice of navigational path queries, nondeterministic NWAs are obtained for which NWA determinization explodes. We show that practical good solutions can be obtained by using *stepwise hedge automata* as intermediates.

Keywords: Automata · Regular Expressions · Nested Words · XPath

1 Introduction

Nested words are nested structures omnipresent in computer science. They were used in particular to represent data trees or Xml documents, or to analyze the call structure of recursive programs. The idea of nested words is to generalize words and unranked trees at the same time. Nested words can be obtained by enriching Dyck words with internal letters, beside of opening and closing parenthesis. Nested words can also be defined recursively as the elements of the least set that contains internal letters from a given alphabet, triples consisting of an opening parenthesis, a nested word, and a closing parenthesis, and all sequences of nested words. Alternatively, nested word can be specified as finite sequences of internal letters, opening closing and parenthesis. Only well-nested sequence are permitted in which every opening parenthesis is properly closed and every closed parenthesis is properly opened. Or else, nested words can be identified with sequences of unranked trees, which are often called hedges.

From the view point of formal language theory, the natural question is how to lift and relate the notions of finite automata and regular expression for words and trees to the case of nested word. Automata for nested words (NWAs) are well studied [1,3,23] and also known as visibly pushdown automata. While having the same expressiveness as hedge automata [26,10], which generalize tree automata from ranked to unranked trees, they are often defined as pushdown automata with visible stacks, meaning that exactly one symbol is pushed when reading an opening parenthesis, and exactly one symbol is popped when reading a closing parenthesis, while the stack is not used otherwise. Their main advantage is a powerful notion of determinism, generalizing both over bottom-up and top-down determinism of tree automata for ranked trees [1]. In contrast to more general pushdown automata, NWAs permit determinization, basically since they are so closely related to tree automata.

Regular expressions for nested words were first introduced under the name of *regular expression types* by Hosoya et. al. in the context of the XML programming language XDuce [19]. We will call them *nested regular expressions* (NREs) instead. Independently, more complex notions of regular expressions were proposed [21,25] that can also deal to some extent with generalizations of nested words, in which dangling opening and closing parentheses are permitted. It was already claimed in [19], that NREs have the same expressiveness as hedge automata [26,10], which in turn have the same expressiveness as NWAs [1]. However, the question under which conditions nested words can be compiled to small deterministic NWAs has not been studied. Whenever possible, one can decide language inclusion or equivalence in P. Otherwise, these problems may not be feasible since EXP-complete for general NWAs or NREs.

Our concrete interest in the universality of deterministic NWAs is motivated by XML stream processing: we want to compute the certain answers of a CoreXPath query on an XML stream [24,15], i.e., those elements that are selected in all possible futures of the stream. Whether an answer is certain is computationally hard for tiny syntactic fragments of CoreXPath [4,15], but can be done in polynomial time for queries defined by deterministic NWAs [16]. A natural question is therefore, whether it is possible to compile CoreXPath queries as in the usual benchmark [14] to deterministic NWAs of reasonable size. Unfortunately, the existing compilers fail to do so [12], since they are based on NWA determinization for dealing with disjunction, negation, and recursive steps. Thereby they produce huge deterministic automata even for very simple CoreXPath queries from the benchmark, or do not terminate after some hours.

In this paper, we consider NREs for defining queries on nested words, since there exist compilers that can map the CoreXPath queries from the usual benchmark to NREs of reasonable size, under the condition that the path query contains only forwards steps. We then distinguish a subclass of "deterministic" NREs that can be complied in polynomial time to deterministic NWAs by generalizing on Glushkov's construction of deterministic finite-state automata (DFAs) from "deterministic" regular expressions [6,7]. However, the NREs obtained by compilation from CoreXPath queries are rarely deterministic, so neither are the NWAs obtained from them by direct compilation. Neither can we apply NWA determinization to them as argued above. We show that deterministic NWAs can be obtained nevertheless based on stepwise hedge automata (SHAs), that we introduce. SHAs combine stepwise tree automata [8] for unranked trees with finite state automata on words (NFAs). They can be determinized in a bottom-up and left-to-right manner, simply by combining the determinization procedures for tree automata and for NFAs. Furthermore, we can compile deterministic SHAs to deterministic NWAs in polynomial time. Conversely, NWAs can be compiled to SHAs in polynomial time too, but at the cost of introducing nondeterminism.

By composing these compilers and determinization algorithms, NREs can be compiled to deterministic NWAs in the following two manners. The first method is to compile the NRE to an SHA, from there to an NWA, which is then determinized. The second way consists of compiling the NRE to an SHA, determinize it, and convert the result to a deterministic NWA. In an experimental study, we consider a collection of NREs that we constructed automatically

from CoreXPath queries in the XMark benchmark [14]. It turns out a little surprisingly that both above algorithms yield a satisfactory solution: they produce small deterministic NWAs for all NREs in our collection. The sizes of the deterministic may differ, sometimes in favor of the one or the other algorithm. We also discuss, why the NWA determinization behaves reasonably for the NWAs obtained from SHAs, while it behaved so badly for NWAs obtained directly from NREs. The reason seems to be that the former NWAs in contrast to the latter have the single entry property, which basically states that the NWA performs all its work in a bottom-up and left-to-right manner, and none when moving top-down. This conjecture is supported by practical evidence rather than some formal statement.

Related Work. CoreXPath [17] is a fragment of nested regular path queries on data trees, in which recursion is restricted to basic steps up, down, left and right. Nested regular path queries were introduced in the seventies [13] under the name of the propositional dynamic logic (PDL). There they were applied to general labeled graphs, rather than being restricted to data trees.

Since certain query answering for CoreXPath was considered as difficult, the currently existing approaches to CoreXPath evaluation on XML streams [12,24] either approximate certain query answers based on nondeterministic machines or restrict the queries so that answers certainty can be decided without latency [22,4]. This also holds for recent streaming algorithms on words without nesting in the context of complex event processing [18].

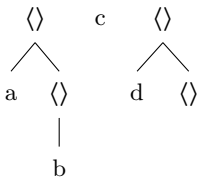
2 Nested Words

Nested words are words with parentheses that are well-nested. They can be identified with hedges, that is sequences of internal symbols and unranked trees.

Nested words are constructed with an opening and a closing parentheses, respectively $\langle$ and $\rangle$. An unranked alphabet Σ is a possibly infinite set of so called "internal" symbols, that does not contain the two parentheses. Nested words over Σ then have the following abstract syntax:

$$h, h' ::= \varepsilon \quad | \quad a \quad | \quad \langle h \rangle \quad | \quad h \cdot h' \qquad \text{where } a \in \Sigma$$

The empty word is denoted by ε and assumed to satisfy $\varepsilon \cdot h = h = h \cdot \varepsilon$. Nested words can be identified with hedges, i.e., words of trees and internal symbols. 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 $\langle h \rangle$.



Variants. Our notion of nested words accepts only well-nested words without dangling opening or closing parentheses in contrast to others [1,3]. This will lead to simpler notion of regular expressions, avoiding the more complex operators as with visibly rational expressions [5,25]. A less important difference is that we do not support labeled parentheses.

Labeled trees. Labeled parentheses can be simulated by using internal letters. For instance, the labeled tree $a(b(), c())$ can be represented by the nested word

of type tree $\langle a \cdot \langle b \rangle \cdot \langle c \rangle \rangle$. In this way, the labeled tree $a()$ is represented by the nested word $\langle a \rangle$ which is of type tree (while the internal letter a alone is not). XML documents are particular labeled trees, such as 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 trees satisfying the XML data model can be represented as nested words over a signature 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 $\langle elem \cdot a \cdot \langle attr \cdot name \cdot u \cdot f \cdot f \rangle \langle elem \cdot b \cdot \langle text \cdot i \cdot s \cdot g \cdot a \cdot g \cdot a \rangle \langle elem \cdot d \rangle \rangle \langle elem \cdot c \rangle \rangle$

3 Nested Regular Expressions

We present nested regular expressions (NREs), that were introduced under the name *regular expression types* in the context of XDuce [19] up to minor details. A NRE over alphabet Σ has the following abstract syntax:

$$E, E' ::= \varepsilon \mid a \mid \neg\Sigma' \mid \emptyset \mid E \cdot E' \mid E + E' \mid E\&E' \mid E^* \mid \langle E \rangle \mid \mu a.E$$

where $a \in \Sigma$ and $\Sigma' \subseteq \Sigma$ is finite. We restrict the recursive expressions $\mu a.E$ such that all occurrences of a in E are nested below parentheses. The sets of free and bound symbols $fn(E)$ and $bn(E)$ are defined as usual where $\mu a.E$ binds symbol a with scope E and there is no other binder.

There are two differences with the regular expression types from [19]. First, our NREs treat labels as internal symbols instead of labels of parentheses. Second, they provide recursion through the μ-operator instead of using recursive equation systems. Even though not needed from the view point of expressiveness, we allow conjunctions $E\&E'$ to simplify the compilation of CoreXPath expressions with filters to NREs. NREs having no subexpressions $E\&E'$ are called conjunction-free (CF-NREs). Any NRE describes a language of nested words that is defined by structural induction as follows:

$$\begin{array}{llll} L(\varepsilon) = \{\varepsilon\} & L(a) = \{a\} & L(\neg\Sigma') = \Sigma \setminus \Sigma' & L(\emptyset) = \emptyset \\ L(E \cdot E') = L(E) \cdot L(E') & & L(E^*) = L(E)^* & \\ L(E + E') = L(E) \cup L(E') & & L(E\&E') = L(E) \cap L(E') & \\ L(\langle E \rangle) = \{\langle h \rangle \mid h \in L(E)\} & & L(\mu a.E) = \cup_{n \geq 0} L(\mu^n a.E) & \end{array}$$

A negation $\neg\Sigma'$ stands for $\Sigma \setminus \Sigma'$. This is useful for dealing with infinite alphabets and with large finite alphabets. For all expressions E, E_1 and E_2, the notation $E[E_1/E_2]$ stands for the expression E where all the occurrences of E_1 have been replaced by E_2. The semantics of a μ-operator is then defined using the shortcuts $\mu^0 a.E = E[a/\emptyset]$ and $\mu^n a.E = E[a/\mu^{n-1} a.E]$ for all $n \geq 1$. Note that $\mu a.\ b \cdot a \cdot c + \varepsilon$ would define the string language $\{b^n \cdot c^n \mid n \geq 0\}$ which is not regular. But this expression is ruled out since the μ-bound name a is not nested below parentheses.

In the context of XML queries, we can express the child and descendant-or-self axes of XPATH expressions by using the following NREs:

$$\begin{array}{lll} ch(E) & =_{\text{df}} T \cdot \langle E \rangle \cdot T \qquad T =_{\text{df}} \mu x.\ (\langle x \rangle + \neg\emptyset)^* & \\ ch^*(E) & =_{\text{df}} \mu x.\ (E + ch(x)) & \text{where } x \notin fn(E) \\ ch^+(E) & =_{\text{df}} \mu x.\ (ch(E) + ch(x)) & \text{where } x \notin fn(E) \end{array}$$

Thereby, the XPath expression $a[\mathit{following\text{-}sibling}::b]/\mathit{descendant}::c$ can be expressed as a NRE, in which $x \in \Sigma$ serves as the selection variable, while the negation $\neg\{x\}$ expresses nonselection.

$$\langle \mathit{elem} \cdot a \cdot \neg\{x\} \cdot \mathit{ch}^+(\langle \mathit{elem} \cdot c \cdot x \cdot T\rangle)\rangle \cdot T \cdot \langle \mathit{elem} \cdot b \cdot \neg\{x\} \cdot T\rangle \cdot T$$

Our next objective is to distinguish NREs that can be evaluated deterministically in polynomial time, for instance by compilation to deterministic NWAs. For this, we consider the language of NREs $\mathit{nregexp}(\mathit{ch}, T)$ extended by the constant T and the unary constructor ch.

Definition 1. *An expression of $\mathit{nregexp}(\mathit{ch}, T)$ is deterministic if it does not contain a subexpression of any of the forms: $E_1 + E_2$, E^*, $T \cdot E$, $\mu a.E$.*

Note in particular that $\mathit{ch}(a)$ is a deterministic expression of $\mathit{nregexp}(\mathit{ch}, T)$. In contrast, the semantically equivalent expression $T.\langle a\rangle.T$ is not deterministic. Similarly, T is deterministic while the equivalent expression $\mu x.(\langle x\rangle + \neg\emptyset)^*$ is not. The expression $\mathit{ch}^*(E)$ is not deterministic since its definition relies on the μ-operator.

4 Nested Word Automata

Nested word automata (NWAs) are pushdown automata reading nested words, whose stacks are visible: they push a single stack symbol when reading an opening parenthesis, pop a single stack symbol when reading a closing parenthesis, and don't alter or inspect the stack otherwise.

Definition 2. *An* NWA *is a tuple $A = (Q_h, Q_t, \Sigma, \Gamma, \Delta, I, F)$ consisting of a possibly infinite set Σ of internal symbols, finite sets Q_h and Q_t of states of type hedge and tree respectively, sets of initial and final states $I, F \subseteq Q_h$, a finite set Γ of stack symbols, and a finite set Δ of transition rules of the forms:*

hedge rules	$a^{\Delta}, \_^{\Delta}, \varepsilon^{\Delta} \subseteq Q_h \times Q_h$	*where* $a \in \Sigma$
opening rules	$\langle_\gamma^{\Delta} \subseteq Q_h \times Q_h$	*where* $\gamma \in \Gamma$
hedge ending rules	$\mathit{tree}^{\Delta} \subseteq Q_h \times Q_t$	
closing rules	$\rangle_\gamma^{\Delta} \subseteq Q_t \times Q_h$	

Our NWAs are symbolic, in that they come with else rules, i.e elements of $(q, q') \in \_^{\Delta}$ that we will denote by $q \dashrightarrow q'$, for dealing with large or infinite alphabets. An example for an NWA is given in a graphical syntax in Fig. 1. Tree states are drawn in circles that are filled in light gray (q), while hedge states are in unfilled circles (q). Initial states are drawn as $\rightarrow$(q) and final states as ((q)). Hedge rules that have the form $(q_1, q_2) \in o^{\Delta'}$ where $o \in \Sigma \cup \{\_, \varepsilon, \mathit{tree}\}$ are denoted by $q_1 \xrightarrow{o} q_2$. They are either label, else, epsilon, or tree rules depending of the type of letter o. Opening rules $(q_1, q_2) \in \langle_\gamma^{\Delta}$ are represented as $q_1 \xrightarrow{\langle\downarrow\gamma} q_2$ and closing rules $(q_1, q_2) \in \rangle_\gamma^{\Delta}$ as $q_1 \xrightarrow{\rangle\uparrow\gamma} q_2$.

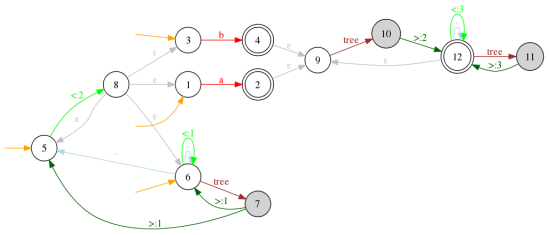


Fig. 1: Nested word automaton $nwa(ch^*(a+b))$.

Our notion of NWAs supports factorization in the spirit of [9]. It is obtained by distinguishing two types of states $q \in Q_h$ and $p \in Q_t$, and adding explicit type coercion rules $q \xrightarrow{tree} p$. Semantically, both kinds of states could be merged when replacing type the coercion rules by the epsilon rule $q \xrightarrow{\varepsilon} p$, but at the cost of introducing additional nondeterminism. This may lead to quadratically larger deterministic automata, as we will illustrate at the NWA in Fig. 5.

The language of nested words between two states $q_1, q_2 \in Q_h$ is defined as the least language such that:

$$\begin{aligned} L_{q_1,q_2}(\Delta) = \{\varepsilon \mid \text{if } q_1 = q_2 \text{ or } q_1 \xrightarrow{\varepsilon} q_2 \text{ wrt.} \Delta\} \cup \textstyle\bigcup_{q_3 \in Q_h} L_{q_1,q_3}(\Delta) \cdot L_{q_3,q_2}(\Delta) \\ \cup \{a \mid \text{if } q_1 \xrightarrow{a} q_2 \in \Delta \text{ or } (q_1 \dashrightarrow q_2 \in \Delta' \text{ and } \neg\exists q_2'.\ q_1 \xrightarrow{a} q_2' \in \Delta)\} \\ \cup \{\langle h \rangle \mid \exists q_1', q_2' \in Q_h . \exists q_3 \in Q_t . \exists \gamma \in \Gamma.\ q_1 \xrightarrow{\langle\downarrow\gamma} q_1', h \in L_{q_1',q_2'}(\Delta), \\ q_2' \xrightarrow{tree} q_3 \in \Delta \text{ and } q_3 \xrightarrow{\rangle\uparrow\gamma} q_2 \in \Delta\}. \end{aligned}$$

The language of the NWA then is $L(A) = \bigcup_{q_1 \in I, q_2 \in F} L_{q_1,q_2}(\Delta)$. NWAs can be determinized by adapting the usual determinization procedure for NWAs [12,1] so that it can account for hedge ending and else rules, as shown in the appendix.

As argued earlier, NREs have the same expressiveness as NWAs and thus as deterministic NWAs. But in practice, the sizes may blow-up surprisingly by determinization [12]. We next discuss on a compiler from expression an E of $nregexp(T, ch)$ to an NWA $nwa(E)$ that preserves determinism. For instance, the NWA for the expression $ch^*(a+b)$ is shown in Fig. 1. For regular expressions without nesting, the compiler is based on Glushkov's construction recursively on the structure of the expression while eliminating ε-edges on the fly. Such construction is known to preserve determinism [7]. For deterministic expressions $ch(E)$, we adapt ideas from [12]. As for conjunctions, product of automata are used. Special care has to be given to $\mu a.E$ expressions in order to preserve the recognized language and also have the following complexity result:

Theorem 1. *For any CF-*NRE *E we can construct in time $O(|E|^2)$ an* NWA *A while preserving determinism such that $L(A) = L(E)$.*

This quadratic time result generalizes on a previous result for the Glushkov construction [6]. Because of automata products used to build them, NREs having conjunctions may in the worst case yield NWAs with an exponential size. As a consequence of Theorem 1 small deterministic CF-NREs can be compiled to

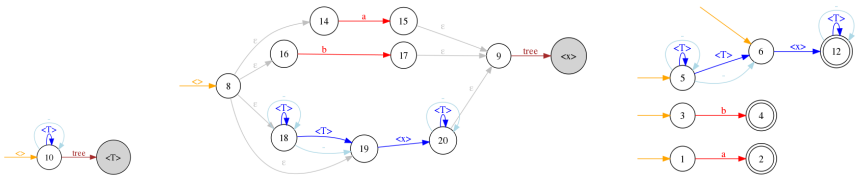


Fig. 2: Stepwise hedge automaton $sha(ch^*(a + b))$: the part with the stepwise tree automaton is on the left and middle, and the NFA part on the right.

small deterministic NWAs. This gives a first positive answer to the motivating question of the present paper.

As for nondeterministic expressions, the NWA determinization procedure is not a solution to the problem at hand, due to huge size increase. For instance, the deterministic NWA $det(nwa(ch^*(a+b)))$ in Fig. 15 has size 271, which may seem way too large. Even worse cases can be found in the experimental section. The problem is not solved by factorization, and actually confirms a size increase reported earlier for NWAs obtained from XPath by a different compiler [12]. So the question is, whether there do not exist better methods to obtain smaller deterministic NWAs for nested regular expressions.

5 Stepwise Hedge Automata

We propose SHAs as an extension of stepwise tree automata [8] to recognize not only unranked trees but also hedges. The problematic notion of determinism of the hedge automata from [20,10,26] is avoided.

Our notion of SHAs will be symbolic in using else rules, and factorized in the sense of [9]: there are two types of states for hedges and trees and an operator for explicit type coercion. We also propose a novel treatment of internal letters inspired by nested word automata, so that SHAs generalize both on stepwise tree automata and on NFAs.

Definition 3. *A SHA is a tuple $S = (Q_h, Q_t, \Sigma, \Delta, I, F)$ such that Q_t and Q_h are finite set of states of two types t for tree and respectively h for hedge, Σ an alphabet of internal letters (that may be infinite), $I, F \subseteq Q_h$ subsets of initial and final states respectively, and Δ a finite set of transition rules such that for all $q \in Q_t$ and $a \in \Sigma$:*

$$\begin{array}{ll} \textit{hedge rules} & q^{\Delta}, a^{\Delta}, \_^{\Delta}, \varepsilon^{\Delta} \subseteq Q_h \times Q_h \\ \textit{tree final rules} & \textit{tree}^{\Delta} \subseteq Q_h \times Q_t \\ \textit{tree initial states} & \langle\rangle^{\Delta} \subseteq Q_h \end{array}$$

An example for a SHA is given in graphical syntax in Fig. 2. It recognizes all hedges which are either just a or b or contain some tree node that contains either just a or b. In the graphical syntax, the states of type tree $q \in Q_t$ are drawn in circles filled in light gray (q), while the states of type hedge $q' \in Q_h$ are drawn in unfilled circles (q'). The right part of the graph is an NFA which

uses tree states as additional edge labels, while the left part is a stepwise tree automaton, that defines the tree languages of these tree states.

Let Δ_h be the restriction of Δ to the hedge rules. Then, $(Q_h, \Sigma \uplus Q_t, \Delta_h, I, F)$ is a standard NFA with ε-rules, which is symbolic [11] in providing else rules for dealing with large or infinite alphabets in addition. Therefore, we denote the initial states $q \in I$ by $\rightarrow(q)$ and the final states $q \in F$ by $((q))$. A rule with an internal letter $(q_1, q_2) \in a^\Delta$ is denoted by $q_1 \xrightarrow{a} q_2$ wrt Δ stating that a hedge in state q_1 can be extended by the internal letter a leading to a hedge in state q_2. Similarly, an epsilon rule $(q_1, q_2) \in \varepsilon^\Delta$ is denoted by $q_1 \xrightarrow{\varepsilon} q_2$, and an else rule $(q_1, q_2) \in \_^\Delta$ is denoted by $q_1 \dashrightarrow q_2$. In the same spirit, a hedge rule $(q_1, q_2) \in q^\Delta$ is denoted by $q_1 \xrightarrow{q} q_2$ wrt. Δ, stating that a hedge in state q_1 can be extended by a tree in state q leading to a hedge in state q_2.

A tree initial state $q \in \langle\rangle^\Delta$ is graphically denoted by $\xrightarrow{\langle\rangle} q$ and a tree final rule $(q_1, q_2) \in \mathit{tree}^\Delta$ by $q_1 \xrightarrow{\mathit{tree}} q_2$. Intuitively, a tree $\langle h \rangle$ can be evaluated to state q if h can be evaluated starting with some tree initial state $\xrightarrow{\langle\rangle} q_1$ to some state q_2 such that $q_2 \xrightarrow{\mathit{tree}} q$. More formally, the hedge languages $L_{q_1,q_2}(S)$ between any two hedge states $q_1, q_2 \in Q_h$ are defined as follows:

$$\begin{aligned} L_{q_1,q_2}(S) = \{\varepsilon \mid \text{if } q_1 = q_2 \text{ or } q_1 \xrightarrow{\varepsilon} q_2 \text{ wrt. } \Delta\} \cup \textstyle\bigcup_{q_3 \in Q_h} L_{q_1,q_3}(S) \cdot L_{q_3,q_2}(S) \\ \cup \{a \mid \text{if } q_1 \xrightarrow{a} q_2 \in \Delta \text{ or } (q_1 \dashrightarrow q_2 \in \Delta \text{ and } \neg\exists q_2'.\ q_1 \xrightarrow{a} q_2' \in \Delta)\} \\ \cup \textstyle\bigcup_{q_1 \xrightarrow{q} q_2 \in \Delta} L_q(S) \end{aligned}$$

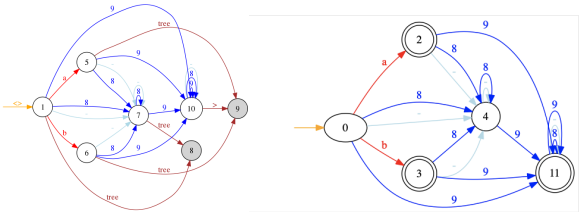
This definition is mutually recursive with the definition of the tree languages $L_q(S)$ of all tree states $q \in Q_t$:

$$L_q(S) = \{\langle h \rangle \mid \xrightarrow{\langle\rangle} q_1,\ h \in L_{q_1,q_2}(S),\ q_2 \xrightarrow{\mathit{tree}} q\}$$

The hedge language $L(S)$ that is recognized by automaton is $\bigcup_{q_1 \in I, q_2 \in F} L_{q_1,q_2}(S)$. The rules of standard bottom-up tree automata have the form $a(q_1, \ldots, q_n) \to q$ where a is a symbol of arity n. With SHAs, this rule can be encoded by the sequence $\xrightarrow{\langle\rangle} (p_0) \xrightarrow{a} (p_1) \xrightarrow{q_1} \ldots \xrightarrow{q_n} (p_n) \xrightarrow{\mathit{tree}} q$ where the states $q_1, \ldots, q_n, q$ are all tree states, and $p_0, \ldots, p_n$ fresh hedge states. Stepwise hedge automata have a natural notion of determinism, generalizing both on that of stepwise tree automata and on NFAs, in contrast to the earlier notion of hedge automata in [10,26], as presented in the appendix. For instance, the SHA in Fig. 3 is obtained by determinization of the automaton in Fig. 2. It consists of a DFA on the right and a deterministic stepwise tree automaton on the left. We show that

Proposition 1. *Any SHA can be made deterministic in at most exponential time such that the hedge language is preserved.*

Any expression E can be compiled to a SHA $\mathit{sha}(E) = (Q_h, Q_t, \Sigma, \Delta, I, F)$ such that $Q_t = \{E' \mid E' = \langle E'' \rangle \text{ subexpression of } E\}$ and $L_t(E') = L(E')$ for all tree states $E' \in Q_t$. The SHA $\mathit{sha}(E)$ can be partitioned into disjoint SHAs $\mathit{sha}(E) = A^{top} \cup \bigcup_{E' \in Q_t} A^{E'}$ such that $A^{top} = (Q_h^{top}, Q_t, \Sigma, \Delta^{top}, I, F)$ and $A^{E'} = (Q_h^{E'}, Q_t, \Sigma, \Delta^{E'}, \emptyset, \emptyset)$ for all $E' \in Q_t$ and $\langle^{\Delta^{top}} = \emptyset$.



$0 = \{1, 3, 5, 6\}$
$1 = \{8, 10, 14, 16, 18, 19\}$
$2 = \{2, 5, 6\}$
$3 = \{4, 5, 6\}$
$4 = \{5, 6\}$
$5 = \{9, 10, 15, 18, 19\}$
$6 = \{9, 10, 17, 18, 19\}$
$7 = \{10, 18, 19\}$
$8 = \{\langle T \rangle\}$
$9 = \{\langle T \rangle, \langle x \rangle\}$
$10 = \{9, 10, 18, 19, 20\}$
$11 = \{5, 6, 12\}$

Fig. 3: The determinized SHA $det(sha(ch^*(a + b)))$.

Proposition 2. *For any CF-*NRE E *we can construct in time* $O(|E|^2)$ *a SHA* $sha(E)$ *such that* $L(sha(E)) = L(E)$.

However, the construction does not preserve determinism. For the deterministic NRE $\langle a_1 \cdot \langle a_2 \cdot \ldots \cdot \langle a_n \rangle \ldots \rangle \rangle$, one would have an SHA having a tree initial state for each of the $\langle a_i \ldots \rangle$ subtree, implying nondeterminism. This is in contrast to the compiler to NWAs, which can rely on top-down determinism that is unavailable for SHAs though. Furthermore, as for NWAs, conjunctions may cause an exponential blow-up of the produced SHA.

6 NWAs versus SHAs

We next show how to compile SHAs to NWAs such that determinism is preserved, and back while introducing nondeterminism. Thereby we can obtain small NWAs for NREs such as $E = ch^*(a+b)$ for which $det(nwa(E))$ blows up in size in a surprising manner.

SHAs to NWAs. Any SHA $S = (Q_h, Q_t, \Sigma, \Delta, I, F)$ can be compiled to an NWA $nwa(S) = (Q_h, Q_t, \Sigma, \Gamma, \Delta', I, F)$ such that $L_{q_1,q_2}(S) = L_{q_1,q_2}(nwa(S))$. We set $\Gamma = Q_h$, $\_^{\Delta'} = \_^{\Delta}$, $a^{\Delta'} = a^{\Delta}$ for all $a \in \Sigma$, $\varepsilon^{\Delta'} = \varepsilon^{\Delta}$, $tree^{\Delta'} = tree^{\Delta}$:

$$\frac{q_1 \xrightarrow{q} q_2 \in \Delta \quad p \in \langle\rangle^{\Delta}}{q_1 \xrightarrow{\langle\downarrow q_1} p \in \Delta' \text{ and } q \xrightarrow{\rangle\uparrow q_1} q_2 \in \Delta'}$$

Clearly, if S is deterministic then is $nwa(S)$, since p is unique in this case in particular. One might be tempted to restrict the above construction rule to states p such that $L_q(S[\langle\rangle^{\Delta}/\{p\}]) \neq \emptyset$ where the set of tree initial states $\langle\rangle^{\Delta}$ is replaced by $\{p\}$. However, this would lead to huge blow-up when determinizing these NWAs, basically since this change spoils the single-entry property discussed in Definition 4.

The conversion of $sha(ch^*(a + b))$ in Fig. 2 yields the NWA in Fig. 4. Note that the opening rules are deterministic (but not the whole NWA), since for all tree states q there is at most one hedge state p with $\langle\rangle \to p$ such that q is accessible from p. The NWA has size 64, while its determinization has size 159 (see Fig. 17 of the appendix). The size increase raised by determinization is thus $95 = 159 - 64$ for this NWA.

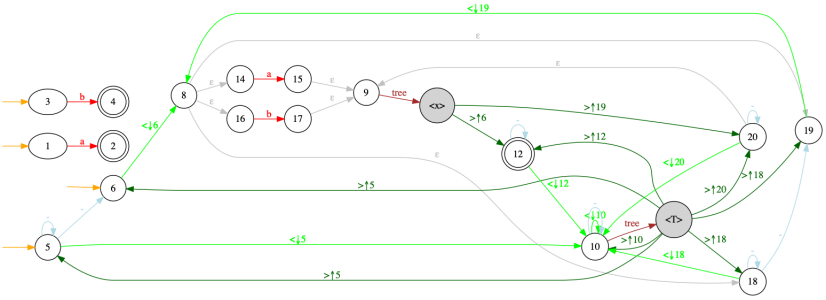


Fig. 4: The NWA from the SHA $nwa(sha(ch^*(a+b)))$.

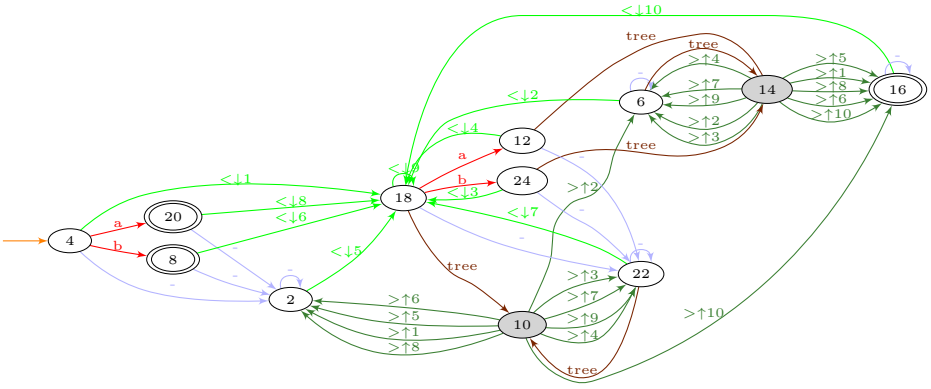


Fig. 5: Deterministic NWA: $nwa(det(sha(ch^*(a+b))))$.

The size increase for determinization is considerably smaller for the NWA obtained from the regular expressions by indirection via a stepwise hedge automaton, than for NWAs obtained by direct compilation. Indeed, the determinization of $nwa(ch^*(a+b))$ blows the size from 39 to 271. The size increase for the determinization of $nwa(ch^*(a+b))$ is thus $242 = 271 - 39$ and while for $nwa(sha(ch^*(a+b)))$ is is only $95 = 159 - 64$.

The experiments will show that this is not an exception but the general rule. Intuitively, the reason is that NWAs obtained from stepwise hedge automata do all work bottom-up, where NWAs obtained directly from the regular expression do a considerable amount of work top-down. In terms of [2] this restriction can be characterized syntactically by the single-entry property:

Definition 4. *An* NWA A *has the single-entry property, if there exists a single state* $q_{entry} \in Q_h$ *such that all opening rules of* A *have the form* $q \xrightarrow{\langle\downarrow q} q_{entry}$.

It can be shown that $nwa(S)$ has the single-entry property for all SHAs S for which the p's are unique in the above construction rule, i.e. such that $\langle\rangle \to p$. Note that this wasn't the case for $sha(ch*(a+b))$ in Fig. 2 but could have been imposed wlog leading to a slightly different NWA than in Fig. 4.

$$\frac{o \in \Sigma \cup \{\mathit{tree}, \_, \varepsilon\} \quad q_1 \xrightarrow{o} q_2 \in \Delta \quad q \in Q_h}{(q, q_1) \xrightarrow{o} (q, q_2) \in \Delta^s} \qquad \frac{q_1 \xrightarrow{\langle\downarrow\gamma} q_2 \in \Delta}{\xrightarrow{\langle\rangle} (q_2, q_2) \in \Delta^s}$$

$$\frac{q_1 \xrightarrow{\langle\downarrow\gamma} q_2 \in \Delta \quad q_3 \in Q_t \quad q_3 \xrightarrow{\rangle\uparrow\gamma} q_4 \in \Delta \quad q \in Q_h}{(q, q_1) \xrightarrow{(q_2,q_3)} (q, q_4) \in \Delta^s}$$

Fig. 6: NWA to stepwise conversion.

The conversion of the determinization $\mathit{det}(\mathit{sha}(\mathit{ch}^*(a+b)))$ in Fig. 3 yields the deterministic NWA in Fig. 5. The size goes up slightly from 53 to 73. It should be noticed, that factorization avoids a quadratic blow up in this case. This can be observed at state 14, which has 3 incoming tree-edges and 10 outgoing closing edges. Without factorization, the 3 tree edges could be replaced by 3 ε-edges whose elimination would produce 30 closing edges. This would increase the number $3 + 10$ edges to $3 * 10$ edges.

NWAs to SHAs. Conversely, NWAs can be compiled to stepwise hedge automata, but at the cost of introducing nondeterminism, since an NWA may traverse the branches of a tree top-down, while a stepwise must traverse them bottom-up. For this, the stepwise guesses the state in which the NWA will arrive from above and then evaluates the subtree starting with this state, while verifying the correctness of the guess later on. Let $A = (Q_h, Q_t, \Sigma, \Delta', I, F)$ be an NWA. We build a SHA $\mathit{sha}(A) = (Q_h^s, Q_t^s, \Sigma, \Delta^s, I^s, F^s)$ where $Q_h^s = Q_h \times Q_h$, $Q_t^s = Q_h \times Q_t$, $I^s = \{(q, q) \mid q \in I\}$, $F^s = I \times F$ and Δ^s is the smallest satisfying the rule schemas in Fig. 6. The construction is such that $L(A) = L(\mathit{sha}(A))$.

For the NWA $\mathit{nwa}(\mathit{ch}^*(a+b))$ in Fig. 1 we obtain the stepwise in Fig. 16 up-to removing useless states and separating the top-level. Determinization yields $\mathit{det}(\mathit{sha}(\mathit{nwa}(\mathit{ch}^*(a+b)) = \mathit{det}(\mathit{sha}(\mathit{ch}^*(a+b)))$ in Fig. 3.

7 Experimental Results and Discussion

We now compare the sizes of deterministic NWAs that we can obtain by composing the various compilers in different orders.

We test the $A1, \ldots, A8$ XPath queries in the usual XPath benchmark [14], which contain not only forward child, descendant and following-sibling axes, but also filters and path compositions. Note that the queries A4 until A8 contain filters, which are mapped to NREs with conjunctions. We compiled these queries automatically to nested regular expressions, then compiled these expressions to deterministic NWAs, by composing the various compilers presented earlier in all reasonable manners. A1 is the only query for which we obtain a deterministic regular expression. But since we replaced $\mathit{ch}(E)$ systematically by $T \cdot \langle E \rangle \cdot T$ in our experiments, all nested regular expression become nondeterministic.

The overall size of the resulting automata and the number of their rules are given in Fig. 7. We can see that determinization applied to the NWAs for these expressions fails. Only 2 out of 8 automata have a size less than 400000, and for the others, the determinization run out of time. In contrast, 3 of the 4 other methods – that use stepwise hedge automaton intermediately – produce reasonable small deterministic NWAs. For the fourth method in the last

	$det(nwa(.))$	$nwa(det($ $sha(.)))$	$det(nwa($ $sha(.)))$	$nwa(det($ $sha(nwa(.))))$	$det(nwa($ $sha(nwa(.))))$
A1		398 (37)	302 (62)	398 (37)	398 (37)
A2	362600 (6782)	668 (57)	4889 (221)	1648 (127)	4105 (148)
A3	318704 (8216)	469 (44)	542 (66)	625 (56)	907 (62)
A4		487 (42)	335 (67)	487 (42)	487 (42)
A5		676 (55)	1054 (110)	856 (67)	1192 (73)
A6		548 (45)	332 (62)	548 (45)	548 (45)
A7		468 (41)	285 (54)	468 (41)	468 (41)
A8		2520 (124)	1236 (137)	1804 (118)	

Fig. 7: Deterministic NWAs for XPath benchmark: size (#states).

	$det(nwa(.))$	$nwa(det($ $sha(.)))$	$det(nwa($ $sha(.)))$	$nwa(det($ $sha(nwa(.)))$	$det(nwa($ $sha(nwa(.))))$
$ch^3[a]$	19828 (1281)	85 (13)	157 (30)	192 (24)	352 (32)
$ch^4[a]$		177 (21)	206 (39)	664 (56)	2200 (88)
$ch^5[a]$		457 (37)	255 (48)	3336 (168)	
$ch^7[a]$		4825 (133)	353 (66)		
$ch^9[a]$			451 (84)		

Fig. 8: Deterministic NWAs queries $ch^n[a]$ for $n = 3, 4, 5, 7, 9$: size (#states).

column, NWA-determinization didn't terminate on $nwa(sha(nwa(A8)))$ after a few hours.

We also tested our algorithms on collections of XPath queries with a scalable parameter, such as the queries $ch^n(a)$ for increasing n. This series is known to require many states for deterministic bottom-up evaluation. Indeed, the determinization for stepwise hedge automata $nwa(det(sha))$ leads to a size explosion. The method $det(nwa(sha(.)))$, however, still yields small deterministic automata! Generally this method produced satisfactory results in all our experiments. In quite some cases, however, $nwa(det(sha(.)))$ still behaves better.

We conjecture that these differences are related to the lack of minimization in our current implementation. The main problem here is that minimal deterministic NWAs do not exist for all regular languages of nested words [2]. This is in strict contrast to the cases of word automata, tree automata, and SHAs.

We point out that SHAs are determinized in bottom-up and left-to-right manner by combining the usual bottom-up determinization algorithms for tree automata and the usual left-to-right determinization algorithm for NFAs. In contrast to deterministic NWAs, they cannot support top-down determinism in combination with bottom-up and left-to-right determinism though. The NWAs obtained by compilation from SHAs are special in that they perform all their work in a bottom-up and left-to-right manner, and nothing top-down. Such NWAs were characterized syntactically as single-entry NWAs, and deterministic single-entry NWAs are shown to admit a unique minimization in [2]. Our experiments show that NWA determinization often works nicely for single-entry NWAs, while it explodes quickly without the single-entry restriction. The intuition is that single-entry NWAs behave like SHAs.

References

1. Alur, R.: Marrying words and trees. In: 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 233–242. ACM-Press (2007), http://dx.doi.org/10.1145/1265530.1265564
2. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Automata, Languages and Programming, 32nd International Colloquium. Lecture Notes in Computer Science, vol. 3580, pp. 1102–1114. Springer Verlag (2005). https://doi.org/10.1007/11523468_89, http://dx.doi.org/10.1007/11523468_89
3. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th ACM Symposium on Theory of Computing. pp. 202–211. ACM-Press (2004), http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=1007390
4. Benedikt, M., Jeffrey, A., Ley-Wild, R.: Stream Firewalling of XML Constraints. In: ACM SIGMOD International Conference on Management of Data. pp. 487–498. ACM-Press (2008)
5. Bozzelli, L., Sánchez, C.: Visibly rational expressions. Acta Inf. **51**(1), 25–49 (Feb 2014). https://doi.org/10.1007/s00236-013-0190-6, http://dx.doi.org/10.1007/s00236-013-0190-6
6. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoretical Computer Science **120**(2), 197–213 (Nov 1993). https://doi.org/10.1016/0304-3975(93)90287-4, http://dx.doi.org/10.1016/0304-3975(93)90287-4
7. Brüggemann-Klein, A., Wood, D.: One-Unambiguous regular languages. Information and Computation **142**(2), 182–206 (May 1998)
8. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: 19th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 3091, pp. 105–118. Springer Verlag (2004), http://www.ps.uni-sb.de/Papers/abstracts/stepwise.html
9. Champavère, J., Gilleron, R., Lemay, A., Niehren, J.: Efficient inclusion checking for deterministic tree automata and XML schemas. Information and Computation **207**(11), 1181–1208 (2009). https://doi.org/10.1016/j.ic.2009.03.003, http://dx.doi.org/10.1016/j.ic.2009.03.003
10. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available online since 1997: http://tata.gforge.inria.fr (Oct 2007)
11. D'Antoni, L., Alur, R.: Symbolic visibly pushdown automata. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 209–225. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_14, https://doi.org/10.1007/978-3-319-08867-9_14
12. 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). https://doi.org/10.1016/j.tcs.2015.01.017, http://dx.doi.org/10.1016/j.tcs.2015.01.017
13. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**(2), 194–211 (1979). https://doi.org/10.1016/0022-0000(79)90046-1, https://doi.org/10.1016/0022-0000(79)90046-1
14. Franceschet, M.: Xpathmark performance test. https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html, accessed: 2020-03-27
15. Gauwin, O., Niehren, J.: Streamable fragments of forward XPath. In: Markhoff, B.B., Caron, P., Champarnaud, J.M., Maurel, D. (eds.) International Conference

on Implementation and Application of Automata. Lecture Notes in Computer Science, vol. 6807, pp. 3–15. Springer (2011). https://doi.org/10.1007/978-3-642-22256-6_2, http://dx.doi.org/10.1007/978-3-642-22256-6_2

16. Gauwin, O., Niehren, J., Tison, S.: Earliest query answering for deterministic nested word automata. In: 17th International Symposium on Fundamentals of Computer Theory. Lecture Notes in Computer Science, vol. 5699, pp. 121–132. Springer Verlag (2009), http://hal.inria.fr/inria-00390236/en
17. Gottlob, G., Koch, C., Pichler, R.: The complexity of XPath query evaluation. In: 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 179–190 (2003)
18. Grez, A., Riveros, C., Ugarte, M.: A formal framework for complex event processing. In: Barceló, P., Calautti, M. (eds.) 22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal. LIPIcs, vol. 127, pp. 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2019). https://doi.org/10.4230/LIPIcs.ICDT.2019.5, https://doi.org/10.4230/LIPIcs.ICDT.2019.5
19. Hosoya, H., Pierce, B.C.: Xduce: A statically typed XML processing language. ACM Trans. Internet Techn. **3**(2), 117–148 (2003). https://doi.org/10.1145/767193.767195, https://doi.org/10.1145/767193.767195
20. Martens, W., Niehren, J.: On the Minimization of XML-Schemas and Tree Automata for Unranked Trees. Journal of Computer and System Science **73**(4), 550–583 (2007). https://doi.org/10.1016/j.jcss.2006.10.021, https://hal.inria.fr/inria-00088406, special issue of DBPL 05
21. Mozafari, B., Zeng, K., Zaniolo, C.: From regular expressions to nested words: Unifying languages and query execution for relational and XML sequences. PVLDB **3**(1), 150–161 (2010). https://doi.org/10.14778/1920841.1920865, http://www.vldb.org/pvldb/vldb2010/pvldb_vol3/R13.pdf
22. Mozafari, B., Zeng, K., Zaniolo, C.: High-performance complex event processing over XML streams. In: Candan, K.S., Chen, Y., Snodgrass, R.T., Gravano, L., Fuxman, A., Candan, K.S., Chen, Y., Snodgrass, R.T., Gravano, L., Fuxman, A. (eds.) SIGMOD Conference. pp. 253–264. ACM (2012). https://doi.org/10.1145/2213836.2213866, http://dx.doi.org/10.1145/2213836.2213866
23. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. In: Foundations of Software Technology and Theoretical Computer Science. Lecture Notes in Computer Science, vol. 1530, pp. 134–145. Springer Verlag (1998)
24. Olteanu, D.: SPEX: Streamed and progressive evaluation of XPath. IEEE Trans. on Know. Data Eng. **19**(7), 934–949 (2007). https://doi.org/10.1109/TKDE.2007.1063, http://dx.doi.org/10.1109/TKDE.2007.1063
25. Pitcher, C.: Visibly pushdown expression effects for xml stream processing. In: PlanX (2005)
26. Thatcher, J.W.: Characterizing derivation trees of context-free grammars through a generalization of automata theory. Journal of Computer and System Science **1**, 317–322 (1967)

A XPath Benchmark

We consider the navigational XPath queries A1-A8 from the XPath benchmark with forward axes child and descendant axes, path compositions, and filters. These are recalled in Fig. 9 We encoded these XPath queries semi-automatically to nested regular expressions, some of which are given in Fig. 10. Without filters as in A1 and A2 this is can be done automatically. In the case of nonoverlapping filters as in A4, we test them in all possible orders.

B NWAs for μ-Expressions

For illustration, we consider the NRE $E = \mu a.\langle a^* \rangle$. The reader should be warned that constructing an NWA for E is less trivial than it might seem at first sight. One has to start from the NWA for $\langle a^* \rangle$) which is given in Fig. 11. Simply adding epsilon edges to capture the operator μa will not work though. It will lead to the wrong automaton in Fig. 12. This automaton will wrong accept the hedge $\langle \rangle \langle \rangle$, since this hedge does not belong to $L(E)$.

Fig. 13 illustrates the product automaton for E obtained by our construction, where only accessible states are kept.

C Determinization of NWAs

We adapt the usual determinization procedure for NWAs [12,1] so that they can account for hedge ending and else rules.

Definition 5. *An* NWA *is* deterministic *if I is a singleton or empty, ε^Δ is empty, for all $a \in \Sigma$ a^Δ and $\_^\Delta$ are partial functions from Q_h to Q_h, for all $q \in Q_h$ and $\gamma \in \Gamma$ there exists a most one $q' \in Q_h$ such that $q' \in \langle_\gamma^\Delta$, and for all $\gamma \in \Gamma$, $\rangle_\gamma^\Delta$ is a partial function from Q_h to Q_t.*

Given an NWA $A = (Q_h, Q_t, \Sigma, \Gamma, \Delta, I, F)$, the difficulty is to deal with concurrent opening rules $q \xrightarrow{\langle\downarrow\gamma_1} q_1$ and $q \xrightarrow{\langle\downarrow\gamma_2} q_2$ in Δ during determinization without mixing up the stack symbols γ_1 and γ_2. Therefore, we use transition relations as states of the determinized automaton $\det(A) = (Q_h^{det}, Q_t^{det}, \Sigma, \Gamma^{det}, \Delta^{det}, I^{det}, F^{det})$, that is $Q_h^{det} = 2^{Q_h \times Q_h}$, $Q_t^{det} = 2^{Q_h \times Q_t}$. The only initial state is the transition id_I which relates all initial states of A to themselves, i.e., $I^{det} = \{id_I\}$. The set of final states is $F^{det} = \{\tau \in Q_h^{det} \mid \tau \cap (I \times F) \neq \emptyset\}$. Schemas generating the transition rules in Δ^{det} are given in Fig. 14. For a transition $\tau \in Q \times Q$, we write $lab(\tau) = \{a \in \Sigma \mid \exists (q, q') \in \tau, q'' \in Q.q' \xrightarrow{a} q''$ wrt. $\Delta\}$. These schemas for generate transition rules for the accessible transitions only.

The determinization of $det(nwa(ch^*(a + b)))$ is given in Fig. 15 of the appendix: it has size 271 while the nondeterministic version has size 39. Even worse, we will see in Section 7, that the size increase is really too bad in many practical cases of the XPath benchmark suite. Unfortunately, this problem is not solved by adding factorization and restricting the compiler to accessible transitions. Indeed, the problem of such a huge size increase was also reported earlier for

NWAs obtained from XPath by a different compiler [12] via a logical intermediate language. So the question is, whether there do not exist better compilation methods to obtain smaller deterministic NWAs for nested regular expressions.

D Determinization of SHAs

We formalize the notion of determinism for stepwise hedge automata and show that how determinization works.

Definition 6. *A SHA is* deterministic *if* $\langle\rangle^{\Delta}$ *and* I *are both singletons or empty,* ε^{Δ} *is empty, for all* $a \in \Sigma$ *and* $q \in Q_t$, $a^{\Delta}, q^{\Delta}, \_^{\Delta}$ *are partial functions from* Q_h *to* Q_h, *and tree*$^{\Delta}$ *is a partial function from* Q_h *to* Q_t.

Proposition 1 A SHA can be made deterministic in at most exponential time such that the hedge language is preserved.

Proof. In a first step we eliminate ε-rules as usual for NFAs in cubic time. Given a stepwise hedge automaton $S = (Q_h, Q_t, \Sigma, \Delta, I, F)$ without ε-rules, we define an equivalent deterministic stepwise hedge automaton $det(A) = (Q_h^{det}, Q_t^{det}, \Sigma, \Delta^{det}, I^{det}, F^{det})$ such that $Q_h^{det} = 2^{Q_h}$, $Q_t^{det} = 2^{Q_t}$, $I^{det} = \{I\}$ and $F^{det} = \{Q' \subseteq Q_h \mid Q' \cap F \neq \emptyset\}$. There is a unique tree initial state in $\langle^{\Delta^{det}} = \{\langle^{\Delta}\}$ and no ε-rule in $\varepsilon^{\Delta^{det}} = \emptyset$. Furthermore, for all $Q_1, Q_2 \subseteq Q_h$, $a \in lab(Q_1)$, and $Q' \subseteq Q_t$:

$$
\begin{array}{l}
Q_1 \xrightarrow{Q'} Q_2 \text{ wrt. } \Delta^{det} \text{ iff } Q_2 = \{q_2 \mid \exists q_1 \in Q_1, q \in Q'.\ q_1 \xrightarrow{q'} q_2 \text{ wrt } \Delta\} \\
Q_1 \xrightarrow{a} Q_2 \text{ wrt. } \Delta^{det} \text{ iff } Q_2 = \{q_2 \mid \exists q_1 \in Q_1,\ q_1 \xrightarrow{a} q_2 \text{ wrt } \Delta\} \cup \\
\qquad\qquad \{q_2 \mid \exists q_1 \in Q_1.\ q_1 \dashrightarrow q_2 \text{ wrt. } \Delta \text{ and } \nexists q_3 \in Q. q_1 \xrightarrow{a} q_3 \text{ wrt. } \Delta\} \\
Q_1 \xrightarrow{\rangle} Q_2 \text{ wrt. } \Delta^{det} \text{ iff } Q_2 = \{q_2 \mid \exists q_1 \in Q_1,\ q_1 \xrightarrow{\rangle} q_2 \text{ wrt } \Delta\} \\
Q_1 \dashrightarrow Q_2 \text{ wrt. } \Delta^{det} \text{ iff } Q_2 = \{q_2 \mid \exists q_1 \in Q_1,\ q_1 \dashrightarrow q_2 \text{ wrt } \Delta\}
\end{array}
$$

We can show for all $Q_1, Q_2 \subseteq Q_h$ and $Q' \subseteq Q_t$ that $L_{Q_1,Q_2}(\det(S)) = \bigcup_{q_1 \in Q_1, q_2 \in Q_2} L_{q_1,q_2}(S)$ and that $L_{Q'}(\det(S)) = \bigcup_{q' \in Q'} L_{q'}(S)$. Hence $L(det(S)) = \bigcup_{Q' \in F^{det}} L_{I,Q'}(det(S))$ and thus $L(det(S)) = \bigcup_{q_1 \in I} \bigcup_{q_2 \in F} L_{q_1,q_2}(S) = L(S)$.

A1: `/site/closed_auctions/closed_auction/annotation/description/text/keyword`
A2: `//closed_auction//keyword`
A3: `/site/closed_auctions/closed_auction//keyword`
A4: `/site/closed_auctions/closed_auction[annotation/description/text/keyword]/date`
A5: `/site/closed_auctions/closed_auction[descendant::keyword]/date`
A6: `/site/people/person[profile/gender and profile/age]/name`
A7: `/site/people/person[phone or homepage]/name`
A8: `/site/people/person[address and (phone or homepage) and (creditcard or profile)]/name`

Fig. 9: XPath benchmark queries.

A1 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot \langle elem \cdot closed\_auctions \cdot \_ \cdot T \cdot$
$\langle elem \cdot closed\_auction \cdot \_ \cdot T \cdot \langle elem \cdot annotation \cdot \_ \cdot T \cdot$
$\langle elem \cdot description \cdot \_ \cdot T \cdot \langle elem \cdot text \cdot \_ \cdot T \cdot$
$\langle elem \cdot keyword \cdot x \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle$
A2 $\langle doc \cdot \_ \cdot (\mu d_1.(\langle elem \cdot closed\_auction \cdot \_ \cdot T \cdot (\mu d_2.($
$\langle elem \cdot keyword \cdot x \cdot T\rangle \cdot T + \langle T \cdot d_1 \cdot T\rangle \cdot T))\rangle \cdot T + \langle T \cdot d_1 \cdot T\rangle \cdot T))\rangle$
A3 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot \langle elem \cdot closed\_auctions \cdot \_ \cdot T \cdot \langle elem \cdot closed\_auction \cdot \_ \cdot T \cdot ($
$\mu d.(\langle elem \cdot keyword \cdot x \cdot T\rangle \cdot T + \langle T \cdot d \cdot T\rangle \cdot T))\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle$
A4 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot$
$\langle elem \cdot closed\_auctions \cdot \_ \cdot T \cdot \langle elem \cdot closed\_auction \cdot \_ \cdot$
$(T \cdot \langle elem \cdot annotation \cdot \_ \cdot T \cdot \langle elem \cdot description \cdot \_ \cdot T \cdot$
$\langle elem \cdot text \cdot \_ \cdot T \cdot \langle elem \cdot keyword \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T)$
$\&\ (T \cdot \langle elem \cdot date \cdot x \cdot T\rangle \cdot T\rangle \cdot T)\rangle \cdot T\rangle \cdot T\rangle$
A5 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot \langle elem \cdot closed\_auctions \cdot \_ \cdot T \cdot ($
$\langle elem \cdot closed\_auction \cdot \_ \cdot T \cdot ($
$\mu d.(\langle elem \cdot keyword \cdot \_ \cdot T\rangle \cdot T + \langle T \cdot d \cdot T\rangle \cdot T))\rangle \cdot T)$
$\&(\langle elem \cdot closed\_auction \cdot \_ \cdot T \cdot \langle elem \cdot date \cdot x \cdot T\rangle \cdot T\rangle \cdot T)\rangle \cdot T\rangle \cdot T\rangle$
A6 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot \langle elem \cdot people \cdot \_ \cdot T \cdot ((\langle elem \cdot person \cdot \_ \cdot T \cdot$
$\langle elem \cdot profile \cdot \_ \cdot T \cdot \langle elem \cdot gender \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T)$
$\&(\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot profile \cdot \_ \cdot T \cdot \langle elem \cdot age \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T\rangle \cdot T))$
$\&(\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot name \cdot x \cdot T\rangle \cdot T\rangle \cdot T)\rangle \cdot T\rangle \cdot T\rangle$
A7 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot \langle elem \cdot people \cdot \_ \cdot T \cdot ((\langle elem \cdot person \cdot \_ \cdot T \cdot$
$\langle elem \cdot phone \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T+$
$\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot homepage \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T))$
$\&(\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot name \cdot x \cdot T\rangle \cdot T\rangle \cdot T)\rangle \cdot T\rangle \cdot T\rangle$
A8 $\langle doc \cdot \_ \cdot \langle elem \cdot site \cdot \_ \cdot T \cdot \langle elem \cdot people \cdot \_ \cdot T \cdot$
$((((\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot address \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T)$
$\&\ (\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot phone \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T +$
$\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot homepage \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T))$
$\&\ ((\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot creditcard \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T +$
$\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot profile \cdot \_ \cdot T\rangle \cdot T\rangle \cdot T)))$
$\&\ (\langle elem \cdot person \cdot \_ \cdot T \cdot \langle elem \cdot name \cdot x \cdot T\rangle \cdot T\rangle \cdot T)\rangle \cdot T\rangle \cdot T\rangle$

Fig. 10: The NREs of the XPath benchmark queries. Here $\_$ is the NRE that accepts any internal letter, that is $\neg\{\}$. The schema restricts this letter to be either equal to the variable x or its negation $\neg x$. In most case it must be $\neg x$ since x will be read elsewhere, and may occur at most once.

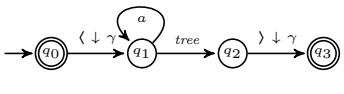


Fig. 11: Automaton for the $\langle a^* \rangle$ expression.

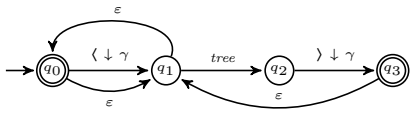


Fig. 12: Bad automaton for $\mu a.\langle a^* \rangle$

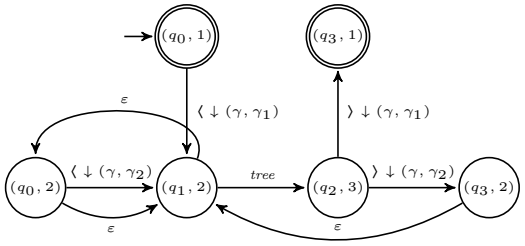


Fig. 13: Good automaton for $\mu a.\langle a^* \rangle$

$$\frac{\tau \in Q_h^{det}}{\tau \xrightarrow{\varepsilon} \tau \circ \_^{\Delta} \in \Delta^{det}} \qquad \frac{\tau \in Q_h^{det} \quad Q' = \{q' \mid \exists (\_, q) \in \tau.\ q \xrightarrow{\langle \downarrow \gamma} q' \in \Delta\}}{\tau \xrightarrow{\langle \downarrow \tau} id_{Q'} \in \Delta^{det}}$$

$$\frac{\tau \in Q_h^{det}}{\tau \xrightarrow{tree} \tau \circ tree^{\Delta} \in \Delta^{det}} \qquad \frac{\tau \in Q_t^{det} \quad \langle \tau \rangle^{\Delta} = \cup_{\gamma \in \Gamma} \langle_{\gamma}^{\Delta} \circ \tau \circ \rangle_{\gamma}^{\Delta}}{\tau \xrightarrow{\rangle \uparrow \tau'} \tau' \circ \langle \tau \rangle^{\Delta} \in \Delta^{det}}$$

$$\frac{\tau \in Q_h^{det} \quad a \in lab(\tau) \quad \tau' = \{(q, q') \in \_^{\Delta} \mid \nexists q''.q \xrightarrow{a} q'' \text{ wrt} \Delta\}}{\tau \xrightarrow{a} \tau \circ (a^{\Delta} \cup \tau') \in \Delta^{det}}$$

Fig. 14: Determinization of NWAs.

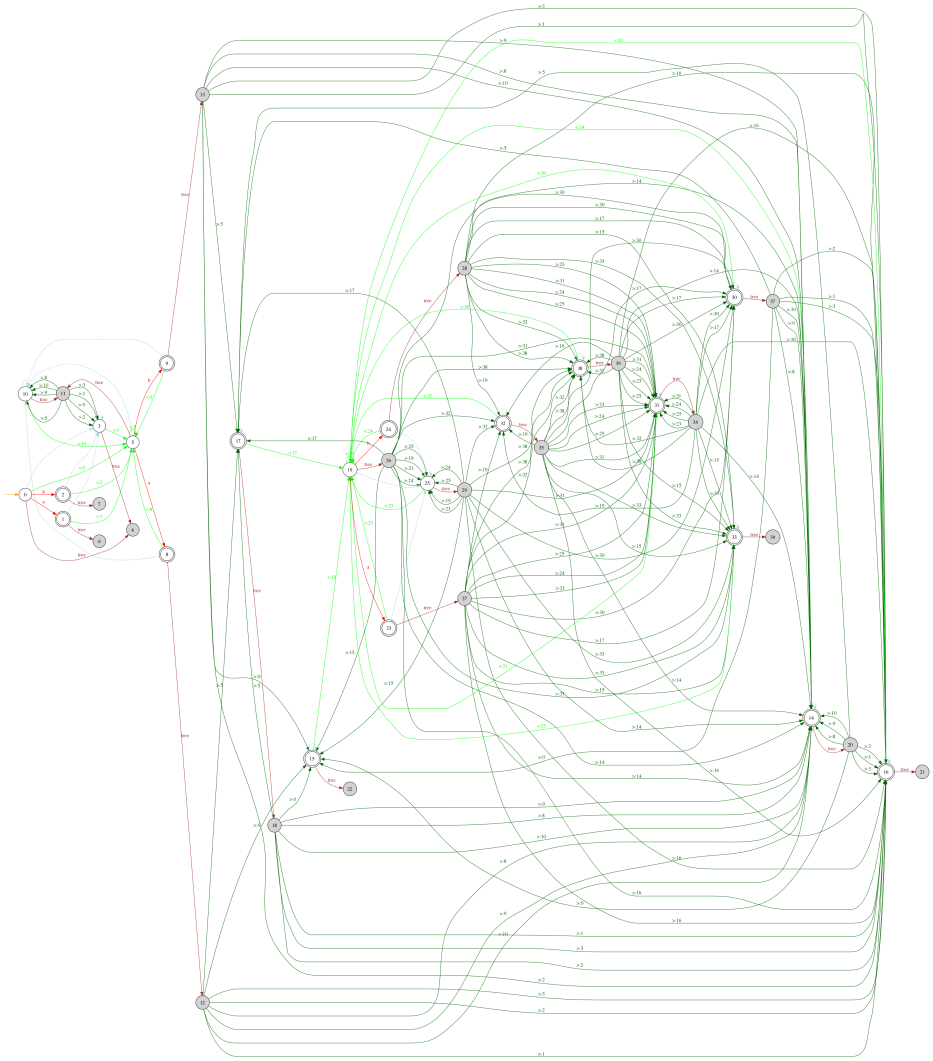


Fig. 15: Deterministic NWA: $det(nwa(ch^*(a+b))$.

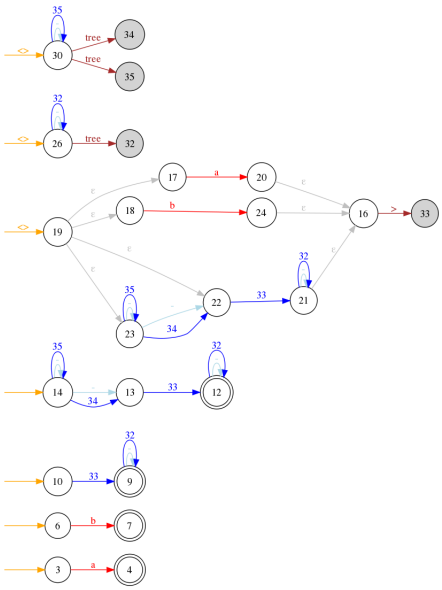


Fig. 16: Stepwise hedge automaton from NWA for $sha(nwa(ch^*(a+b)))$.

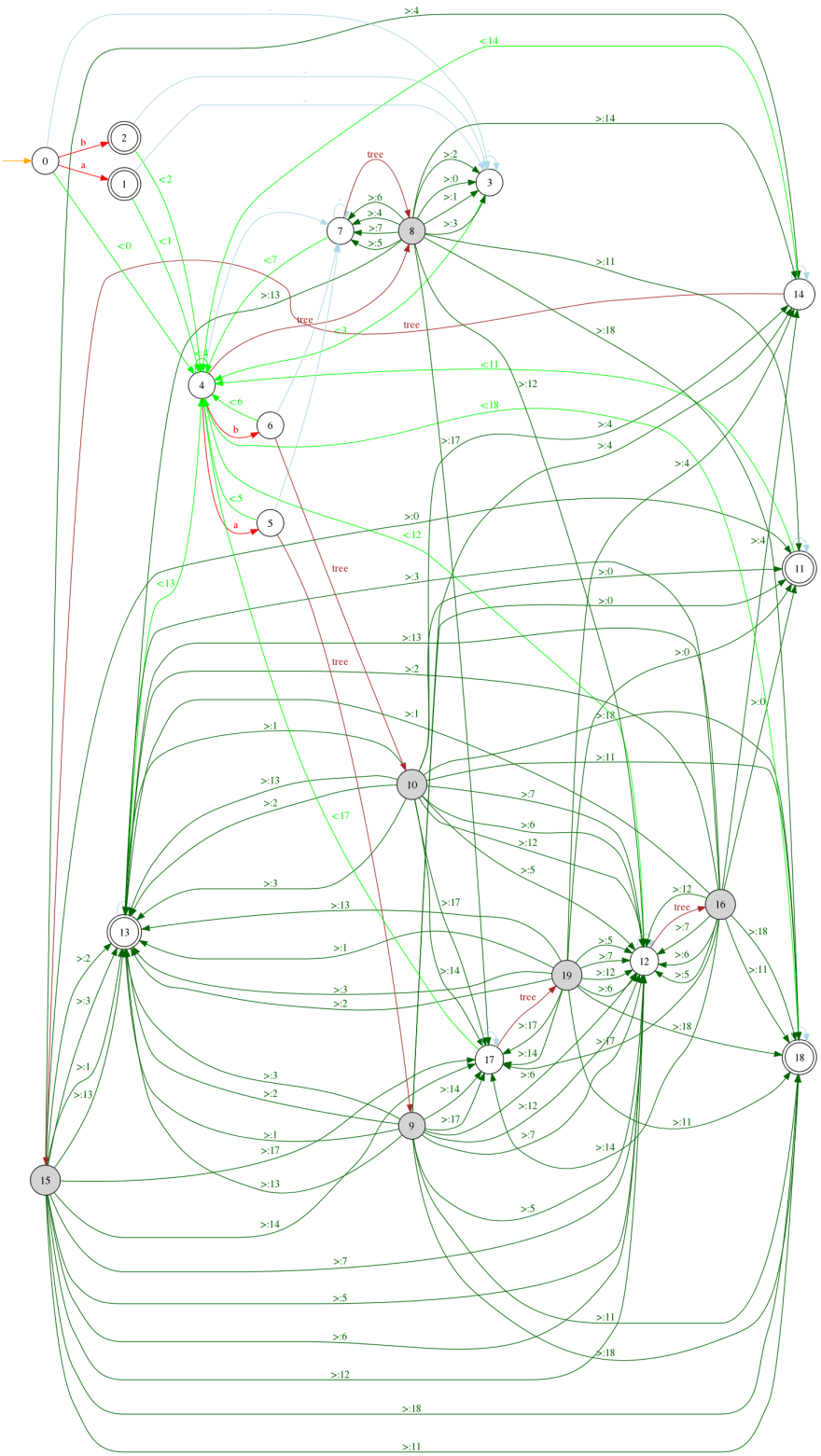


Fig. 17: Determinization of NWA from stepwise hedge automaton: $det(nwa(sha(ch^*(a+b))))$.