International Journal of Foundations of Computer Science
Vol. 26, No. 8 (2015) 1009–1040

DOI: 10.1142/S0129054115400110



# From Finite Automata to Regular Expressions and Back — A Summary on Descriptional Complexity

Hermann Gruber

*knowledgepark AG, Leonrodstr. 68*
*80636 München, Germany*
*hermann.gruber@knowledgepark-ag.de*

Markus Holzer

*Institut für Informatik, Universität Giessen*
*Arndtstr. 2, 35392 Giessen, Germany*
*holzer@informatik.uni-giessen.de*

Received 3 December 2014
Accepted 12 July 2015
Communicated by Zoltán Ésik

The equivalence of finite automata and regular expressions dates back to the seminal paper of Kleene on events in nerve nets and finite automata from 1956. In the present paper we tour a fragment of the literature and summarize results on upper and lower bounds on the conversion of finite automata to regular expressions and *vice versa*. As an interesting special case also one-unambiguous regular expressions, a sort of a deterministic version of a regular expression, are considered. We also briefly recall the known bounds for the removal of spontaneous transitions ($\varepsilon$-transitions) on non-$\varepsilon$-free nondeterministic devices. Moreover, we report on recent results on the average case descriptional complexity bounds for the conversion of regular expressions to finite automata and new developments on the state elimination algorithm that converts finite automata to regular expressions.

*Keywords*: Finite automata; regular expressions; descriptional complexity; upper bounds; lower bounds.



## 1. Introduction

There is a vast literature documenting the importance of the notion of finite automata and regular expressions as an enormously valuable concept in theoretical computer science and applications. It is well known that these two formalisms are equivalent, and in almost all monographs on automata and formal languages one finds appropriate constructions for the conversion of finite automata to equivalent regular expressions and back. Regular expressions, introduced by Kleene [75], are well suited for human users and therefore are often used as interfaces to specify certain patterns or languages. For example, in the widely available programming environment Unix, regular(-like) expressions can be found in legion of software

tools like, e.g., `awk`, `ed`, `emacs`, `egrep`, `lex`, `sed`, `vi`, etc., to mention a few of them. On the other hand, automata [102] immediately translate to efficient data structures, and are very well suited for programming tasks. This naturally raises the interest in conversions among these two different notions. Our tour on the subject covers some (recent) results in the fields of descriptional and computational complexity. During the last decade descriptional aspects on finite automata and regular expressions formed an extremely vivid area of research. For recent surveys on descriptional complexity issues of finite automata and regular expressions we refer to, for example, [44, 62–66, 112]. This was not only triggered by appropriate conferences and workshops on that subject, but also by the availability of mathematical tools and the influence of empirical studies. For obvious reasons, this survey lacks completeness, as finite automata and regular expressions fall short of exhausting the large number of related problems considered in the literature. We give a view of what constitutes, in our opinion, the most interesting recent links to the problem area under consideration.

Before we start our tour some definitions are in order. First of all, our nomenclature of finite automata is as follows: a *nondeterministic finite automaton with $\varepsilon$-transitions* ($\varepsilon$-NFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is the finite set of *states*, $\Sigma$ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the *transition function*. If a finite automaton has no $\varepsilon$-transitions, i.e., the transition function is restricted to $\delta : Q \times \Sigma \to 2^Q$, then we simply speak of a *nondeterministic finite automaton* (NFA). Moreover, a nondeterministic finite automaton is *deterministic* (DFA) if and only if $|\delta(q, a)| = 1$, for all states $q \in Q$ and letters $a \in \Sigma$. The *language accepted* by the finite automaton $A$ is defined as $L(A) = \{ w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \}$, where the transition function is recursively extended to $\delta : Q \times \Sigma^* \to 2^Q$. Second, we turn to the definition of regular expressions: the *regular expressions* over an alphabet $\Sigma$ and the languages they describe are defined inductively in the usual way:[a] $\emptyset$, $\varepsilon$, and every letter $a$ with $a \in \Sigma$ is a regular expression, and when $s$ and $t$ are regular expressions, then $(s + t)$, $(s \cdot t)$, and $(s)^*$ are also regular expressions. The language defined by a regular expression $r$, denoted by $L(r)$, is defined as follows: $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$, $L(s + t) = L(s) \cup L(t)$, $L(s \cdot t) = L(s) \cdot L(t)$, and $L(s^*) = L(s)^*$. For further details on finite automata and regular expressions we refer to, e.g., [67].

We start our tour on the subject with the question on the appropriate measure for finite automata and regular expressions. We discuss this topic in detail in Sec. 2. There we also concentrate on two specific measures: on star height for regular expressions and cycle rank for the automaton side. By Eggan's theorem [29] both measures are related to each other. Recent developments, in particular on the

[a]For convenience, parentheses in regular expressions are sometimes omitted and the concatenation is simply written as juxtaposition. The priority of operators is specified in the usual fashion: concatenation is performed before union, and star before both product and union.

conversion from finite automata to regular expressions, utilize this connection to prove upper and lower bounds. Then in Sec. 3 we take a closer look on the conversion from regular expressions to equivalent finite automata. We recall the most prominent conversion algorithms such as Thompson's construction and its optimized version the follow automaton, the position or Glushkov automaton, and conversion by computations of the (partial-)derivatives. We summarize the known relations on these devices, which were mostly found during the last decade. Significant differences on these constructions are pointed out and the presented developments on lower bound and upper bound results enlighten the efficiency of these algorithms. Some of the bounds are sensitive to the size of the alphabet. We also briefly recall the known bounds on the conversion of one-unambiguous expresssion to automata and *vice versa*. One-unambiguous expressions can be seen as a sort of deterministic variant of a regular expression and is defined *via* the position automaton. Besides worst case descriptional complexity results on the synthesis problem of finite automata from regular expressions, we also list some recent results on the average case complexity of the transformation of regular expressions to finite automata. Finally, in Sec. 4 we consider the converse transformation. Again, we summarize some of the few conversion techniques, but then stick in more detail to the so-called state elimination technique. The reason for that is, that in [105], it was shown that almost all conversion methods can be recast as variants of the state elimination technique. Here, the ordering in which the states are eliminated can largely affect the size of the regular expression corresponding to the given finite automaton. We survey some heuristics that have been proposed for this goal. For appropriate choices of the ordering, nontrivial upper bounds on regular expression size can be proved. By looking at the transition structure of the NFA, results from graph theory can help in obtaining shorter expressions. There we try to illustrate the key insights with the aid of examples, thereby avoiding the need for a deeper dive into graph theoretic concepts. We also explain the technique by which the recent lower bounds on regular expression size were obtained. In this part, the known upper and lower bounds match only in the sense that we can identify the rough order of magnitude. So we observe an interesting tension between algorithms with provable performance guarantees, other heuristics that are observed to behave better in experiments, and some lower bounds, which seize the expectations that we may have on practical algorithms. Finally, we also recall some results on the conversion of automata accepting one-unambiguous languages to one-unambiguous expressions.

## 2. Measures on Finite Automata and Regular Expressions

What can be said about the proper measure on finite automata and regular expressions? For finite automata there are two commonly accepted measures, namely the number of states and the number of transitions. The measure sc (nsc, respectively) counts the number of states of a deterministic (nondeterministic, respectively) finite automaton and tc (ntc, respectively) does the same for the number of transitions

for the appropriate devices. Moreover, $\mathsf{nsc}_\varepsilon$ ($\mathsf{ntc}_\varepsilon$, respectively) gives the number of states (transitions, respectively) in an $\varepsilon$-NFA. The following relations between these measures are well known — see also [92, 94, 102].

**Theorem 1.** *Let $L \subseteq \Sigma^*$ be a regular language. Then*

(1) $\mathit{nsc}_\varepsilon(L) = \mathit{nsc}(L) \leq \mathit{sc}(L) \leq 2^{\mathit{nsc}(L)}$ *and* $\mathit{tc}(L) = |\Sigma| \cdot \mathit{sc}(L)$ *and*
(2) $\mathit{nsc}(L) - 1 \leq \mathit{ntc}_\varepsilon(L) \leq \mathit{ntc}(L) \leq |\Sigma| \cdot (\mathit{nsc}(L))^2$,

*where* $\mathit{sc}(L)$, $\mathit{tc}(L)$ ($\mathit{nsc}(L)$, $\mathit{ntc}(L)$, *respectively*) *refers to the minimum* $\mathsf{sc}$ ($\mathsf{nsc}$, $\mathsf{ntc}$, *respectively*) *among all DFAs* (*NFAs, respectively*) *accepting $L$. Similarly,* $\mathit{nsc}_\varepsilon(L)$ ($\mathit{ntc}_\varepsilon(L)$, *respectively*) *is the minimum* $\mathsf{nsc}_\varepsilon$ ($\mathsf{ntc}_\varepsilon$, *respectively*) *among all $\varepsilon$-NFAs for the language $L$.*

As it is defined above, deterministic transition complexity is not an interesting measure by itself, because it is directly related to $\mathsf{sc}$, the deterministic state complexity. But the picture changes when deterministic transition complexity is defined in terms of partial DFAs. Here, a *partial* DFA is an NFA whose transition function $\delta$ satisfies $|\delta(q, a)| \leq 1$, for all states $q \in Q$ and all alphabet symbols $a \in \Sigma$. A partial DFA cannot save more than one state compared to an ordinary DFA, but it can save a considerable number of transitions in some cases. This phenomenon is studied, e.g., in [33, 83, 84]. Further measures for the complexity of finite automata, in particular measures related to unambiguity and limited nondeterminism, can be found in [44–46, 65, 70, 77–79, 100, 101, 103].

Now let us come to measures on regular expressions. While there are the two commonly accepted measures for finite automata, there is no general agreement in the literature about the proper measure for regular expressions. We summarize some important ones: the measure $\mathsf{size}$ is defined to be the total number of symbols (including $\emptyset$, $\varepsilon$, symbols from alphabet $\Sigma$, all operation symbols, and parentheses) of a completely bracketed regular expression (for example, used in [2], where it is called length). Another measure related to the reverse polish notation of a regular expression is $\mathsf{rpn}$, which gives the number of nodes in the syntax tree of the expressions (parentheses are not counted). This measure is equal to the length of a (parenthesis-free) expression in post-fix notation [2]. The alphabetic width $\mathsf{awidth}$ is the total number of alphabetic symbols from $\Sigma$ (counted with multiplicity) [30, 91]. Relations between these measures have been studied, e.g., in [30, 31, 48, 73].

**Theorem 2.** *Let $L \subseteq \Sigma^*$ be a regular language. Then*

(1) $\mathit{size}(L) \leq 3 \cdot \mathit{rpn}(L)$ *and* $\mathit{size}(L) \leq 8 \cdot \mathit{awidth}(L) - 3$,
(2) $\mathit{awidth}(L) \leq \frac{1}{2} \cdot (\mathit{size}(L) + 1)$ *and* $\mathit{awidth}(L) \leq \frac{1}{2} \cdot (\mathit{rpn}(L) + 1)$, *and*
(3) $\mathit{rpn}(L) \leq \frac{1}{2} \cdot (\mathit{size}(L) + 1)$ *and* $\mathit{rpn}(L) \leq 4 \cdot \mathit{awidth}(L) - 1$,

*where* $\mathit{size}(L)$ ($\mathit{rpn}(L)$, $\mathit{awidth}(L)$, *respectively*) *refers to the minimum* $\mathsf{size}$ ($\mathsf{rpn}$, $\mathsf{awidth}$, *respectively*) *among all regular expressions denoting $L$.*

Further measures for the complexity of regular expressions can be found in [8, 30, 31, 54]. To our knowledge, these latter measures received far less attention to date.

In the remainder of this section we concentrate on two important measures on regular expression and finite automata that at first glance do not seem to be related to each other: *star height* and *cycle rank* or *loop complexity*. Both measures are very important, in particular, for the conversion of finite automata to regular expressions and for proving lower bound results on the latter. Intuitively, the star height of an expression measures the nesting depth of Kleene-star operations. More precisely, for a regular expression, the *star height* is inductively defined by

$$\begin{aligned} \mathsf{height}(\emptyset) = \mathsf{height}(\varepsilon) = \mathsf{height}(a) &= 0, \\ \mathsf{height}(s+t) = \mathsf{height}(s \cdot t) &= \max\left(\mathsf{height}(s), \mathsf{height}(t)\right), \end{aligned}$$

and

$$\mathsf{height}(s^*) = 1 + \mathsf{height}(s).$$

The star height of a regular language $L$, denoted by $\mathsf{height}(L)$ is then defined as the minimum star height among all regular expressions describing $L$. The seminal work dealing with the star height of regular expressions [29] established a relation between the theory of regular languages and the theory of digraphs. The *cycle rank*, or *loop complexity*, of a digraph $D$ is defined inductively by the following rules: (i) the cycle rank of an acyclic digraph is zero, (ii) cycle rank of a strongly connected component (SCC) of the digraph with at least one arc is 1 plus the minimum cycle rank among the digraphs obtainable from $D$ by deleting a vertex, and (iii) the cycle rank of a digraph with multiple SCCs equals the maximum cycle rank among the sub-digraphs induced by these components. So, roughly speaking, the cycle rank of a digraph is large if the cycle structure of the digraph is intricate and highly connected. The following relation between cycle rank of automata and star height of regular languages became known as *Eggan's Theorem* [29, 105]:

**Theorem 3.** *The star height of a regular language $L$ equals the minimum cycle rank among all $\varepsilon$-NFAs accepting $L$.*

An apparent difficulty with applying Eggan's Theorem is that the minimum is taken over infinitely many automata, and the cycle rank of the minimum DFA for the language does not always attain that minimum. That makes the star height a very intricate property of regular languages. Indeed, the decidability status of the star height problem was open for more than two decades, until a very difficult algorithm was given in [60]. For recent progress on algorithms for the star height problem, the reader is referred to [74]. From the above it is immediate that $\mathsf{height}(L) \leq \mathsf{nsc}(L)$. If the language is given as a regular expression, a result from [49] tells us a much sweeter truth:

**Lemma 4.** *Let $L \subseteq \Sigma^*$ be a regular language with alphabetic width $n$. Then $\mathsf{height}(L) \leq 3\log(n+1)$.*

The idea behind the proof of this lemma is that we can convert a regular expression into a $\varepsilon$-NFA of similar size. The cycle structure of that automaton is well-behaved; and thus its cycle rank is low compared to the size of the automaton. Then Eggan's Theorem is used to convert the automaton back into a regular expression of low star height. We return to the relationship between required size and star height of regular expressions later on. Now let us turn our attention to the conversion of regular expressions into equivalent finite automata.

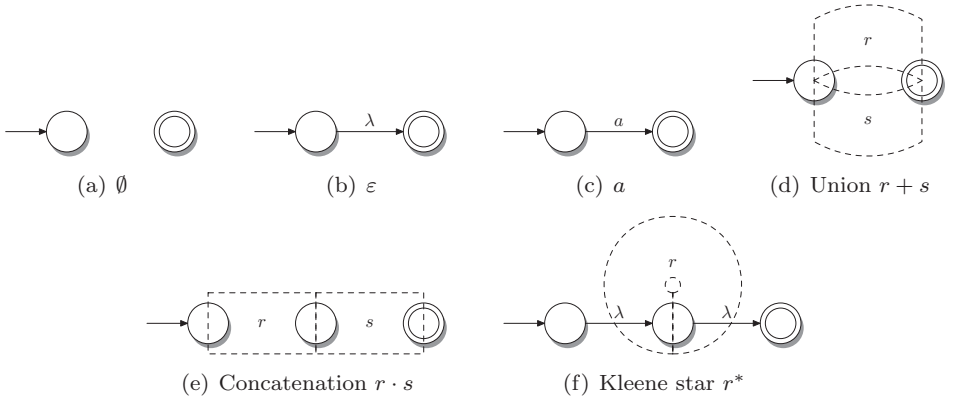
## 3. From Regular Expressions to Finite Automata

The conversion of regular expressions into small finite automata has been intensively studied for more than half a century. Basically the algorithms can be classified according to whether the output is an $\varepsilon$-NFA, NFA, or even a DFA. In principle one can distinguish between the following three major construction schemes and variants thereof:

(1) *Thompson's construction* [109] and optimized versions, such as the *follow automaton* [73, 99],
(2) construction of the *position automaton*, or *Glushkov automaton* [43, 91], and
(3) computation of the (*partial*) *derivative automaton* [4, 16].

Further automata constructions from regular expressions can be found in, e.g., [6, 14, 21, 34, 72, 111]. We briefly explain some of these approaches in the course of action — for further readings on the subject we refer to [105].

Thompson's construction [109] was popularized by the implementation of the Unix command `grep` (globally search a regular expression and print). It amounts to the recursive connection of sub-automata *via* $\varepsilon$-transitions. These sub-automata are connected in parallel for the union, in series for the concatenation, and in an iterative fashion for the Kleene star. This yields an $\varepsilon$-NFA with a linear number of states and transitions. A structural characterization of the Thompson automaton in terms of the underlying digraph is given in [41, 42]. Thompson's classical construction went through several stages of adaption and optimization. The construction with the least usage of $\varepsilon$-transitions was essentially given already in 1961 by Ott and Feinstein [99], which also can be found in [27, 85, 90] — see Fig. 1. Later this construction was refined by Ilie and Yu [73] and promoted under the name *follow automaton*. In fact, the follow automaton is constructed from a regular expression $r$ by recursively applying the construction of Ott and Feinstein and simultaneously improving on the use of $\varepsilon$-transitions in the following sense: (i) in the concatenation construction a $\varepsilon$-transition into the common state to both sub-automata leads to an appropriate state merging; similarly a state merging is done for an $\varepsilon$-transition leaving the common state, (ii) in the Kleene star construction, if the middle state is on a cycle of $\varepsilon$-transitions, all these transitions are removed, and all states of the cycle are merged, and (iii) after the construction is finished, a possible $\varepsilon$-transition from the start state is removed and both involved states are merged appropriately.




(a) $\emptyset$ (b) $\varepsilon$ (c) $a$ (d) Union $r + s$

(e) Concatenation $r \cdot s$ (f) Kleene star $r^*$

Fig. 1. The inductive construction of Ott and Feinstein yielding the precursor of the follow automaton $A_f(r)$ for a regular expression $r$.

Notice that the automaton thus constructed may still contain $\varepsilon$-transitions. In order to amend the situation, an $\varepsilon$-removal procedure is applied: simply replace any sequence of an $\varepsilon$-transition followed by an $a$-transition by directly connecting the states on both ends of the sequence by a single $a$-transition directly. A final step takes care about the $\varepsilon$-transition to the final state. This results in the follow automaton $A_f(r)$ of [73], for the regular expression $r$.

**Example 5.** *Imagine a software buffer supporting the actions $a$ ("add work packet") and $b$ ("remove work packet"), with a total capacity of $n$ packets. Let $r_n$ denote the regular expression for the action sequences that result in an empty buffer and never cause the buffer to exceed its capacity. Then*

$$r_1 = (ab)^* \quad \textit{and} \quad r_n = (a \cdot r_{n-1} \cdot b)^*, \quad \textit{for } n \geq 2.$$

*Following the construction of the follow automaton as described in* [73] *results in the automaton depicted in Fig.* 2*. Observe the constructed automaton is minimal, which is not the case in general. This is our running example, where the behaviour of the state elimination technique described in the next section is discussed in more detail.* □

Preliminary bounds on the required size of a finite automaton equivalent to a given regular expression were given in [73]. Later, a tight bound in terms of reverse polish notation [56], and also a tight bound in terms of alphabetic width was found [48]. In the next theorem we summarize the results from [48, 56, 73] —

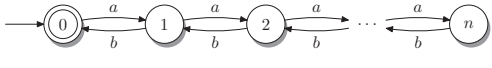



Fig. 2. The follow automaton $A_f(r_n)$ accepting $L(r_n)$.

here size of an automaton refers to the sum of the number of states and the number transitions:

**Theorem 6.** *Let $n \geq 1$, and $r$ be a regular expression of alphabetic width $n$. Then size $\frac{22}{5}n$ is sufficient for an equivalent $\varepsilon$-NFA accepting $L(r)$. In terms of reverse polish length, the bound is $\frac{22}{15}(\mathsf{rpn}(r)+1)+1$. Furthermore, there are infinitely many languages for which both bounds are tight.*

The aid for the tight bound in terms of the alphabetic width stated in the previous theorem is a certain normal form for regular expressions, which is a refinement of the *star normal form* from [14]. The definition reads as follows — transformation into strong star normal form preserves the described language, and is weakly monotone with respect to all usual size measures:

**Definition 7.** *The operators $\circ$ and $\bullet$ are defined on regular expressions*[b] *over alphabet $\Sigma$. The first operator is given by: $a^\circ = a$, for $a \in \Sigma$, $(r+s)^\circ = r^\circ + s^\circ$, $r^{?\circ} = r^\circ$, $r^{*\circ} = r^\circ$; finally, $(r \cdot s)^\circ = r \cdot s$, if $\varepsilon \notin L(rs)$ and $r^\circ + s^\circ$ otherwise. The second operator is given by: $a^\bullet = a$, for $a \in \Sigma$, $(r+s)^\bullet = r^\bullet + s^\bullet$, $(r \cdot s)^\bullet = r^\bullet \cdot s^\bullet$, $r^{*\bullet} = r^{\bullet\circ *}$; finally, $r^{?\bullet} = r^\bullet$, if $\varepsilon \in L(r)$ and $r^{?\bullet} = r^{\bullet ?}$ otherwise. The* strong star normal form *of an expression $r$ is then defined as $r^\bullet$.*

What about the transformation of a regular expression into a finite automaton if $\varepsilon$-transitions are not allowed? One way to obtain an NFA directly is to perform the standard algorithm for removing $\varepsilon$-transitions, see, e.g., [67], which may increase the number of transitions at most quadratically. Another way is to directly implement the procedure during the recursive construction using non-$\varepsilon$-transitions to connect the sub-automata appropriately. Constructions of this kind can be found in, e.g., [3, 76]. For the conversion of $\varepsilon$-NFAs to NFAs the lower bound of [71] applies. There it was shown that there are infinitely many languages which are accepted by $\varepsilon$-NFAs with $O(n \cdot (\log n)^2)$ transitions, such that any NFA needs at least $\Omega(n^2)$ transitions. This lower bound is witnessed by a language over a growing size alphabet and shows that, in this case, the standard algorithm for removing $\varepsilon$-transitions cannot be improved significantly. For the case of binary alphabets, a lower bound of $\Omega(n \cdot 2^{c \cdot \sqrt{\log n}})$, for every $c < \frac{1}{2}$, was proved in [71] as well.

Another possibility to obtain ordinary NFAs is to directly construct the *position automaton*, also called the *Glushkov automaton* [43] — see also [91]. Intuitively, the states of this automaton correspond to the alphabetic symbols or, in other words, to positions between subsequent alphabetic symbols in the regular expression. Let us be more precise: assume that $r$ is a regular expression over $\Sigma$ of alphabetic width $n$. In $r$ we attach subscripts to each letter referring to its position (counted

[b]Since $\emptyset$ is only needed to denote the empty set, and the need for $\varepsilon$ can be substituted by the operator $L^? = L \cup \{\varepsilon\}$, an alternative is to introduce also the $^?$-operator and instead forbid the use of $\emptyset$ and $\varepsilon$ inside non-atomic expressions. This is sometimes more convenient, since one avoids unnecessary redundancy already at the syntactic level [48].

from left to right) in $r$. This yields a *marked* expression $\overline{r}$ with distinct input symbols over an alphabet $\overline{\Sigma}$ that contains all letters that occur in $\overline{r}$. To simplify our presentation we assume that the same notation is used for unmarking, i.e., $\overline{\overline{r}} = r$. Then in order to describe the position automaton we need to define the following sets of positions on the marked expression. Let $\mathsf{Pos}(r) = \{1, 2, \ldots, \mathsf{awidth}(r)\}$ and $\mathsf{Pos}_0(r) = \mathsf{Pos}(r) \cup \{0\}$. The position set $\mathsf{First}$ takes care of the possible beginnings of words in $L(\overline{r})$. It is inductively defined as follows:

$$\begin{aligned}
\mathsf{First}(\emptyset) &= \mathsf{First}(\varepsilon) = \emptyset,\\
\mathsf{First}(a_i) &= \{i\},\\
\mathsf{First}(s+t) &= \mathsf{First}(s) \cup \mathsf{First}(t),\\
\mathsf{First}(s \cdot t) &= \begin{cases} \mathsf{First}(s) \cup \mathsf{First}(t) & \text{if } \varepsilon \in L(s)\\ \mathsf{First}(s) & \text{otherwise,}\end{cases}
\end{aligned}$$

and

$$\mathsf{First}(s^*) = \mathsf{First}(s).$$

Accordingly the position set $\mathsf{Last}$ takes care of the possible endings of words in $L(\overline{r})$. Its definition is similar to the definition of $\mathsf{First}$, except for the concatenation, which reads as follows:

$$\mathsf{Last}(s \cdot t) = \begin{cases} \mathsf{Last}(s) \cup \mathsf{Last}(t) & \text{if } \varepsilon \in L(t)\\ \mathsf{Last}(t) & \text{otherwise.}\end{cases}$$

Finally, the set $\mathsf{Follow}$ takes care about the possible continuations in the words in $L(\overline{r})$. It is inductively defined as

$$\begin{aligned}
\mathsf{Follow}(\emptyset) = \mathsf{Follow}(\varepsilon) &= \mathsf{Follow}(a_i) = \emptyset\\
\mathsf{Follow}(s+t) &= \mathsf{Follow}(s) \cup \mathsf{Follow}(t)\\
\mathsf{Follow}(s \cdot t) &= \mathsf{Follow}(s) \cup \mathsf{Follow}(t) \cup \mathsf{Last}(s) \times \mathsf{First}(t)
\end{aligned}$$

and

$$\mathsf{Follow}(s^*) = \mathsf{Follow}(s) \cup \mathsf{Last}(s) \times \mathsf{First}(s).$$

Then the *position automaton* for $r$ is defined as $A_{pos}(r) = (\mathsf{Pos}_0(r), \Sigma, \delta_{pos}, 0, F_{pos})$, where $\delta(0, a) = \{\, j \in \mathsf{First}(\overline{r}) \mid a = \overline{a_j} \,\}$, for every $a \in \Sigma$, and the transition function $\delta(i, a) = \{\, j \mid (i, j) \in \mathsf{Follow}(\overline{r}) \text{ and } a = \overline{a_j} \,\}$, for every $i \in \mathsf{Pos}(r)$ and $a \in \Sigma$, and $F_{pos} = \mathsf{Last}(\overline{r})$, if $\varepsilon \notin L(r)$, and $F_{pos} = \mathsf{Last}(\overline{r}) \cup \{0\}$ otherwise.

**Example 8.** *Consider the regular expression $r_n$ from Example 5. If we mark the regular expression $r_n$, then we obtain $\overline{r_n} = (a_1(a_2(a_3 \ldots b_{2n-2})^* b_{2n-1})^* b_{2n})^*$. Easy calculations show that the position sets read as follows:*

$$\begin{aligned}
\mathit{First}(\overline{r_n}) &= \{1\}\\
\mathit{Last}(\overline{r_n}) &= \{2n\}
\end{aligned}$$

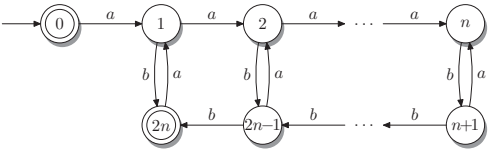



Fig. 3. The position automaton $A_{pos}(r_n)$ accepting $L(r_n)$.

*and*

$$\mathsf{Follow}(\overline{r_n}) = \{\, (i, i+1) \mid 1 \le i < 2n \,\} \cup \{\, (i, 2n-i+1), (2n-i+1, i) \mid 1 \le i \le n \,\}.$$

*The position automaton on state set* $\mathsf{Pos}_0(r)$ *is depicted in Fig.* 3. *Here the set of final states is* $F_{pos} = \{0, 2n\}$, *since* $\varepsilon \in L(r_n)$. *Observe that the follow automaton* $A_f(r_n)$ *can be obtained from* $A_{pos}(r_n)$ *by taking the quotient of automata, i.e., merging of states, with respect to the relation*[c] $\equiv_f$ *described in* [73], *which contains the elements* $(i, 2n-i)$, *for* $0 \le i \le 2n$. *This leads to the merging of states* 0 *and* $2n$, *states* 1 *and* $2n-1$, *states* 2 *and* $2n-2$, *up to states* $n-1$ *and* $n+1$. □

An immediate advantage of the position automaton is observed, e.g., in [1, 7]: for a regular expression $r$ of alphabetic width $n$, for $n \ge 0$, the position automaton $A_{pos}(r)$ always has precisely $n+1$ states. Simple examples, such as the singleton set $\{a^n\}$, show that this bound is tight. Nevertheless, several optimizations have been developed that give NFAs having often a smaller number of states, while the underlying constructions are mathematically sound refinements of the basic construction. A characterization of the position automaton is given in [18]. Moreover, structural comparisons between the position automaton with its refined versions, namely the *follow automaton*, the *partial derivative automaton* [4], or the *continuation automaton* [7] is given in [20, 73]. The partial derivative automaton is known under different names, such as *equation automaton* [93] or *Antimirov automaton* [4]. Further results on structural properties of these automata, when built from regular expressions in star normal form, can be found in [19, 22]. A quantitative comparison on the sizes of the the aforementioned NFAs for specific languages shows that they can differ a lot. The results listed in Table 1 are taken from [73] — here size of an automaton refers to the sum of the number of states and the number transitions. For the reason of comparison also the *common follow set automaton* $A_{cfs}$ is listed — since the description of $A_{cfs}$ is quite involved we refer the reader to [72]. There, this automaton was used to prove an upper bound on the number of transitions. The issue on transitions for NFAs, in particular when changing from an $\varepsilon$-NFA to an NFA, is discussed next.

Despite the mentioned optimizations, except for the common follow set automaton, all of these constructions share the same problem with respect to the number

[c]The relation $\equiv_f$ is defined as follows: for $i, j \in \mathsf{Pos}_0(r)$ let $i \equiv_f j$ if and only if (i) both $i$ and $j$ or none belong to $\mathsf{Last}(r)$ and (ii) their follow positions w.r.t. the expression $r$ are the same, that is, $\{\, i' \mid (i, i') \in \mathsf{Follow}(r) \,\} = \{\, j' \mid (j, j') \in \mathsf{Follow}(r) \,\}$.

Table 1. Comparing sizes of some automata constructions for specific languages from the literature — gray shading marks the smallest automaton. Here $A_f$ refers to the follow automaton, $A_{pd}$ to the partial derivative automaton, $A_{pos}$ to the position automaton, and $A_{cfs}$ to the common follow set automaton. Moreover, $|r_n|$ ($|r_{n,m}|$, respectively) refers to the alphabetic width of the regular expression $r_n$ ($r_{n,m}$, respectively).

| Expression | Finite Automaton | | | |
|---|---|---|---|---|
| | $A_f(\cdot)$ | $A_{pd}(\cdot)$ | $A_{pos}(\cdot)$ | $A_{cfs}(\cdot)$ |
| $r_1 = (a_1 + \varepsilon)^*$ and $r_{n+1} = (r_n + s_n)^*$ with $s_n = r_n[a_j \mapsto a_{j+2^{n-1}}]$ | $\Theta(\lvert r_n\rvert)$ | $\Theta(\lvert r_n\rvert^2)$ | | $\Theta(\lvert r_n\rvert \cdot (\log \lvert r_n\rvert)^2)$ |
| $r_{n,m} = s_n(s_n + t_m)$ with $s_n = \sum_{i=1}^{n} a_i$ and $t_m = \sum_{i=1}^{m} b_i$ | $\Theta(\lvert r_{n,m}\rvert)$ | | $\Theta(\lvert r_{n,m}\rvert^2)$ | $\Theta(\lvert r_{n,m}\rvert \cdot (\log \lvert r_{n,m}\rvert)^2)$ |
| $r_n = \sum_{i=1}^{n} a_i \cdot t_n^*$ with $t_n = \sum_{i=1}^{n} b_i$ | $\Theta(\lvert r_n\rvert)$ | $\Theta(\lvert r_n\rvert^{1/2})$ | $\Theta(\lvert r_n\rvert^{3/2})$ | $\Theta(\lvert r_n\rvert \cdot (\log \lvert r_n\rvert)^2)$ |
| $r_n = \prod_{i=1}^{n} (a_i + \varepsilon)$ | $\Theta(\lvert r_n\rvert^2)$ | | | $\Theta(\lvert r_n\rvert \cdot (\log \lvert r_n\rvert)^2)$ |

of transitions. An easy upper bound on the number of transitions in the position automaton is $O(n^2)$, independent of alphabet size. It is not hard to prove that the position automaton for the regular expression

$$r_n = (a_1 + \varepsilon) \cdot (a_2 + \varepsilon) \cdots (a_n + \varepsilon)$$

has $\Omega(n^2)$ transitions. It appears to be difficult to avoid such a quadratic blow-up in actual size if we stick to the NFA model. Also if we transform the expression first into a $\varepsilon$-NFA and perform the standard algorithm for removing $\varepsilon$-transitions, see, e.g., [67], we obtain no better result. This naturally raises the question of comparing the descriptional complexity of NFAs over regular expressions. For about forty years, it appears to have been considered as an unproven factoid that a quadratic number of transitions will be inherently necessary in the worst case (cf. [72]). A barely super-linear lower bound of $\Omega(n \log n)$ on the number of transitions of any NFA accepting the language of the expression $r_n$ was proved [72]. More interestingly, the main result of that paper is an algorithm transforming a regular expression of size $n$ into an equivalent NFA with at most $O(n \cdot (\log n)^2)$ transitions. See Fig. 4 on how the algorithm of [72] saves transitions for regular expression $r_n$, explained for $n = 5$. In fact, this upper bound made their lower bound look reasonable at once! Shortly thereafter, an efficient implementation of that conversion algorithm was presented [57], and the lower bound was improved in [80] to $\Omega(n \cdot (\log n)^2 / \log \log n)$. Later work [106] established that any NFA accepting language $L(r_n)$ indeed must

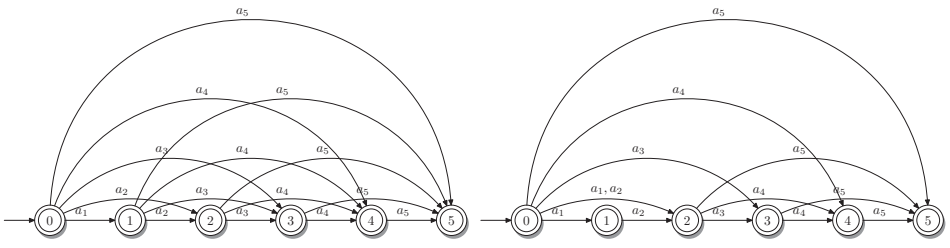



Fig. 4. Let $r_n = (a_1 + \varepsilon) \cdot (a_2 + \varepsilon) \cdots (a_n + \varepsilon)$ and $n = 5$. Position automaton $A_{pos}(r_5)$ (left) and its refined version the common follow set automaton $A_{cfs}(r_5)$ (right) accepting language $L(r_5)$; in both cases the dead state and all transitions leading to it are not shown. The automaton $A_{cfs}(r_5)$ is obtained as follows: the state 1 of $A_{pos}(r_5)$ is split such that the new state gets the outgoing transitions labeled with $a_3$, $a_4$, and $a_5$, and is finally identified with state 2, which can be done since it has the same outgoing transitions.



have at least $\Omega(n \cdot (\log n)^2)$ transitions. So the upper bound of $O(n \cdot (\log n)^2)$ from [72] is asymptotically tight:

**Theorem 9.** *Let $n \geq 1$ and $r$ be a regular expression of alphabetic width $n$. Then $O(n \cdot (\log n)^2)$ transitions are sufficient for an NFA to accept $L(r)$. Furthermore, there are infinitely many languages for which this bound is tight.*

Notice that the example witnessing the lower bound is over an alphabet of growing size. For alphabets of size two, the upper bound was improved first [36] to $O(n \cdot \log n)$, and then even to $n \cdot 2^{O(\log^* n)}$, where $\log^*$ denotes the iterated binary logarithm [106]. Moreover, a lower bound of $\Omega(n \cdot (\log k)^2)$ on the size of NFAs with $k$-letter input alphabet was show in [106], too. Thus the question from [69] whether a conversion from regular expressions over a binary alphabet into NFAs of linear size is possible, is almost settled by now.

**Theorem 10.** *Let $n \geq 1$ and $r$ be a regular expression of alphabetic width $n$ over a binary alphabet. Then $n \cdot 2^{O(\log^* n)}$ transitions are sufficient for a NFA to accept $L(r)$.*

Next, let us briefly discuss the problem of converting regular expressions to DFAs. Again, this problem has been studied by many authors. The obvious way to obtain a DFA is by applying the well known *subset* or *power-set construction* [102]. Due to this construction the obtained DFA may be of exponential size. A more direct and convenient way is to use Brzozowski's derivatives of expressions [16]. A taxonomy comparing many different conversion algorithms is given in [111]. Regarding the descriptional complexity, a tight bound of $2^n + 1$ states in terms of alphabetic width is given in [76]. The mentioned work also establishes a matching lower bound, but for a rather nonstandard definition of size. In terms of alphabetic width, the best lower bound known to date is from [31]. Together, we have the following result:

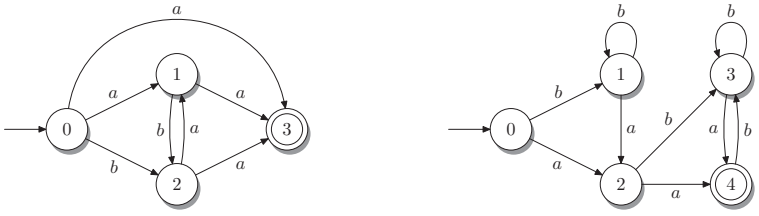



Fig. 5. Two position automata for the equivalent regular expressions $r = (a + b)^*a$ (left) and $s = b^*a(b^*a)^*$ (right). Observe that the left automaton is nondeterministic, while the automaton on the right is a partial deterministic finite automaton.



**Theorem 11.** *Let $n \geq 1$ and $r$ be a regular expression of alphabetic width $n$ over a binary alphabet. Then $2^n + 1$ states are sufficient for a DFA to accept $L(r)$. In contrast, for infinitely many $n$ there are regular expressions $r_n$ of alphabetic width $n$ over a binary alphabet, such that the minimal DFA accepting $L(r_n)$ has at least $\frac{5}{4}2^{\frac{n}{2}}$ states.*

An interesting special case is formed by those regular expressions whose position automaton is a partial deterministic finite automaton. These expressions are referred to as one-unambiguous regular expressions [15], or also as deterministic regular expressions [82, 107]. The intuition behind this concept is that, if we read a word from left to right without look-ahead, it is always clear where in the expression the next symbol can be matched. Recall that for the definition of the position automaton, we introduced the marked version $\overline{r}$ of a regular expression $r$, where the $i$th occurrence of an alphabet symbol is marked with subscript $i$. This comes into play again for the formal definition of one-unambiguous regular expressions. A regular expression $r$ over $\Sigma$ is a *one-unambiguous regular expression* [15], if for all $a \in \Sigma$ and all marked words $u, v, w \in \overline{\Sigma}$ it holds $ua_iv \in L(\overline{r})$ and $ua_jw \in L(\overline{r})$ implies $i = j$. We say that a language is *one-unambiguous*, if it is denoted by some one-unambiguous regular expression.

**Example 12.** *The following examples are from* [15]*. Consider the regular expression $r = (a + b)^*a$. Its marked version is $\overline{r} = (a_1 + b_2)^*a_3$. Taking $u = b_1$, $v = a_3$, and $w = \epsilon$, the words $ua_1v = b_1a_1a_3$ and $ua_2w = b_1a_3$ are both in $L(\overline{r})$. Thus $r$ is not deterministic. The expression $s = b^*a(b^*a)^*$ denotes the same language as $r$ and is one-unambiguous. See Fig.* 5 *for a drawing of the correspondig position automata of these expressions.*

*For $n \geq 1$, the languages denoted by the regular expressions $(a + b)^*a(a + b)^n$ are the canonical examples of languages that are regular but not one-unambiguous. The minimal DFA accepting a one-unambiguous language obeys certain structural restrictions. These can be used to give a rigorous proof that the abovementioned languages are not one-unambiguous.* □

Thus, while deterministic finite automata capture the full class of regular languages, this notion of determinism for regular expressions is strictly less expressive, and the one-unambiguous languages form a proper subclass of the regular

languages.[d] We note in passing that several approaches to generalizing the notion one-unambiguous regular languages have been considered, see, e.g., [40, 58].

There is a linear-time algorithm deciding whether a given regular expression is one-unambiguous, and these expressions admit very fast regular expression matching algorithms [47]. Furthermore, the equivalence problem of one-unambiguous regular expressions, as well as the decision problem whether the language denoted by a regular expression is contained in the language denoted by a one-unambiguous regular expression is solvable efficiently [68]. Also in terms of descriptional complexity, one-unambiguous regular expressions compare favorably[e] to the general case: as shown in [15], a regular expression $r$ is one-unambiguous if and only if $A_{pos}(r)$ is a partial deterministic finite automaton. The language $L_n = \{a^n\}$ is one-unambiguous, and since $\mathsf{nsc}(L_n) = n+1$ and $\mathsf{sc}(L_n) = n+2$, this witnesses the tight bounds for converting one-unambiguous regular expressions into finite automata.

**Theorem 13.** *Let $n \geq 1$ and $r$ be a one-unambiguous regular expression of alphabetic width $n$. Then $n+1$ states ($n+2$ states, respectively) are sufficient for an NFA (DFA, respectively) to accept $L(r)$. Furthermore, there are infinitely many languages for which these bounds are tight.*

Recent developments on the conversion of regular expressions to finite automata show an increasing attention on the study of descriptional complexity in the average case. For instance, in [97] it was shown that, when choosing the expression uniformly at random, the position automaton has $\Theta(n)$ transitions on average, where $n$ refers to the nodes in the parse tree of the expression. A similar result holds w.r.t. alphabetic width, for the position automaton as well as for the partial derivative automaton [12]. A closer look reveals that the number of transitions in the partial derivative automaton is, on average, half the size of the number of transitions in the position automaton [12], for large alphabet sizes; this also holds for the number of states [11]. Results on the average size of $\varepsilon$-NFAs built from Thompson's construction and variants thereof [73, 108, 109] can be found in [13] — in their investigation the authors consider the follow automaton before the final $\varepsilon$-removal is done. Let us call this device *$\varepsilon$-follow automaton*. It turns out that the $\varepsilon$-follow automaton is superior to the other constructions considered. In particular, the number of $\varepsilon$-transitions asymptotically tends to zero, i.e., the $\varepsilon$-follow automaton approaches the follow-automaton.

Almost all of these results were obtained with the help of the framework of analytic combinatorics [32]. The idea to use this approach is quite natural. Recall

[d]The reader may have noticed the difference to the notion of *unambiguous* regular expressions from [9]. These require that for each word there is at most one sequence of positions of symbols in the regular expression that matches the word. Every regular language is denoted by some unambiguous regular expression.

[e]This of course depends on the viewpoint. From another perspective, regular expressions can be exponentially more succinct than DFAs, which is no longer true for one-unambiguous regular expressions.

that the number of regular expressions of a certain size measured by, e.g., alphabetic width, can be counted by using generating functions — for more involved measures, one has to use multivariate generating functions. To this end one transforms a grammar describing regular expressions such as, e.g., the grammar devised in [55], into a generating function. Since the grammar describes a combinatorial class, the generating function can be obtained by the *symbolic method* of [32], and the coefficients of the power series can be estimated to give approximations of the measure under consideration.

Finally, let us note that the results on the average size of automata depends on the probability distribution that is used for the average-case analysis. In [98] it was shown that the number of transitions of the position automaton is in $\Theta(n^2)$ under a distribution that is inspired from random binary search trees (BST-like model). To our knowledge, average case analysis under the BST-like model for other automata such as the follow automaton or the partial derivative automaton, has not been conducted so far.

## 4. From Finite Automata to Regular Expressions

There are a few classical algorithms for converting finite automata into equivalent regular expressions, namely

(1) the *algorithm based on Arden's lemma* [5, 24], and
(2) the *McNaughton-Yamada algorithm* [91], and
(3) the *state elimination technique* [17].

These procedures look different at first glance. We briefly explain the main idea of these approaches — for a detailed description along with an explanation of the differences between the methods, the reader is referred to [105]. There it is shown that all of the above approaches are more or less reformulations of the same underlying algorithmic idea, and they yield (almost) the same regular expressions.[f]

An algebraic approach to solve the conversion problem from finite automata to regular expressions is the *algorithm based on Arden's lemma* [5, 24]. It puts forward a set of language equations for a given finite automaton. Here, the $i$th equation describes the set $X_i$ of words $w$ such that the given automaton can go from the $i$th state to an accepting state on reading $w$. That system of equations can be resolved by eliminating the indeterminates $X_i$ using a method that resembles Gaussian elimination. But we work in a an algebraic structure different from a field, so for the elimination of variables, we have to resort to *Arden's lemma*:

**Lemma 14.** *Let $\Sigma$ be an alphabet, and let $K, L \subseteq \Sigma^*$, where $K$ does not contain the empty word $\varepsilon$. Then the set $K^*L$ is the unique solution to the language equation $X = K \cdot X + L$, where $X$ is the indeterminate.*

[f]Let us also mention that there is another algebraic algorithm from [24], which is based on the recursive decomposition of matrices into blocks. Here, the precise relation to the aforementioned algorithms remains to be investigated [105].

Now let us have a look on how Arden's lemma can be applied to our running example.

**Example 15.** *From the automaton depicted in Fig.* 2 *one reads off the equations*

$$X_0 = a \cdot X_1 + \varepsilon, \quad X_i = a \cdot X_{i+1} + b \cdot X_{i-1}, \quad \text{for } 1 \leq i < n, \text{ and } \quad X_n = b \cdot X_{n-1}.$$

*Substituting the right hand side of* $X_n$ *in the next to last equation and solving it by Arden's lemma results in* $X_{n-1} = (ab)^* b \cdot X_{n-2}$. *For short,* $X_{n-1} = r_1 \cdot b \cdot X_{n-2}$, *where* $r_i$ *is defined as in Example* 5. *Next this solution is substituted into the equation for* $X_{n-2}$. *Solving for* $X_{n-2}$ *gives us* $X_{n-2} = r_2 \cdot b \cdot X_{n-3}$. *Proceeding in this way up to the very first equation gives us* $X_0 = a \cdot r_{n-1} \cdot b \cdot X_0 + \varepsilon$. *The solution to the indeterminate* $X_0$ *is according to Arden's lemma* $(a \cdot r_{n-1} \cdot b)^* \cdot \varepsilon = r_n$, *by applying obvious simplifications. Hence, for instance, in case* $n = 6$ *we obtain* $(a(a(a(a(a(ab)^*b)^*b)^*b)^*b)^*b)^*$. □

The *McNaughton-Yamada algorithm* [91] maintains a matrix with regular expression entries, where the rows and columns are the states of the given automaton. The iterative algorithm uses a ranking on the state set, and proceeds in $n$ rounds, if $n$ is the number of states in the given automaton $A$. In the matrix $(a_{jk})_{j,k}$ computed in round $i$, the entry $a_{jk}$ is an expression describing the nonempty labels $w$ of computations of $A$ starting in $j$ and ending in $k$, such that none of the intermediate states of the computation is ranked higher than $i$. From these expressions, it is not difficult to obtain a regular expression describing $L(A)$.

**Example 16.** *Running the McNaughton-Yamada algorithm on the automaton depicted in Fig.* 2 *for* $n = 3$ *with the ranking* $3, 2, 1, 0$ *starts with the following matrix:*

$$\begin{array}{c} \\ 3 \\ 2 \\ 1 \\ 0 \end{array} \begin{array}{c} \begin{array}{cccc} 3 & 2 & 1 & 0 \end{array} \\ \begin{pmatrix} \emptyset & b & \emptyset & \emptyset \\ a & \emptyset & b & \emptyset \\ \emptyset & a & \emptyset & b \\ \emptyset & \emptyset & a & \emptyset \end{pmatrix} \end{array}.$$

*If* $(a_{jk})_{j,k}$ *denotes the matrix computed in round* $i$, *then the matrix* $(b_{jk})_{j,k}$ *for round* $i + 1$ *can be computed using the rule*

$$b_{jk} = a_{jk} + a_{ji}(a_{ii})^* a_{ik}\,.$$

*After the first round, the entry in the upper left corner of the matrix reads as* $\emptyset + \emptyset\emptyset^*\emptyset$. *It is of course helpful to simplify the intermediate regular expressions, by applying some obvious simplifications. As noted in* [91], *we can use in particular*

$$b_{ij} = (a_{ii})^* a_{ij} \quad \text{and} \quad b_{ji} = a_{ji}(a_{ii})^*.$$

*Then the matrix computed in the first round reads as*

$$\begin{pmatrix} \emptyset & b & \emptyset & \emptyset \\ a & ab & b & \emptyset \\ \emptyset & a & \emptyset & b \\ \emptyset & \emptyset & a & \emptyset \end{pmatrix},$$

*the one from the second round is*

$$\begin{pmatrix} b(ab)^*a & b(ab)^* & b(ab)^*b & \emptyset \\ (ab)^*a & (ab)^*ab & (ab)^*b & \emptyset \\ a(ab)^*a & a(ab)^* & a(ab)^*b & b \\ \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix},$$

*and the computation is continued in the same vein. Finally, the entry in the lower-right corner of the matrix reads as* $(a(a(ab)^*b)^*b)^*a(a(ab)^*b)^*b$*, and the desired regular expression describing* $L_3$ *is obtained by adding the empty word:* $\varepsilon + (a(a(ab)^*b)^*b)^*a(a(ab)^*b)^*b$. □

A few industrious readers, who have worked out the calculation of the previous example until the final matrix, may have observed that many of the intermediate expressions were actually not needed for the final result. Indeed, in a computer implementation [87, page 8] of the basic McNaughton-Yamada algorithm during the 1960s, the author notes: *"a basic fault of the method is that it generates such cumbersome and so numerous expressions initially."* Below we discuss how the generation of unnecessary sub-expressions can be avoided.

We now come to an algorithm that we describe in greater detail, namely the *state elimination algorithm* [17]. This procedure maintains an extended finite automaton, whose transitions are labeled with regular expressions, rather than alphabet symbols. The computation of an NFA $A$ can be thought of as reading the input word letter by letter, thereby nondeterministically changing its state with each letter in a way that is consistent with its transition table $\delta$. On reading a word $w \in \Sigma$, we say that the finite automaton $A$ and can go on input $w$ from state $j$ to state $k$, if there is a computation on input $w$ taking $A$ from state $j$ to $k$. Similarly, for a subset $U$ of the state set $Q$ of the automaton $A$, we say that $A$ can go on input $w$ from state $j$ *through* $U$ to state $k$, if there is a computation on input $w$ taking $A$ from state $j$ to $k$, without going through any state outside $U$, except possibly $j$ and $k$. With the *rôles* of $j$, $k$, and $U$ fixed as above, we now define the language $L_{jk}^U$ as the set of input words on which the automaton $A$ can go from $j$ to $k$ through $U$. The state elimination scheme fixes an ordering on the state set $Q$. Starting with $U = \emptyset$, regular expressions denoting the languages $L_{jk}^\emptyset$ for all pairs $(j, k) \in Q \times Q$ can be easily read off from the transition table of $A$. Now an important observation is that for each state $i \in Q \setminus U$ holds

$$L_{jk}^{U \cup \{i\}} = L_{jk}^U \cup L_{ji}^U \cdot (L_{ii}^U)^* \cdot L_{ik}^U.$$

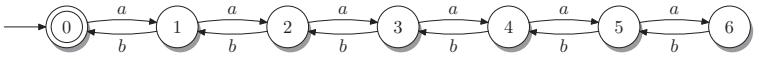



Fig. 6. A minimal DFA accepting the language $L_6 := L(r_6)$.

Letting $i$ run over all states according to the ordering, we can grow the set $U$ one by one, in each round computing the intermediate expressions $r_{jk}^{U\cup\{i\}}$ for all $j$ and $k$. The final regular expression is obtained by utilizing the fact $L(A) = \bigcup_{f\in F} L_{q_0 f}^{Q}$.

As observed already by McNaughton and Yamada [91], we have the inequality $\mathsf{awidth}(L_{jk}^{\emptyset}) \leq |\Sigma|$, and each round increases the alphabetic width of each intermediate sub-expression by a factor of at most 4. Another convenient trick is to modify the automaton, by adding a new initial state $s$ and a new final state $t$ to the automaton without altering the language, such that $t$ is the single final state, and there are no transitions entering $s$ or leaving $t$. Then $s$ and $t$ need not to be added to the set $U$. Instead, observe that $L(A) = L_{st}^{U}$, with $U = Q \setminus \{s, t\}$. We also note[g] that the computation of $r_{jk}^{U}$ needs to be carried out only for those $j$ and $k$ not in $U$. We thus obtain the following bound:

**Theorem 17.** *Let $n \geq 1$ and $A$ be an $n$-state NFA over alphabet $\Sigma$. Then alphabetic width $|\Sigma| \cdot 4^n$ is sufficient for a regular expression describing $L(A)$. Such an expression can be constructed by state elimination.*

In contrast, the state elimination algorithm might suddenly yield a much simpler regular expression once we change the ordering in which the states are eliminated. We illustrate the influence of the elimination ordering on a small example.

**Example 18.** *Consider our software buffer from Example 5 for $n = 6$. Let $L_n := L(r_n)$. For illustration, a minimal DFA for $L_6$ is depicted in Fig. 6. The two regular expressions*

$$(a(a(a(a(a(ab)^*b)^*b)^*b)^*b)^*b)^*$$

*and*

$$\begin{aligned}&\varepsilon + a(ab+ba)^*b\\ &\quad + a(ab+ba)^*aa\,(ab+ba+bb(ab+ba)^*aa+aa(ab+ba)^*bb)^*\,bb(ab+ba)^*b\end{aligned}$$

*both describe the language $L_6$. The first expression is obtained by eliminating the states in the order 6, 5, 4, 3, 2, 1, and 0, while the second expression is produced by the order 0, 2, 4, 6, 1, 5, and 3. Note that the expressions have very different structure. The first is much shorter, but has star height 6, while the second, and longer expression, has star height 2. Indeed, in [89] it was shown that the minimum star height among all regular expressions denoting $L_n$ equals $\lfloor \log(n+1) \rfloor$, so the star*

[g]The same trick applies for the McNaughton-Yamada algorithm: if the single initial and the single final state are not eliminated, we can erase the entries of the $i$th row and the $i$th column of the computed matrix in round $i$.

*height of the second expression is optimal. The authors suspect that this language family exhibits a trade-off in the sense that the regular expressions for $L_n$ cannot be simultaneously short and of low star height.* □

Perhaps the earliest reference mentioning the influence of the elimination ordering is from 1960. In [91], they proposed to identify the states that "bear the most traffic," i.e., those vertices in the underlying graph with the highest degree, and to eliminate these states at last. Since then, various heuristics for computing elimination orderings that yield short regular expressions have been proposed in the literature. In [81], a simple greedy heuristic was devised. It was proposed to assign a measure to each state, and this measure is recomputed each time when a state is eliminated. This measure indicates the priority in which the states are eliminated. Observe that eliminating a state tends to introduce new arcs in the digraph underlying the automaton. Thus we can order the states by a measure that is defined as the number of ingoing arcs times the number of outgoing arcs. In [26] a refined version of the same idea is proposed, which takes also the lengths of the intermediate expressions into account, instead of just counting the ingoing and outgoing arcs. Later, a different strategy for accounting the priority of a state was suggested: as measure function, simply take the number of cycles passing through a state. There are some automata, where this heuristic outperforms the one we previously described, but on most random DFAs the performance is comparable. For the heuristic based on counting the number of cycles, recomputing the measure after the elimination of each state does not make a big difference [95]. Another idea is to look for simple structures in finite automata, such as bridge states [59]. A bridge state typically exists if the language under consideration can be written as the concatenation of two nontrivial regular languages. Unfortunately, a random DFA almost surely contains no bridge states at all, as the number of states grows larger [95]. These and other heuristics were compared empirically on a large set of random DFAs as input in [53, 95]. Although there are also advanced strategies for choosing an elimination ordering, which have provable performance guarantees, the greedy heuristic from [26] performs best in most cases.

Beyond heuristics, we can use elimination orderings to prove nontrivial upper bounds on the conversion of DFAs over small alphabets into regular expressions. For the case of binary alphabets, a bound of $O(1.742^n)$ was given in [50], which was then improved to $O(1.682^n)$ in [28]. These bounds can be reached with state elimination by using appropriate elimination orderings.

**Theorem 19.** *Let $n \geq 1$ and $A$ be an $n$-state DFA over a binary alphabet. Then size $O(1.682^n)$ is sufficient for a regular expression describing $L(A)$.*

Similar bounds, but with somewhat larger constants in place of 1.682, can be derived for larger alphabets.[h] Moreover, the same holds for NFAs having a comparably low density of transitions.

[h]An improved bound of $O(1.588^n)$ was announced in a previous version of this paper, but a proof of that bound has eluded us so far.

We sketch how to establish a simpler upper bound than this, which after all gives $o(4^n)$ for all alphabets of constant size. To get things going, assume that we want to determine $L_{jk}^U$, and that the underlying sub-graph induced by $U$ falls apart into two mutually disconnected sub-graphs $A$ and $B$. Then on reading a word $w$, the automaton goes from $j$ to $k$ *either* through $A$ *or* through $B$, and thus $L_{jk}^U = L_{jk}^A \cup L_{jk}^B$, and this is reflected by the regular expressions computed using state elimination. In particular, if the sub-graph induced by $U$ is an independent set, i.e., a set of isolated vertices, in the underlying graph, then $L_{jk}^U = \bigcup_{i \in U} L_{jk}^{\{i\}}$. In this case, the blow-up factor incurred by eliminating $U$ is linear in $|U|$, instead of exponential in $|U|$. For a DFA $A$ over constant alphabet, the underlying graph has a linear number of edges. It is known that such graphs have an independent set of size $cn$, where $c$ is a constant depending on the number of edges. Suppose that $U$ is such an independent set. Then we partition the state set of $A$ into an "easy" part $U$ and a "hard" part $Q \setminus U$. Eliminating $U$ increases the size of the intermediate expressions by a factor linear in $|U|$. Thereafter, eliminating the remaining $(1-c)n$ states may incur a size blow-up by a factor of $4^{(1-c)n}$. Altogether, this gives a regular expression of alphabetic width in $|\Sigma| \cdot o(4^n)$ for $L(A)$.

Let us again take a look at an example.

**Example 20.** *For illustrating the above said, consider the language*

$$L_3 = (a_1 b_1)^* \sqcup\!\sqcup (a_2 b_2)^* \sqcup\!\sqcup (a_3 b_3)^*,$$

*where the* interleaving*, or* shuffle*, of two languages $L_1$ and $L_2$ over alphabet $\Sigma$ is*

$$L \sqcup\!\sqcup M = \{\, w \in \Sigma^* \mid w \in x \sqcup\!\sqcup y \text{ for some } x \in L \text{ and } y \in M \,\},$$

*and the interleaving $x \sqcup\!\sqcup y$ of two words $x$ and $y$ is defined as the set of all words of the form $x_1 y_1 x_2 y_2 \cdots x_n y_n$, where $x = x_1 x_2 \cdots x_n$, $y = y_1 y_2 \cdots y_n$ with $x_i, y_i \in \Sigma^*$, for $n \geq 1$ and $1 \leq i \leq n$. Note that in this definition, some of the sub-words $x_i$ and $y_i$ can be empty.*

*The language $L_3$ can be accepted by a partial DFA over the state set $\{0,1\}^3$, and whose transition function is given such that input $a_i$ sets the $i$th bit left of the rightmost bit of the current state from* 0 *to* 1*, and input $b_i$ resets the $i$th bit, again counting from right to left, of the current state from* 1 *to* 0*. All other transitions are undefined. The initial state is* 000*, which is also the single final state. Notice that the graph underlying this automaton is the* 3*-dimensional cube, with* 8 *vertices — see Fig.* 7*. Generalizing this example to $d \geq 3$, the underlying graph of $L_d$ is the $d$-dimensional hypercube, with $2^d$ many vertices.*

*It is well known that the $d$-dimensional hypercube is* 2*-colorable, and thus has an independent set that contains at least half of the vertices. Eliminating this independent set before the other vertices yields a regular expression of alphabetic width $O(n \cdot 2^n)$, which is way better than the trivial bound of $O(4^n)$.* □

We present another application of this idea. Planar finite automata are a special case of finite automata, which were first studied in [10]. To convert a planar finite

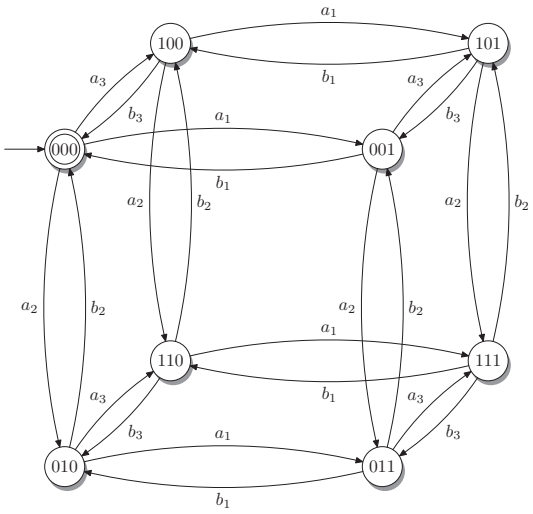



Fig. 7. Automaton accepting the language $L_3 = (a_1b_1)^* \sqcup\!\sqcup (a_2b_2)^* \sqcup\!\sqcup (a_3b_3)^*$. The underlying graph is the 3-dimensional cube.

automaton into a regular expression, one can look for a small set of vertices, whose removal leaves to mutually disconnected sub-graphs with vertex sets $A$ and $B$. Then again, we have $L^U_{jk} = L^A_{jk} \cup L^B_{jk}$, and this is reflected by the regular expressions computed by state elimination. Since the sub-graphs induced by $A$ and $B$ are again planar, one can apply the trick recursively. Also for this special case, tight upper and lower bounds were found recently [31, 49, 52].

**Theorem 21.** *Let $n \geq 1$ and $A$ be an $n$-state planar DFA or NFA over alphabet $\Sigma$. Then size $|\Sigma| \cdot 2^{O(\sqrt{n})}$ is sufficient for a regular expression describing $L(A)$. Such an expression can be constructed by state elimination.*

Taking this idea again a step further, one can arrive at a parametrization where the conversion problem from finite automata to regular expressions is fixed-parameter tractable, in the sense that the problem is exponential in that parameter, but not in the size of the input. Recall that we have introduced the concept of cycle rank of a digraph in the course of discussing the star height in Sec. 2. Now for a digraph $D$, let $D^{\mathrm{sym}}$ denote the symmetric digraph obtained by replacing each arc in $D$ with a pair of anti-parallel arcs. The *undirected cycle rank* of $D$ is defined as the cycle rank of $D^{\mathrm{sym}}$. If the conversion problem from finite automata to regular expressions is parametrized by the undirected cycle rank of the given automaton, one can prove the following bound [52]:

**Theorem 22.** *Let $n \geq 1$ and $A$ be an $n$-state DFA or NFA over alphabet $\Sigma$, whose underlying digraph is of undirected cycle rank at most $c$, for some $c \geq 1$. Then size $|\Sigma| \cdot 4^c \cdot n$ is sufficient for a regular expression describing $L(A)$. Such an expression can be constructed by state elimination.*

We note that fixed-parameter tractability also holds in the sense of computational complexity, since computing the undirected cycle rank is fixed-parameter tractable, see, e.g., [104].

But in the general case, the exponential blow-up when moving from finite automata to regular expressions is inherent, that is, independent of the conversion method. Already in the 1970s the existence of languages $L_n$ was shown, that admit $n$-state finite automata, but require regular expressions of alphabetic width at least $2^{n-1}$, for all $n \geq 1$, see [30]. Their witness language is over an alphabet of growing size, which is quadratic in the number of states. Their proof technique was tailored to the witness language involved. The question whether a comparable size blow-up can also occur for constant alphabet size [31] was settled only a few years ago. The answer was provided around the same time by two independent groups of researchers, who worked with different proof techniques, and gave different examples [38, 49].

How are such lower bounds established? We shall describe a general method, which has been used to prove lower bounds on regular expression size in various contexts [37,49,51,61]. For this purpose, a more convenient formulation of Lemma 4 is the *star height lemma*, which reads as follows:

**Lemma 23.** *Let $L$ be a regular language. Then $\mathsf{awidth}(L) \geq 2^{\Omega(\mathsf{height}(L))}$.*

That is, the minimum regular expression size of a regular language is at least exponential in the minimum required star height. But now this looks as if we have replaced one evil with another, since determining the star height is eminently difficult in general [74]. But there is an important special case, in which the star height can be determined more easily: a *partial* deterministic finite automaton is called bideterministic, if it has a single final state, and if the NFA obtained by reversing all transitions and exchanging the rôles of initial and final state is again a partial DFA — notice that, by construction, this NFA in any case accepts the reversed language. A regular language $L$ is *bideterministic* if there exists a bideterministic finite automaton accepting $L$. These languages form a proper subclass of the regular languages. For these languages, *McNaughton's Theorem* [88] states that the star height is equal to the cycle rank of the digraph underlying the minimal partial DFA.

**Example 24.** *Define the languages $K_m = \{ w \in \{a,b\}^* \mid |w|_a \equiv 0 \mod m \}$ and $L_n = \{ w \in \{a,b\}^* \mid |w|_b \equiv 0 \mod n \}$. For simplicity, assume $m \leq n$. It is straightforward to construct deterministic finite automata with $m$ states (with $n$ states, respectively) arranged in a directed cycle describing the languages $K_m$ and $L_n$, respectively. By applying the standard product construction on these automata, we obtain a deterministic finite automaton $A$ accepting the language $K_m \cap L_n$. The digraph underlying automaton $A$ is the directed discrete torus. This digraph can be described as the Cartesian graph product of two directed cycles, see Fig. 8 for illustration. The cycle rank of the $(m \times n)$-torus is equal to $m$ if $m = n$, and equal*

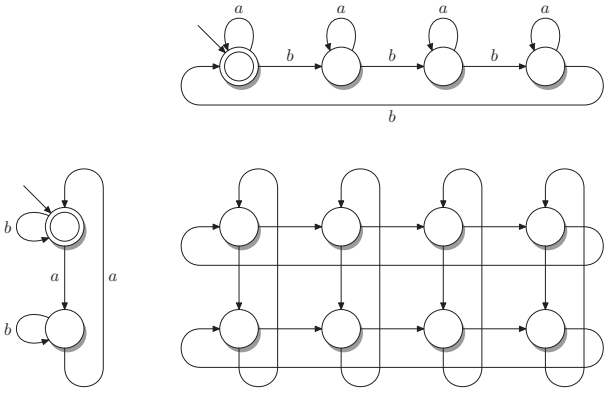



Fig. 8. Drawing of the discrete directed $(m \times n)$-torus in the case where $m = 2$ and $n = 4$, induced by the automata for the languages $K_m$ and $L_n$.

*to $m + 1$ otherwise* [49]. *It is easily observed that the automaton $A$ is bideterministic, hence the star height of $L(A)$ coincides with the cycle rank of its underlying digraph. By invoking the star height lemma, we can derive a lower bound of $2^{\Omega(m)}$ on the minimum regular expression size required for $L_m \cap K_n$.* □

For the succinctness gap between DFAs and regular expressions over binary alphabets, a lower bound of $2^{\Omega(\sqrt{n/\log n})}$ was reported in [38], while a parallel effort [49] resulted in an asymptotically tight lower bound of $2^{\Omega(n)}$. We have the following result:

**Theorem 25.** *Let $n \geq 1$ and $A$ be an $n$-state DFA or NFA over alphabet $\Sigma$. Then size $|\Sigma| \cdot 2^{\Theta(n)}$ is sufficient and necessary in the worst case for a regular expression describing $L(A)$. This already holds for alphabets with at least two letters.*

Recall that the notation $2^{\Theta(n)}$ implies a lower bound of $c^n$, for some $c > 1$. The hidden constant in the lower bound for binary alphabets is much smaller compared to the lower bound of $2^{n-1}$ previously obtained in [30] for large alphabets. The upper bound from Theorem 19 implies that $c$ can be at most 1.588 for alphabets of size two. Narrowing down the interval for the best possible $c$ for various alphabet sizes is a challenge for further research.

We turn our attention to some interesting special cases of regular languages, namely the *finite* and the *unary* regular languages. Here, the situation is significantly different, as we can harness specialized techniques which are more powerful than state elimination. Also, finite and unary languages have star height at most 1, and thus more tailored techniques than the star height lemma are needed to establish lower bounds. Indeed, the case of finite languages was already addressed in the very first paper on the descriptional complexity of regular expressions [30]. They give a specialized conversion algorithm for finite languages, which is different from the state elimination algorithm. Their results imply that every $n$-state DFA

accepting a finite language can be converted into an equivalent regular expression of size $n^{O(\log n)}$. The method is quite interesting, since it is not based on state elimination, but rather on a clever application of the repeated squaring trick. They also provide a lower bound of $n^{\Omega(\log \log n)}$ when using an alphabet of size $O(n^2)$. The challenge of tightening this gap was settled more than thirty years later in [54], where a lower bound technique from communication complexity is adapted, which originated in the study of monotone circuit complexity.

**Theorem 26.** *Let $n \geq 1$ and $A$ be an $n$-state DFA or NFA over alphabet $\Sigma$ accepting a finite language. Then size $|\Sigma| \cdot n^{\Theta(\log n)}$ is sufficient and necessary in the worst case for a regular expression describing $L(A)$. This still holds for constant alphabets with at least two letters.*

The case of unary languages was discussed in [35,86,110]. Here the main idea is that one can exploit the simple cycle structure of unary DFAs and of unary NFAs in Chrobak normal form [23]. In the case of NFAs, elementary number theory helps to save a logarithmic factor of the quadratic upper bound [35]. The main results are summarized in the following theorem.

**Theorem 27.** *Let $n \geq 1$ and $A$ be an $n$-state DFA over a unary alphabet. Then size $\Theta(n)$ is sufficient and necessary in the worst case for a regular expression describing $L(A)$. When considering NFAs, the upper bound changes to $O(n^2/\log n)$.*

The tight bounds for the conversion of unary NFAs to regular expressions thus remain to be determined.

We return again to the case of one-unambiguous regular languages. In the seminal paper about one-unambiguous regular expressions [15], it was shown that a DFA accepting a one-unambiguous regular language can be converted in exponential time into an equivalent one-unambiguous regular expression. Although they do not explicitly state an upper bound on the size of this regular expression, the following bound can be readily deduced from the constructive proof of Thm. G in [15]. The construction starts from the given DFA $A$. It computes the final expression recursively from subautomata of $A$ with fewer transitions, and the algorithm spawns in each recursive step at most $|\Sigma| + 1$ branches, where $\Sigma$ denotes the input alphabet. The terminating cases for the recursion correspond to one-unambiguous regular expressions of constant size each. One thus obtains the following bound:

**Theorem 28.** *Let $n \geq 1$ and $A$ be an $n$-state DFA accepting a one-unambiguous regular language over an alphabet of size $k$. Then size $O\left((k+1)^{kn}\right)$ is sufficient for a one-unambiguous regular expression describing $L(A)$.*

Regarding a lower bound, we note that the witness languages from [30, 39, 49] showing an exponential succinctness gap between DFAs and regular expressions are not one-unambiguous, cf. [82]. But in the seminal paper on one-unambiguous regular expressions [15], they also mention a family of one-unambiguous languages whose

minimal DFA is exponentially more succinct than a minimal one-unambiguous regular expression. Namely, the language $\Sigma^* a_1 a_2 \dots a_n$ — with $\Sigma$ denoting the input alphabet and $\{a_1, a_2, \dots, a_n\} \subseteq \Sigma$ — would incur an exponential blow-up when moving from a DFA to an equivalent one-unambiguous regular expression. Since that language obviously admits a regular expression whose size is linear in the size of the minimal DFA, we are in need for a quite different proof technique. Remarkably, a proof of the claimed lower bound was provided only recently in [82].

**Theorem 29.** *Let $n \geq 1$ and $A$ be an $n$-state DFA accepting a one-unambiguous regular language. Then size at least $2^{\Omega(n)}$ is necessary in the worst case for a one-unambiguous regular expression describing $L(A)$.*

Also for the conversion of deterministic finite automata into equivalent one-unambiguous regular expressions, the tight upper and lower bounds on the succinctness gap need to be investigated. Another witness language for the exponential gap is given in what follows.

**Example 30.** *For $n \geq 0$, let $L^{[n]}$ be the language denoted by the regular expression $((a+\epsilon)^n b)^* a^n$. This language is taken from* [82]*, where it is shown that $L^{[n]}$ can be accepted by an $(n+1)$-state DFA, while every equivalent one-unambiguous regular expression has size $2^{\Omega(n)}$. The deterministic automaton accepting the language $L^{[3]}$ is depicted in Fig.* 9*. For the language $L^{[3]}$, a one-unambiguous regular expression of alphabetic width* 29 *is given by*

$$b^*a\left(bb^*\right)^* aa\left(bb^*a\left(bb^*\right)^* a\right)^* aa\left(bb^*a\left(bb^*\right)^* aa\left(bb^*a\left(bb^*\right)^* a\right)^* a\right)^* a.$$

*This one-unambiguous regular expression is derived using the construction given in Thm. G of* [15]*; for this special case, the algorithm essentially boils down to the recursive language equation $L^{[n+1]} = L^{[n]} a (b L^{[n]})^* a$, with $L^{[0]} = b^*$. With ordinary regular expressions, we can describe this language more succinctly as*

$$((a+\epsilon)(a+\epsilon)(a+\epsilon)b)^* aaa,$$

*whose alphabetic width is only* 7*. The numbers witnessing the succinctness gap quickly become more striking for larger values of $n$: for $n = 5$, the above recurrence yields a one-unambiguous regular expression of alphabetic width* 125*, while there is a regular expression of alphabetic width only* 11*.* □

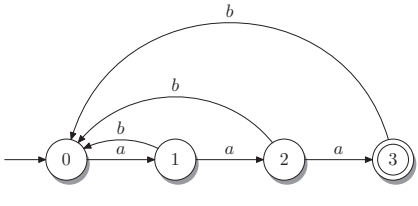



Fig. 9. A four state DFA accepting the language $L^{[3]}$.

Another interesting open problem concerns the succinctness gap between regular expressions and one-unambiguous regular expressions: the only known technique for converting a regular expression denoting a one-unambiguous language into an equivalent one-unambiguous regular expression entails the conversion of the given expression to a DFA, followed by converting the DFA into a one-unambiguous regular expression. Recall from Theorem 11 that moving from a regular expression to a DFA may incur an exponential blow-up, and the above theorem shows that the second step is exponential as well. It remains open whether such a double-exponential blow-up is inherent when moving from regular expressions to one-unambiguous regular expressions, cf. [25].

Finally, let us note that the conversion problem has been studied also for a few other special cases of finite automata. Examples include finite automata whose underlying digraph is an acyclic series-parallel digraph [96], Thompson digraphs [41], and Glushkov automata [18].



## Acknowledgments

Thanks to Sebastian Jakobi for helpful comments and suggestions on an earlier draft of this paper, and to Katja Losemann for providing a full version of [82].

## References

[1] A. Aho, R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques, and Tools* (Addison Wesley, 1986).

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms* (Addision-Wesley, 1974).

[3] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling* (Prentice-Hall, 1972).

[4] V. Antimirov, Partial derivatives of regular expressions and finite automaton constructions, *Theoretical Computer Science* **155** (March 1996) 291–319.

[5] D. N. Arden, Delayed-logic and finite-state machines, *Proceedings of the* 1*st and* 2*nd Annual Symposium on Switching Theory and Logical Design*, ed. T. Mott (American Institute of Electrical Engineers, New York, Detroit, Michigan, USA, October 1961), pp. 133–151.

[6] A. Asperti, C. S. Coen and E. Tassi, Regular expressions, *au point*, arXiv:1010.2604v1 [cs.FL] (October 2010).

[7] G. Berry and R. Sethi, From regular expressions to deterministic automata, *Theoretical Computer Science* **48**(3) (1986) 117–126.

[8] P. Bille and M. Thorup, Regular expression matching with multi-strings and intervals, *Proceedings of the* 21*st Annual ACM-SIAM Symposium on Discrete Algorithms*, ed. M. Charikar (Society for Industrial and Applied Mathematics, Austin, Texas, USA, October 2010), pp. 1297–1308.

[9] R. Book, S. Even, S. Greibach and G. Otto, Ambiguity in graphs and expressions, *IEEE Transactions on Computers* **20**(2) (1971) 149–153.

[10] R. V. Book and A. K. Chandra, Inherently nonplanar automata, *Acta Informatica* **6**(1) (1976) 89–94.

[11] S. Broda, A. Machiavelo, N. Moreira and R. Reis, On the average number of states of partial derivative automata, *Proceedings of the* 14*th International Conference*

*Developments in Language Theory*, eds. Y. Gao, H. Lu, S. Seki and S. Yu, *LNCS* (Springer, London, Ontario, Canada, August 2010), pp. 112–123.

[12] S. Broda, A. Machiavelo, N. Moreira and R. Reis, On the average size of Glushkov and partial derivative automata, *International Journal of Foundations of Computer Science* **23** (August 2012) 969–984.

[13] S. Broda, A. Machiavelo, N. Moreira and R. Reis, A hitchhiker's guide to descriptional complexity through analytic combinatorics, *Theoretical Computer Science* **528** (April 2014) 85–100.

[14] A. Brüggemann-Klein, Regular expressions into finite automata, *Theoretical Computer Science* **120** (1993) 197–213.

[15] A. Brüggemann-Klein and D. Wood, One-unambiguous regular languages, *Information and Computation* **140**(2) (1998) 229–253.

[16] J. A. Brzozowski, Derivatives of regular expressions, *Journal of the ACM* **11** (1964) 481–494.

[17] J. A. Brzozowski and E. J. McCluskey, Signal flow graph techniques for sequential circuit state diagrams, *IEEE Transactions on Computers* **C-12** (April 1963) 67–76.

[18] P. Caron and D. Ziadi, Characterization of Glushkov automata, *Theoretical Computer Science* **233** (February 2000) 75–90.

[19] J.-M. Champarnaud, F. Ouardi and D. Ziadi, Normalized expressions and finite automata, *International Journal of Algebra and Computation* **17**(1) (2007) 141–154.

[20] J.-M. Champarnaud and D. Ziadi, Canonical derivatives, partial derivatives and finite automaton constructions, *Theoretical Computer Science* **289** (October 2002) 137–163.

[21] C. Chang and R. Paige, From regular expressions to DFA's using compressed NFA's, *Proceedings of the* 3*rd Annual Symposium on Combinatorial Pattern Matching*, eds. A. Apostolico, M. Chrochemore, Z. Galil and U. Manber, *LNCS* (Springer, Tucson, Arizon, USA, April–May 1992), pp. 90–110.

[22] H. Chen, Finite automata of expressions in the case of star normal form and one-unambiguity, Technical Report ISCAS-LCS-10-11, Chinese Academy of Sciences, Institute of Software, State Key Laboratory of Computer Science, Beijing 100190 China (2010).

[23] M. Chrobak, Finite automata and unary languages, *Theoretical Computer Science* **47** (June 1986) 149–158.

[24] J. H. Conway, *Regular Algebra and Finite Machines* (Chapman and Hall, 1971).

[25] W. Czerwinski, C. David, K. Losemann and W. Martens, Deciding definability by deterministic regular expressions, *Proceedings of the* 16*th International Conference on Foundations of Software Science and Computation Structures*, ed. F. Pfenning, *LNCS* (Springer, Rome, Italy, March 2013), pp. 289–304.

[26] M. Delgado and J. Morais, Approximation to the smallest regular expression for a given regular language, *Proceedings of the* 9*th Conference on Implementation and Application of Automata*, eds. M. Domaratzki, A. Okhotin, K. Salomaa and S. Yu, *LNCS* (Springer, Kingston, Ontario, Canada, July 2004), pp. 312–314.

[27] D.-Z. Du and K.-I. Ko, *Problem Solving in Automata, Languages, and Complexity* (John Wiley & Sons, 2001).

[28] K. Edwards and G. Farr, Improved upper bounds for planarization and series-parallelization of degree-bounded graphs, *The Electronic Journal of Combinatorics* **19**(2) (2012) p. #P25.

[29] L. C. Eggan, Transition graphs and the star height of regular events, *Michigan Mathematical Journal* **10** (1963) 385–397.

[30] A. Ehrenfeucht and H. P. Zeiger, Complexity measures for regular expressions, *Journal of Computer and System Sciences* **12** (April 1976) 134–146.

[31] K. Ellul, B. Krawetz, J. Shallit and M.-W. Wang, Regular expressions: New results and open problems, *Journal of Automata, Languages and Combinatorics* **9**(2/3) (2004) 233–256.

[32] P. Flajolet and R. Sedgewick, *Analytic Combinatorics* (Cambridge University Press, 2009).

[33] Y. Gao, K. Salomaa and S. Yu, Transition complexity of incomplete DFAs, *Fundamenta Informaticae* **110** (September 2011) 143–158.

[34] P. García, D. López, J. Ruiz and G. I. Álvarez, From regular expressions to smaller NFAs, *Theoretical Computer Science* **412** (2011) 5802–5807.

[35] P. Gawrychowski, Chrobak normal form revisited, with applications, *Proceedings of the* 16*th Conference on Implementation and Application of Automata*, eds. B. Bouchou-Markhoff, P. Caron, J.-M. Champarnaud and D. Maurel, *LNCS* (Springer, Blois, France, July 2011), pp. 142–153.

[36] V. Geffert, Translation of binary regular expressions into nondeterministic $\epsilon$-free automata with $O(n \log n)$ transitions, *Journal of Computer and System Sciences* **66** (May 2003) 451–472.

[37] W. Gelade, Succintness of regular expressions with interleaving, intersection, and counting, *Theoretical Computer Science* **411**(31–33) (2010) 2987–2998.

[38] W. Gelade and F. Neven, Succinctness of complement and intersection of regular expressions, *Proceedings of the* 25*th International Symposium on Theoretical Aspects of Compter Science*, eds. S. Albers and P. Weil, *Leibniz International Proceedings in Informatics* **1** (Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Bordeaux, France, February 2008), pp. 325–336.

[39] W. Gelade and F. Neven, Succinctness of the complement and intersection of regular expressions, *ACM Transactions on Computational Logic* **13** (January 2012) p. No. 4.

[40] D. Giammarresi, R. Montalbano and D. Wood, Block-deterministic regular languages, *Proceedings of the* 17*th Italian Conference on Theoretical Computer Science*, eds. A. Restivo, S. R. D. Rocca and L. Roversi, *LNCS* (Springer, Torino, Italy, October 2001), pp. 184–196.

[41] D. Giammarresi, J.-L. Ponty, D. Wood and D. Ziadi, A characterization of thompson digraphs, *Discrete Applied Mathematics* **134** (January 2004) 317–337.

[42] D. Giammarresi, J.-L. Pony and D. Wood, Thompson languages, *Jewels are Forever: Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, eds. J. Karhumäki, H. Maurer, G. Păun and G. Rozenberg (Springer, 1999), pp. 16–24.

[43] V. M. Glushkov, The abstract theory of automata, *Russian Mathematics Surveys* **16** (1961) 1–53.

[44] J. Goldstine, M. Kappes, C. M. R. Kintala, H. Leung, A. Malcher and D. Wotschke, Descriptional complexity of machines with limited resources, *Journal of Universal Computer Science* **8**(2) (2002) 193–234.

[45] J. Goldstine, C. M. R. Kintala and D. Wotschke, On measuring nondeterminism in regular languages, *Information and Computation* **86**(2) (1990) 179–194.

[46] J. Goldstine, H. Leung and D. Wotschke, On the relation between ambiguity and nondeterminism in finite automata, *Information and Computation* **100** (1992) 261–170.

[47] B. Groz, S. Maneth and S. Staworko, Deterministic regular expressions in linear time, *Proceedings of the* 31*st ACM SIGMOND-SIGACT-SIGART Symposium on Principles of Database Systems*, eds. M. Benedikt, M. Krötzsch and M. Lenzerini (ACM, Scottsdale, AZ, USA, May 2012), pp. 49–60.

[48] H. Gruber and S. Gulan, Simplifying regular expressions, *Proceedings of the* 4*th International Conference Language and Automata Theory and Applications*, eds. A. H. Dediu, H. Fernau and C. Martín-Vide, *LNCS* (Springer, Trier, Germany, May 2010), pp. 285–296.

[49] H. Gruber and M. Holzer, Finite automata, digraph connectivity, and regular expression size, *Proceedings of the* 35*th International Colloquium on Automata, Languages and Propgramming*, eds. L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir and I. Walkuwiewicz, *LNCS* (Springer, Reykjavik, Iceland, July 2008), pp. 39–50.

[50] H. Gruber and M. Holzer, Provably shorter regular expressions from deterministic finite automata (extended abstract), *Proceedings of the* 12*th International Conference Developments in Language Theory*, eds. M. Ito and M. Toyama, *LNCS* (Springer, Kyoto, Japan, September 2008), pp. 383–395.

[51] H. Gruber and M. Holzer, Tight bounds on the descriptional complexity of regular expressions, *Proceedings of the* 13*th International Conference Developments in Language Theory*, eds. V. Diekert and D. Nowotka, *LNCS* (Springer, Stuttgart, Germany, June–July 2009), pp. 276–287.

[52] H. Gruber and M. Holzer, Provably shorter regular expressions from finite automata, *International Journal of Foundations of Computer Science* **24** (November 2013) 1255–1279.

[53] H. Gruber, M. Holzer and M. Tautschnig, Short regular expressions from finite automata: Empirical results, *Proceedings of the* 14*th Conference on Implementation and Application of Automata*, ed. S. Maneth, *LNCS* (Springer, Sydney, Australia, July 2009), pp. 188–197.

[54] H. Gruber and J. Johannsen, Tight bounds on the descriptional complexity of regular expressions, *Proceedings of the* 11*th Conference Foundations of Software Science and Computational Structures*, ed. R. Amadio, *LNCS* (Springer, Budapest, Hungary, March–April 2008), pp. 273–286.

[55] H. Gruber, J. Lee and J. Shallit, Enumerating regular expressions and their languages, arXiv:1204.4982 [cs.FL] (April 2012).

[56] S. Gulan and H. Fernau, An optimal comstruction of finite automata from regular expressions, *Proceedings of the* 28*th Conference on Foundations of Software Technology and Theoretical Compter Science*, eds. R. Hariharan, M. Mukund and V. Vinay, *Dagstuhl Seminar Proceedings* **08002** (Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Bangalore, India, December 2008), pp. 211–222.

[57] C. Hagenah and A. Muscholl, Computing $\epsilon$-free NFA from regular expressions in $O(n \log^2(n))$ time, *RAIRO–Informatique théorique et Applications/Theoretical Informatics and Applications* **34** (September–October 2000) 257–277.

[58] Y.-S. Han and D. Wood, Generalizations of 1-deterministic regular languages, *Information and Computation* **206**(9–10) (2008) 1117–1125.

[59] Y.-S. Hand and D. Wood, Obtaining shorter regular expressions from finite-state automata, *Theoretical Computer Science* **370** (February 2007) 110–120.

[60] K. Hashiguchi, Algorithms for determining the relative star height and star height, *Information and Computation* **78**(2) (1988) 124–169.

[61] M. Holzer and S. Jakobi, Chop operations and expressions: Descriptional complexity considerations, *Proceedings of the* 15*th International Conference Developments in Language Theory*, eds. G. Mauri and A. Leporati, *LNCS* (Springer, Milan, Italy, July 2011), pp. 264–275.

[62] M. Holzer and M. Kutrib, Nondeterministic finite automata — Recent results on the descriptional and computational complexity, *International Journal of Foundations of Computer Science* **20**(4) (2009) 563–580.
[63] M. Holzer and M. Kutrib, The complexity of regular(-like) expressions, *Proceedings of the* 14*th International Conference Developments in Language Theory*, eds. Y. Gao, H. Lu, S. Seki and S. Yu, *LNCS* (Springer, London, Ontario, Canada, August 2010), pp. 16–30.
[64] M. Holzer and M. Kutrib, Descriptional complexity — An introductory survey, *Scientific Applications of Language Methods*, ed. C. Martín-Vide (World Scientific, 2010), pp. 1–58.
[65] M. Holzer and M. Kutrib, Descriptional complexity of (un)ambiguous finite state machines and pushdown automata, *Proceedings of the* 4*th Workshop on Reachability Problems*, eds. A. Kucera and I. Potapov, *LNCS* (Springer, Brno, Czech Republic, August 2010), pp. 1–23.
[66] M. Holzer and M. Kutrib, Descriptional and computational complexity of finite automata — A survey, *Information and Computation* **209** (March 2011) 456–470.
[67] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, 1979).
[68] D. Hovland, The inclusion problem for regular expressions, *Journal of Computer and System Sciences* **78**(6) (2012) 1795–1813.
[69] J. Hromkovič, Descriptional complexity of finite automata: Concepts and open problems, *Journal of Automata, Languages and Combinatorics* **7**(4) (2002) 519–531.
[70] J. Hromkovič, J. Karhumäki, H. Klauck, G. Schnitger and S. Seibert, Communication complexity method for measuring nondeterminism in finite automata, *Information and Computation* **172**(2) (2002) 202–217.
[71] J. Hromkovič and G. Schnitger, NFAs with and without $\epsilon$-transitions, *Proceedings of the* 32*nd International Colloquium Automata, Languages and Programming*, eds. L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi and M. Yung, *LNCS* (Springer, Lisbon, Portugal, July 2005), pp. 385–396.
[72] J. Hromkovič, S. Seibert and T. Wilke, Translating regular expressions into small $\epsilon$-free automata, *Journal of Computer and System Sciences* **62** (June 2001) 565–588.
[73] L. Ilie and S. Yu, Follow automata, *Information and Computation* **186** (October 2003) 140–162.
[74] D. Kirsten, Distance desert automata and the star height problem, *RAIRO–Informatique théorique et Applications/Theoretical Informatics and Applications* **39**(3) (2005) 455–509.
[75] S. C. Kleene, Representation of events in nerve nets and finite automata, *Automata studies*, eds. C. E. Shannon and J. McCarthy, *Annals of mathematics studies* **34** (Princeton University Press, 1956), pp. 2–42.
[76] E. Leiss, The complexity of restricted regular expressions and the synthesis problem for finite automata, *Journal of Computer and System Sciences* **23** (December 1981) 348–254.
[77] H. Leung, On finite automata with limited nondeterminism, *Acta Informatica* **35**(7) (1998) 595–624.
[78] H. Leung, Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata, *SIAM Journal on Computing* **27** (August 1998) 1073–1082.
[79] H. Leung, Descriptional complexity of NFA of different ambiguity, *International Journal of Foundations of Computer Science* **16** (October 2005) 975–984.

[80] Y. Lifshits, A lower bound on the size of $\epsilon$-free NFA corresponding to a regular expression, *Information Processing Letters* **85** (March 2003) 293–299.

[81] S. Lombardy, Y. Régis-Gianas and J. Sakarovitch, Introducing VAUCANSON, *Theoretical Computer Science* **328** (November 2004) 77–96.

[82] K. Losemann, W. Martens and M. Niewerth, Descriptional complexity of deterministic regular expressions, *Proceedings of the* 37*th International Symposium on Mathematical Foundations of Computer Science*, eds. B. Rovan, V. Sassone and P. Widmayer, *LNCS* (Springer, Bratislava, Slovakia, August 2012), pp. 643–654.

[83] E. Maia, N. Moreira and R. Reis, Incomplete transition complexity of basic operations on finite languages, *Proceedings of the* 18*th International Conference on Implementation and Application of Automata*, ed. S. Konstantinidis, *LNCS* (Springer, Halifax, Nova Scotia, Canada, July 2013), pp. 349–356.

[84] E. Maia, N. Moreira and R. Reis, Incomplete transition complexity of some basic operations, *Proceedings of the* 39*th International Conference on Current Trends in Theory and Practice of Computer Science*, eds. P. v. Emde Boas, F. C. A. Groen, G. F. Italiano, J. R. Nawrocki and H. Sack, *LNCS* (Springer, Špindlerøuv Mlýn, Czech Republic, January 2013), pp. 319–331.

[85] Z. Manna, *Mathematical Theory of Computation* (McGraw-Hill, 1974).

[86] A. Martinez, Efficient computation of regular expressions from unary NFAs, *Pre-Proceedings of the* 4*th Workshop on Descriptional Complexity of Formal Systems*, eds. J. Dassow, M. Hoeberechts, H. Jürgensen and D. Wotschke, *Report No.* **586** (Department of Computer Science, The University of Western Ontario, Canada, London, Ontario, Canada, August 2002), pp. 216–230.

[87] H. V. McIntosh, REEX: A CONVERT program to realize the McNaughton-Yamada analysis algorithm, Tech. Rep. AIM-153, MIT Artificial Intelligence Laboratory (January 1968).

[88] R. McNaughton, The loop complexity of pure-group events, *Information and Control* **11** (July–August 1967) 167–176.

[89] R. McNaughton, The loop complexity of regular events, *Information Sciences* **1** (1969) 305–328.

[90] R. McNaughton, *Elementary Computability, Formal Languages, and Automata* (Prentice-Hall, 1982).

[91] R. McNaughton and H. Yamada, Regular expressions and state graphs for automata, *IRE Transactions on Electronic Computers* **EC-9**(1) (1960) 39–47.

[92] A. R. Meyer and M. J. Fischer, Economy of description by automata, grammars, and formal systems, *Proceedings of the* 12*th Annual Symposium on Switching and Automata Theory* (IEEE Computer Society Press, 1971), pp. 188–191.

[93] B. G. Mirkin, An algorithm for constructing a base in a language of regular expressions, *Engineering Cybernetics* **5** (September–October 1966) 110–116.

[94] F. R. Moore, On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata, *IEEE Transaction on Computing* **C-20** (1971) 1211–1219.

[95] N. Moreira, D. Nabais and R. Reis, State elimination ordering strategies: Some experimental results, *Proceedings of the* 12*th Workshop on Descriptional Complexity of Formal Systems*, eds. I. McQuillan and G. Pighizzini, *EPTCS*, Saskatoon, Saskatchewan, Canada (August 2010), pp. 139–148.

[96] N. Moreira and R. Reis, Series-parallel automata and short regular expressions, *Fundamenta Informaticae* **91**(3–4) (2009) 611–629.

[97] C. Nicaud, On the average size of Glushov's automaton, *Proceedings of the* 3*rd International Conference Language and Automata Theory and Applications*, eds.

A. H. Dediu, A. M. Ionescu and C. Martín-Vide, *LNCS* (Springer, Tarragona, Spain, April 2009), pp. 626–637.

[98] C. Nicaud, C. Pivoteau and B. Razet, Average analysis of Glushkov automata under a BST-like model, *Proceedings of the* 30*th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, eds. K. Lodaya and M. Mahajan, *Leibniz International Proceedings in Informatics* **8** (Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Chennai, India, December 2010), pp. 388–399.

[99] G. Ott and N. H. Feinstein, Design of sequential machines from their regular expressions, *Journal of the ACM* **8** (October 1961) 585–600.

[100] A. Palioudakis, K. Salomaa and S. Akl, Comparisons between measures of nondeterminism on finite automata, *Proceedings of the* 15*th International Workshop Descriptional Complexity of Formal Systems*, eds. H. Jürgensen and R. Reis, *LNCS* (Springer, London, Ontario, Canada, July 2013), pp. 229–240.

[101] A. Palioudakis, K. Salomaa and S. G. Akl, State complexity and limited nondeterminism, *Proceedings of the* 14*th Workshop on Descriptional Complexity of Formal Systems*, eds. M. Kutrib, N. Moreira and R. Reis, *LNCS* (Springer, Braga, Portugal, July 2012), pp. 252–265.

[102] M. O. Rabin and D. Scott, Finite automata and their decision problems, *IBM Journal of Research and Development* **3** (1959) 114–125.

[103] B. Ravikumar and O. H. Ibarra, Relating the type of ambiguity of finite automata to the succinctness of their representation, *SIAM Journal on Computing* **18** (December 1989) 1263–1282.

[104] F. Reidl, P. Rossmanith, F. S. Villaamil and S. Sikdar, A faster parameterized algorithm for treedepth, arXiv:1401.7540v3 [cs.DS] (February 2014).

[105] J. Sakarovitch, *Elements of Automata Theory* (Cambridge University Press, 2009).

[106] G. Schnitger, Regular expressions and NFAs without $\epsilon$-transitions, *Proceedings of the* 23*th International Symposium on Theoretical Aspects of Computer Science*, eds. B. Durand and W. Thomas, *LNCS* (Springer, Marseille, France, February 2006), pp. 432–443.

[107] T. Schwentick, Foundations of XML based on logic and automata: A snapshot, *Proceedings of the* 7*th International Symposium on Foundations of Information and Knowledge Systems*, eds. T. Lukasiewicz and A. Sali, *LNCS* (Springer, Kiel, Germany, March 2012), pp. 23–33.

[108] S. Sippu and E. Soisalon-Soininen, *Parsing Theory, Volume I: Languages and Parsing*, EATCS Monographs on Theoretical Computer Science, Vol. 15 (Springer, 1988).

[109] K. Thompson, Regular expression search algorithm, *Communications of the ACM* **11** (June 1968) 419–422.

[110] A. W. To, Unary finite automata vs. arithmetic progressions, *Information Processing Letters* **109**(17) (2009) 1010–1014.

[111] B. W. Watson, Taxonomies and toolkits of regular language algorithms, PhD thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science (September 1995).

[112] S. Yu, State complexity of regular languages, *Journal of Automata, Languages and Combinatorics* **6** (2001) 221–234.