Tight Bounds for Cut-Operations on Deterministic Finite Automata

Frank Drewes[1], Markus Holzer[2(✉)], Sebastian Jakobi[2], and Brink van der Merwe[3]

[1] Department of Computing Science, Umeå University, Umeå, Sweden
drewes@cs.umu.se
[2] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{holzer,sebastian.jakobi}@informatik.uni-giessen.de
[3] Department of Mathematical Sciences, Computer Science Division, University of Stellenbosch, Stellenbosch, South Africa
abvdm@cs.sun.ac.za

Abstract. We investigate the state complexity of the cut and iterated cut operation for deterministic finite automata (DFAs), answering an open question stated in [M. Berglund, et al.: Cuts in regular expressions. In *Proc. DLT*, LNCS 7907, 2011]. These operations can be seen as an alternative to ordinary concatenation and Kleene star modelling leftmost *maximal* string matching. We show that the cut operation has a matching upper and lower bound of $(n-1)\cdot m+n$ states on DFAs accepting the cut of two individual languages that are accepted by n- and m-state DFAs, respectively. In the unary case we obtain $\max(2n-1, m+n-2)$ states as a tight bound. For accepting the iterated cut of a language accepted by an n-state DFA we find a matching bound of $1+(n+1)\cdot \mathsf{F}(\,1, n+2, -n+2; n+1 \mid -1\,)$ states on DFAs, where F refers to the generalized hypergeometric function. This bound is in the order of magnitude $\Theta((n-1)!)$. Finally, the bound drops to $2n-1$ for unary DFAs accepting the iterated cut of an n-state DFA and thus is similar to the bound for the cut operation on unary DFAs.

1 Introduction

The equivalence of finite automata and regular expressions is well known, and appropriate constructions for the conversion between these representations of regular languages can be found in almost all monographs on automata and formal languages. Although the concepts are the same, the implementation of regular expression matching engines may result in fundamentally different finite state devices. Besides using deterministic or nondeterministic finite automata as string matchers, the main difference is their performance characteristics and operational semantics when performing the string matching of the input word against the constructed matcher. Recently, the behaviour of nondeterministic matchers was investigated in detail with respect to exponential matching time, also referred to as catastrophic backtracking [2,6]. One possibility to control the

J. Durand-Lose and B. Nagy (Eds.): MCU 2015, LNCS 9288, pp. 45–60, 2015.
DOI: 10.1007/978-3-319-23111-2_4

work-flow of the matcher is to use operations that prevent backtracking, similarly as in the logic programming language PROLOG [3]. In fact, language operations with such a behaviour were recently introduced in [1] as an alternative to ordinary concatenation and Kleene star modelling leftmost *maximal* string matching. In order to explain the behaviour of these new regularity preserving operations consider the following pseudo-code example, which is literally taken from [1]—and assume that `match_regex` matches the *longest* prefix possible:

```
match = match_regex("(a*b)*", s);
if(match != null) then
     match = match_regex("ab*c", match.string_remainder);
     if(match != null) then
          return match.string_remainder == "";
return false;
```

For the string $s = abac$, this program first matches $R = (a^*b)^*$ to the sub-string ab, leaving ac as a remainder, which is matched by $S = ab^*c$, returning the empty string as a remainder, indicating a positive match. On the other hand, for $s = aababc$ in an execution of the program above, regular expression R matches $aabab$, leaving the remainder c, which *cannot* be matched by S, thus returning `false`, although s belongs to $R \cdot S$. Exactly this behaviour on leftmost maximal matching is modelled by the cut and iterated cut operation. In [1] basic properties of these operations with respect to formal languages and computational complexity were investigated. In particular, both operations preserve regularity. One of the many open questions stated in [1] is to develop a better or complete understanding of the upper and lower bounds on the state complexity of finite automata for both variants of cut operations. We solve this question by giving exact matching upper and lower bounds in the number of states for deterministic finite automata (DFAs) accepting the cut of two languages or the iterated cut of a single language.

In the next section we introduce the necessary notation on DFAs. Moreover, the cut and iterated cut operation is defined and the basic automata constructions for both cut operations on languages are recalled. Then in Sect. 3 the state complexity of the cut-operation on DFAs in general and on unary DFAs is investigated. Both bounds are polynomial in n and m. To be more precise, $(n-1) \cdot m + n$ states are sufficient and necessary to accept the cut of two languages accepted by n- and m-state DFAs, and $\max(2n-1, m+n-2)$ states are sufficient and necessary for *unary* DFAs. Here a DFA is unary if it has a singleton input alphabet. The tight bound for general regular languages is best possible, since the lower bound even holds for languages over a two letter alphabet. The iterated cut operation is studied in Sect. 4. Here the situation is much more involved. For DFAs in general we obtain a sufficient and necessary bound of $1+(n+1)\cdot \mathsf{F}(\,1, n+2, -n+2; n+1 \mid -1\,)$ on the exact number of states, where F refers to the generalized hypergeometric function. It is shown that this bound is in the order of $\Theta((n-1)!)$. In the unary case the bound drops to $2n-1$. Observe that the lower bound for the iterated cut operation for regular languages in general even holds for languages over a three letter alphabet. Whether a bound in the order of $\Theta((n-1)!)$ can already be obtained by a language over a two letter

alphabet is left open. Moreover, for all presented results we also discuss the effect of the number of accepting states in the involved automata to the upper and lower bounds for the cut operations. Finally, we summarize our results in the concluding section and state some open problems for future research. Owing to space constraints some proofs had to be shortened or left out. Complete proofs will be given in a forthcoming journal version of the paper.

2 Preliminaries

We recall some definitions on finite automata as contained in [5]. A *deterministic finite automaton* (DFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of *states*, Σ 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 $\delta\colon Q \times \Sigma \to Q$ is the *transition function*. The *language accepted* by the DFA A is defined as $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \in F \,\}$, where the transition function is recursively extended to $\delta\colon Q \times \Sigma^* \to Q$. A DFA is *unary*, if the input alphabet Σ is a singleton set, that is, $\Sigma = \{a\}$, for some input symbol a.

In [1] the *cut operation* on two languages L an L', denoted by $L\,!\,L'$, is defined as

$$L\,!\,L' = \{\, uv \mid u \in L,\ v \in L', \text{ and } uv' \notin L, \text{ for every } v' \in \mathrm{pref}(v) \,\},$$

where $\mathrm{pref}(v)$ denotes the set of all nonempty prefixes of the word v. Moreover, also an iterated version of the cut operation was defined. The *iterated cut* of a language L, denoted by $L^{!*}$, is the smallest language that satisfies

$$\{\lambda\} \cup (L\,!\,(L^{!*})) \subseteq L^{!*},$$

i.e., $L\,!\,(L\,!\,\ldots\,(L\,!\,(L\,!\,\{\lambda\}))\ldots) \subseteq L^{!*}$ for any number of repetitions of the cut. In other words, the language $L^{!*}$ is the least fixed point of $X \mapsto \{\lambda\} \cup (L\,!\,X)$. We also define $L^{!+}$ as the smallest language that satisfies $L \cup (L\,!\,(L^{!+})) \subseteq L^{!+}$, or equivalently, as least fixed point of $X \mapsto L \cup (L\,!\,X)$. Notice that $L^{!*} = L^{!+} \cup \{\lambda\}$. The above defined cut operations preserve regularity as shown in [1]. Since we are interested in the descriptional complexity of both operations we briefly recall both constructions from [1]—we slightly adapt these constructions such that they also work in case the initial state of the automaton has incoming transitions and is a possible final state. We start recalling the construction for the cut operation.

Let $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ be two DFAs accepting the languages L and L', respectively. Then define the automaton $C = (Q, \Sigma, \delta, q_0, F)$, with state set $Q = Q_A \cup Q_A Q_B$. The idea behind δ is to let C first run A and then, as soon as A has accepted a prefix of the input, both A and B in parallel, so that B can be reset to its initial state each time A encounters another (longer) prefix in L. Therefore, for all states $q_A, r_A \in Q_A$, $q' = q_A\, q_B \in Q$, and inputs $a \in \Sigma$ with $\delta_A(q_A, a) = r_A$ we define

$$\delta(q_A, a) = \begin{cases} r_A & \text{if } r_A \notin F_A \\ r_A\, q_{0,B} & \text{otherwise} \end{cases}$$

and

$$\delta(q', a) = \begin{cases} r_A\, \delta_B(q_B, a) & \text{if } r_A \notin F_A \\ r_A\, q_{0,B} & \text{otherwise} \end{cases}$$

and $q_0 = q_{0,A}$, if $\lambda \notin L(A)$, and $q_0 = q_{0,A}\, q_{0,B}$, otherwise. The set of final states is set to $F = Q_A F_B$. Then $L(C) = L\,!\,L'$. Since the states of C are non-empty sequences of length at most two, we refer to an element $q \in Q$ as a *stack of states* or a *stack state*. The *height* of a stack state is the length of its sequence of states. This view on the state set is used in the iterated cut construction, which is more subtle.

Again, let $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ be a DFA accepting the language L. Before we define the DFA $C = (Q, \Sigma, \delta, q_0, F)$ that accepts $L^{!+}$, we need some prerequisites in order to keep the presentation of C simple. The idea for the construction of C is as follows: first the automaton behaves like A. If it reaches one of the final states of A, say q_1, it continues in a state $q_1 q_{0,A}$, working essentially like the automaton for the language $L(A)\,!\,L(A)$. In particular, it resets the second copy each time the first copy encounters a final state of A. However, if the second copy reaches a final state q_2 of A, while $q_1 \notin F$, a third copy is initialized, thus resulting in a state of the form $q_1 q_2 q_{0,A}$, and so on. In order to keep the set of states finite we need a function $\pi : Q_A^+ \to Q_A^{\leq |Q_A|}$, which is defined as follows: for all $s = q_1 q_2 \dots q_k \in Q_A^+$: if $k = 1$, then $\pi(s) = s$, and if $k > 1$, then

$$\pi(s) = \begin{cases} \pi(q_1 q_2 \dots q_{k-1}) & \text{if } q_k \in \{q_1, q_2, \dots, q_{k-1}\} \\ \pi(q_1 q_2 \dots q_{k-1}) q_k & \text{otherwise.} \end{cases}$$

Obviously, function π removes repeated states in the state sequence from right to left. Hence, the set $\pi(Q_A^+)$ consists only of those sequences, where every state appears at most once. Now we are ready to describe C. Let $Q = \pi(Q_A^+)$ and $q_0 = q_{0,A}$. As in the previous cut construction, the elements of Q are called *stacks of states* from Q_A. Then for every $q \in Q$ with $q = q_1 q_2 \dots q_k$ and $a \in \Sigma$ let $q_i' = \delta_A(q_i, a)$, for $1 \leq i \leq k$, and set

$$\delta(q, a) = \begin{cases} \pi(q_1' q_2' \dots q_k') & \text{if } q_1', q_2', \dots, q_k' \notin F_A \\ \pi(q_1' q_2' \dots q_\ell' q_{0,A}) & \text{if } \ell = \min\{\, i \mid 1 \leq i \leq k \text{ and } q_i' \in F_A.\,\} \end{cases}$$

The set of final states is

$$F = \{\, q \in Q \mid q = q_1 q_2 \dots q_k \text{ with } q_k \in F_A \text{ or } k > 1 \text{ and } q_{k-1} \in F_A \,\}.$$

Observe, that a *reachable* final state $q = q_1 q_2 \dots q_k$ with $q_{k-1} \in F_A$ must fulfill $q_k = q_{0,A}$ by construction. Moreover, if $q = q_1 q_2 \dots q_k$ is a *reachable* final state with $q_k \in F_A \setminus \{q_{0,A}\}$, then we must have $q_{0,A} \in \{q_1, q_2, \dots, q_{k-1}\}$. The language accepted by C is $L(C) = L^{!+}$. Because $L^{!*} = L^{!+} \cup \{\lambda\}$, a DFA for $L^{!*}$ can be obtained from C by simply introducing an additional *accepting* copy of the initial state of C (unless $q_{0,A} \in F_A$ and thus $L(C) = L^{!+} = L^{!*}$).

In the forthcoming sections we consider the descriptional complexity of both operations, when the regular languages are given by DFAs. The above presented constructions show an asymptotic upper bound of $O(n \cdot (m+1))$ for the cut operation, and an asymptotic upper bound of $O(n!)$ for the iterated cut operation, if A and B are DFAs with n and m states, respectively.

3 The Descriptional Complexity of the Cut Operation

In this section we prove a tight bound for a DFA accepting the cut of two languages, when these languages are represented by an n- and m-state DFA, respectively. This exact tight bound is $(n-1) \cdot m + n$, which is witnessed by automata using binary input alphabets. Then we consider the special case of unary languages. Here we have to do a detailed analysis of the structure of unary DFAs in order to prove a tight bound of $\max(2n-1, n+m-2)$ on the number of states for unary DFAs. Notice that for $m \leq n$ this bound only depends on the first automaton, but not on the second.

First we consider a few special cases. Let Σ be the input alphabet of the DFA A. If all states in A are accepting, then $L(A) = \Sigma^*$, and if A does not have an accepting state at all, then $L(A) = \emptyset$. Thus in both cases, the cut of $L = L(A)$ with any other language L', i.e., $L \,!\, L'$, is empty or equal to Σ^* (the latter being the case if $L = \Sigma^*$ and $\lambda \in L'$). Thus, in each of these cases the resulting language can be described by a DFA with single state only. In general we obtain the following result.

Theorem 1. *Let A be an n-state and B an m-state deterministic finite automaton. Then $(n-1) \cdot m + n$ states are sufficient and necessary in the worst case for any deterministic finite automaton accepting the language $L(A) \,!\, L(B)$. The lower bound even holds for automata with binary input alphabet.*

Proof. Let $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ be the n- and m-state DFAs, respectively. Applying the previously described construction for the cut gives a DFA C that accepts $L(A) \,!\, L(B)$ with a state set consisting of stack states of height at most two. A careful inspection of this construction reveals that some stack states are not reachable, since they do not have any incoming transitions. (i) Among the stack states of height one only those that are *not* final states in A are possibly reachable. (ii) For the stack states of height two we consider two sub-cases, namely if the first element of the sequence is in F_A, then the second one is always the initial state of B, and if the first element of the sequence is in $Q_A \setminus F_A$, the second one may be an arbitrary element of Q_B. Thus, the state set Q_C of C can be restricted to contain stack states from

$$Q_C = \{\, p \mid p \in Q_A \setminus F_A \,\} \cup F_A\{q_{0,B}\} \cup \{\, pq \mid p \in (Q_A \setminus F_A) \text{ and } q \in Q_B \,\}$$

without changing the accepted language. Thus, we have at most

$$n - |F_A| + |F_A| + (n - |F_A|) \cdot m = (n - |F_A|) \cdot m + n$$

states, which is maximal if A has only a single accepting state, leading to an upper bound of $(n-1) \cdot m + n$.

Next we show that this upper bound can be reached. To this end we define the n-state DFA $A = (Q_A, \{a,b\}, \delta_A, q_{0,A}, F_A)$, where $Q_A = \{0, 1, \ldots, n-1\}$, $q_{0,A} = 0$, $F_A = \{n-1\}$, and

$$\delta_A(i,a) = i+1 \pmod{n} \quad \text{and} \quad \delta_A(i,b) = i, \quad \text{for } 0 \leq i \leq n-1.$$

Moreover, we define the m-state DFA $B = (Q_B, \{a,b\}, \delta_B, q_{0,B}, F_B)$, where $Q_B = \{0, 1, \ldots, m-1\}$, $q_{0,B} = 0$, $F_B = \{0\}$, and

$$\delta_B(i,a) = i, \quad \text{for } 0 \leq i \leq m-1 \quad \text{and} \quad \delta_B(i,b) = i+1 \pmod{m}$$

Both finite automata are depicted in Fig. 1. Again, let C be the DFA constructed from A and B by applying the construction for the cut, where the state set Q_C is restricted according to our previous considerations. Moreover, let $q_{0,C}$ be the initial state of C. To prove that C needs exactly the claimed number of states, it can be shown that all states in Q_C are reachable and pairwise distinguishable. Due to space constraints the proof is omitted. □

As we have seen in the previous proof, the upper bound on the number of states for the cut of two DFAs implicitly depends on the number of accepting states of the first automaton. It is easy to generalize the above argument to lead to an even more precise tight upper and lower bound of $(n-f) \cdot m + n$ states, where f with $1 \leq f < n$ refers to the number of accepting states of the "left" automaton A. For the lower bound proof one simply has to alter the definition of the automaton A by setting its accepting states to $F_A = \{n-f, \ldots, n-2, n-1\}$. The details are left to the reader.

In the remainder of this section we briefly mention (without proofs) the descriptional complexity of the cut operation on unary languages, that is, languages over a single letter alphabet. Deterministic finite automata for unary

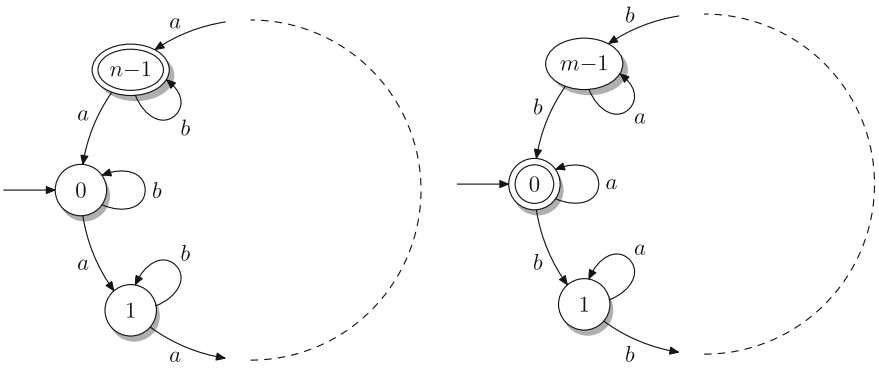

Fig. 1. The binary DFAs A (left) and B (right) with n and m states, respectively, that witness the state complexity lower bound for the cut operation.

languages obey a very simple structure: a (possibly empty) initial chain, followed by a cycle.

Theorem 2. *Let A be an n-state and B an m-state deterministic finite automaton accepting a unary language. Then $g(n,m)$ states, where*

$$g(n,m) = \begin{cases} 1 & \text{if } n = 1, \\ n & \text{if } n \geq 2 \text{ and } m = 1, \\ 2n-1 & \text{if } n,m \geq 2 \text{ and } m \leq n, \\ n+m-2 & \text{if } n,m \geq 2 \text{ and } m > n, \end{cases}$$

are sufficient and necessary in the worst case for any deterministic finite automaton accepting the language $L(A)\,!\,L(B)$. □

Similarly to the non-unary case, the state complexity of the cut on unary DFAs also depends on the number of accepting states of the "left" automaton A. If $L(A)$ is infinite, and automaton A has f accepting states, then we obtain a tight bound of $2n-f$ states for a DFA that accepts the language $L(A)\,!\,L(B)$. However, if $L(A)$ is finite then the bound stays $n+m-2$, regardless of the number of accepting states.

Sometimes, when studying descriptional complexity of unary regular languages, one does not simply count the number of states of a DFA, but rather distinguishes between the length of the initial chain and the length of the cycle. So instead of asking for the number $g(n,m)$ of states of a DFA for accepting the cut of the languages described by n-state and m-state DFAs, one could also study the following: given unary DFAs A_1 and A_2, with t_1 (t_2, respectively) states in the initial chain and k_1 (k_2, respectively) states in the cycle, determine bounds (as functions in t_1, t_2, k_1, k_2) for the number t of states in the initial chain and the number k of states in the cycle of a DFA for the language $L(A_1)\,!\,L(A_2)$. Since the results on tight bounds for t and k branch out into many different sub-cases, we will not go into details here, but rather summarize our results in Table 1.

4 The Descriptional Complexity of the Iterated-Cut Operation

In this section we turn our attention to the state complexity of the iterated cut operation on DFAs. In order to properly state our result we need some more notation. A *generalized hypergeometric function* [4] is a power series in x with $r+s$ parameters, and it is defined as follows in terms of rising factorial powers:

$$\mathsf{F}\left(\begin{matrix} a_1, a_2, \ldots, a_r \\ b_1, b_2, \ldots, b_s \end{matrix} \,\middle|\, x \right) = \sum_{\ell \geq 0} \frac{a_1^{\overline{\ell}} a_2^{\overline{\ell}} \ldots a_r^{\overline{\ell}}}{b_1^{\overline{\ell}} b_2^{\overline{\ell}} \ldots b_s^{\overline{\ell}}} \cdot \frac{x^\ell}{\ell!}.$$

Here the *rising factorial* is defined as $x^{\overline{\ell}} = x(x+1)\cdots(x+(\ell-1))$ and the *falling factorial* by $x^{\underline{\ell}} = x(x-1)\cdots(x-(\ell-1))$. By convention $x^{\overline{0}} = x^{\underline{0}} = 1$.

Table 1. Tight bounds for the length t of the initial chain and the length k of the cycle of a DFA for the language $L(A_1)\,!\,L(A_2)$, where A_i has an initial chain of length t_i and a cycle of length k_i, for $i = 1, 2$. The table is ordered by ascending values of k_1. If a tuple (t_1, k_2, t_2, k_2) matches multiple lines, the additional condition column has to be checked.

t_1	k_1	t_2	k_2	condition	t	k
0	1	≥ 0	≥ 1		0	1
≥ 1	1	≥ 0	≥ 1	accepting state in cycle 1	t_1	1
≥ 1	1	≥ 0	≥ 1	no accepting state in cycle 1	$t_1 + t_2 - 1$	k_2
≥ 0	≥ 2	0	1		$t_1 + k_1 - 1$	1
≥ 0	≥ 2	≥ 1	1		$t_1 + k_1 - 1$	k_1
≥ 0	2	0	≥ 2		$t_1 + k_1 - 1$	k_1
≥ 0	2	1	2		$t_1 + k_1 - 1$	1
≥ 0	2	1	≥ 3		$t_1 + k_1 - 1$	k_1
≥ 0	2	≥ 2	≥ 2		$t_1 + k_1 - 1$	k_1
≥ 0	≥ 3	≥ 0	≥ 2	$k_1 \leq t_2 + k_2$	$t_1 + k_1 - 1$	k_1
≥ 0	≥ 3	0	≥ 2	$k_1 > k_2$ and $k_2 \bmod k_1 > 0$	$t_1 + k_1 - 1$	k_1
≥ 0	≥ 3	0	≥ 2	$k_1 > k_2$, and $k_1 = k \cdot k_2$, and 1 accepting in k_1-loop	$t_1 + k_1 - 1$	k_2
≥ 0	≥ 3	0	≥ 2	$k_1 > k_2$, and $k_1 = k \cdot k_2$, and ≥ 2 accepting in k_1-loop	$t_1 + k_1 - 2$	k_1
≥ 0	≥ 3	≥ 1	≥ 2	$k_1 > t_2 + k_2$	$t_1 + k_1 - 1$	k_1

Then our result on the number of states that are sufficient and necessary to accept the iterated cut $L^{!+}$ of a single language accepted by an n-state DFA reads as follows—a corresponding result for $L^{!*}$ will be given later.

Theorem 3. *Let A be a deterministic finite automaton with n states. Then*

$$(n+1) \cdot \mathsf{F}\left(\begin{matrix} 1, n+2, -n+2 \\ n+1 \end{matrix} \,\middle|\, -1 \right)$$

states are sufficient and necessary in the worst case for a deterministic finite automaton to accept the language $L(A)^{!+}$. The lower bound even holds for automata with ternary input alphabet.

Before we prove this theorem by the upcoming two lemmata, we first show that the following combinatorial identity holds.

Theorem 4. *For natural numbers n with $n \geq 2$ we have the identity*

$$(n+1) \cdot \mathsf{F}\left(\begin{matrix} 1, n+2, -n+2 \\ n+1 \end{matrix} \,\middle|\, -1 \right) = \sum_{\ell=0}^{n-2} (n+\ell+1) \cdot (n-2)^{\underline{\ell}}. \qquad (1)$$

Proof. The proof outline follows the presentation on hypergeometric functions given in [4]. Note that the sum on the right hand-side can be changed to sum up for all ℓ with $\ell \geq 0$, because for $\ell > (n-2)$ the falling factorials $(n-2)^{\underline{\ell}}$ are always zero, and thus these terms do not contribute anything to the sum.

Now let the notation of the series be $\sum_{\ell \geq 0} t_\ell$ with $t_0 \neq 0$. If the term ratio $t_{\ell+1}/t_\ell$ is a rational function in ℓ, that is, a quotient of polynomials in ℓ of the form

$$\frac{(\ell + a_1)(\ell + a_2)\dots(\ell + a_r)}{(\ell + b_1)(\ell + b_2)\dots(\ell + b_s)} \cdot \frac{x}{(\ell + 1)}$$

then we can use the ansatz

$$\sum_{\ell \geq 0} t_\ell = t_0 \cdot \mathsf{F}\left(\begin{matrix} a_1, a_2, \dots, a_r \\ b_1, b_2, \dots, b_s \end{matrix} \,\middle|\, x \right).$$

As $t_\ell = (n + \ell + 1)(n-2)^{\underline{\ell}}$, the first term of our sum is $t_0 = (n+1)$, and the other terms have the ratios given by

$$\frac{t_{\ell+1}}{t_\ell} = \frac{(n + \ell + 2)(n-2)^{\underline{\ell+1}}}{(n + \ell + 1)(n-2)^{\underline{\ell}}} = \frac{(n + \ell + 2)(n - 2 - \ell)}{(n + \ell + 1)},$$

which are rational functions of ℓ. Rearranging the terms and introducing the required factor $(\ell + 1)$ in the denominator results in

$$\frac{t_{\ell+1}}{t_\ell} = \frac{(\ell + 1)(\ell + n + 2)(\ell - n + 2)}{(\ell + n + 1)} \cdot \frac{(-1)}{(\ell + 1)},$$

where we can read off the result: the given sum is equal to

$$(n+1) \cdot \mathsf{F}\left(\begin{matrix} 1, n+2, -n+2 \\ n+1 \end{matrix} \,\middle|\, -1 \right),$$

which proves the stated result. □

The first few values of the hypergeometric function in Equation (1) starting with $n = 1$ are $1, 3, 9, 31, 129, 651, 3913, 27399, 219201, 1972819, 19728201, \dots$. In the On-Line Encyclopedia of Integer Sequences (OEIS)—see www.oeis.org—this matches the sequence A111063. A detailed analysis of the behaviour of this sequence is given after the following two lemmata that prove Theorem 3.

Lemma 5. *Let A be deterministic finite automaton with n states. Then*

$$\sum_{\ell=0}^{n-2} (n + \ell + 1) \cdot (n-2)^{\underline{\ell}}$$

states are sufficient for a deterministic finite automaton to accept $L(A)^{!+}$.

Proof. The upper bound can be seen as follows. Let $A = (Q, \Sigma, \delta, q_{0,A}, F)$ be a DFA, and C be the DFA as constructed in Sect. 2 for accepting $L(A)^{!+}$. By construction the state set of C is $\pi(Q^+)$, consisting of stacks of states from Q. Every such stack of height $\ell \geq 1$ is of one of the following forms, called *types*—recall that by the definition of π, all elements in a stack state are pairwise distinct:

Type 1: $q = q_1 q_2 \dots q_\ell$, with $q_1, q_2 \dots, q_\ell \in Q \setminus F$, or
Type 2: $q = q_1 q_2 \dots q_{\ell-1} q_{0,A}$, with $q_1, q_2, \dots, q_{\ell-1} \in Q \setminus F$, and $q_{0,A} \in F$, or
Type 3: $q = q_1 q_2 \dots q_{\ell-1} q_\ell$, with $q_1, q_2, \dots, q_{\ell-1} \in Q \setminus F$, and $q_\ell \in F$, and $q_{0,A} \in \{q_1, q_2, \dots q_{\ell-1}\}$, or
Type 4: $q = q_1 q_2 \dots q_{\ell-2} q_{\ell-1} q_{0,A}$ (and therefore $q_{0,A} \notin \{q_1, q_2, \dots q_{\ell-1}\}$), with $q_1, q_2, \dots, q_{\ell-2} \in Q \setminus F$ and $q_{\ell-1} \in F$.

Let us count the number of states of the different types. Clearly, the number of different stacks of type 1 is $\sum_{\ell=1}^{n-|F|}(n-|F|)^{\underline{\ell}}$, and the number of type 2 stacks is $\sum_{\ell=1}^{n-|F|+1}(n-|F|)^{\underline{\ell-1}}$. To build a stack of type 3 we choose $\ell-2$ non-accepting, non-initial states, permute them, then shuffle $q_{0,A}$ somewhere into these states, and put an accepting state on top. This gives $\sum_{\ell=2}^{n-|F|+1}(n-|F|-1)^{\underline{\ell-2}}\cdot(\ell-1)\cdot|F|$ different stacks. Finally, to count the number of stacks of type 4, we distinguish between the two cases $q_{0,A} \in F$ and $q_{0,A} \notin F$. In the former case, a stack is built by choosing and permuting $\ell-2$ non-accepting states, then putting an accepting, non-initial state and state $q_{0,A}$ on top—this gives $\sum_{\ell=2}^{n-|F|+2}(n-|F|)^{\underline{\ell-2}}\cdot(|F|-1)$ different stacks of type 4. Similarly, for the case where $q_{0,A} \notin F$ we choose and permute $\ell-2$ non-accepting, non-initial states, put an accepting state and then state $q_{0,A}$ on top, which gives $\sum_{\ell=2}^{n-|F|+1}(n-|F|-1)^{\underline{\ell-2}}\cdot|F|$ stacks.

The bound in the statement of the lemma will result from the case where $|F| = 1$ and $q_{0,A} \notin F$. To see that this case indeed yields an upper bound for all the cases, we first argue that the overall number of different possible stacks increases when the number of accepting states decreases.

Given a deterministic finite automaton $A = (Q, \Sigma, \delta, q_{0,A}, F)$ with $|F| \geq 2$, we construct an automaton $B = (Q, \Sigma, \delta, q_{0,A}, F')$ such that

$$F' = \begin{cases} F \setminus \{q_{0,A}\} & \text{if } q_{0,A} \in F, \\ F \setminus \{q_f\} & \text{for some } q_f \in F \text{ if } q_{0,A} \notin F. \end{cases}$$

Denote by $\mathcal{S}(Q, F)$ the set of stacks that can be built by applying the automaton construction for $L^{!+}$ to automaton A, and by $\mathcal{S}(Q, F')$ those that can be built by applying the construction to B. We want to show $|\mathcal{S}(Q, F)| \leq |\mathcal{S}(Q, F')|$. In fact, treating the stacks as words over alphabet Q, we show the inclusion $\mathcal{S}(Q, F) \subseteq \mathcal{S}(Q, F')$. Clearly, every type 1 stack in $\mathcal{S}(Q, F)$ also appears as type 1 stack in $\mathcal{S}(Q, F')$, since $Q \setminus F$ is a subset of $Q \setminus F'$. Now assume we have a type 2 stack $q_1 q_2 \dots q_{\ell-1} q_{0,A} \in \mathcal{S}(Q, F)$, which means that $q_{0,A} \in F$. Then this stack $q_1 q_2 \dots q_{\ell-1} q_{0,A}$ is a type 1 stack in $\mathcal{S}(Q, F')$. For stacks $q_1 q_2 \dots q_{\ell-1} q_\ell \in \mathcal{S}(Q, F)$ of type 3 we have $q_\ell \in F$ and $q_{0,A} \notin F$. Here we distinguish between the two cases $q_\ell \in F'$ and $q_\ell \notin F'$. In the former case, the stack $q_1 q_2 \dots q_{\ell-1} q_\ell$ also appears as type 3 stack in $\mathcal{S}(Q, F')$, and in case $q_\ell \notin F'$ this stack is of type 1 in $\mathcal{S}(Q, F')$. The argumentation for stacks $q_1 q_2 \dots q_{\ell-2} q_{\ell-1} q_{0,A} \in \mathcal{S}(Q, F)$ of type 4, where $q_{\ell-1} \in F$ is similar: if $q_{\ell-1} \in F'$, then $q_1 q_2 \dots q_{\ell-2} q_{\ell-1} q_{0,A}$ is also a type 4 stack in $\mathcal{S}(Q, F')$, and if $q_{\ell-1} \notin F'$, then it appears in $\mathcal{S}(Q, F')$ as a stack of type 1. We have shown $\mathcal{S}(Q, F) \subseteq \mathcal{S}(Q, F')$, so the smaller the set of accepting states, the larger is the number of possible stacks. Therefore, the case

where $|F| = 1$ forms an upper bound—we ignore $|F| = 0$ since the accepted language would be empty. From [1] we already know that languages accepted by DFAs where the initial state is the sole accepting state are closed under iterated cut. Thus, for the upper bound we choose the case $|F| = 1$ and $q_{0,A} \notin F$. Using the sums from above we obtain

$$\begin{aligned}
&\sum_{\ell=1}^{n-1}(n-1)^{\underline{\ell}} + \sum_{\ell=2}^{n}(n-2)^{\underline{\ell-2}} \cdot (\ell-1) + \sum_{\ell=2}^{n}(n-2)^{\underline{\ell-2}} \\
&= \sum_{\ell=1}^{n-1}(n-1) \cdot (n-2)^{\underline{\ell-1}} + (n-2)^{\underline{\ell-1}} \cdot (\ell+1) \\
&= \sum_{\ell=0}^{n-2}(n+\ell+1) \cdot (n-2)^{\underline{\ell}}
\end{aligned}$$

as an upper bound for the number of states of a DFA for the language $L(A)^{!+}$. Notice that we ignore type 2 stacks and choose the sum for the second case of type 4 stacks, since we have $q_{0,A} \notin F$. □

The next lemma provides a matching lower bound, and thus concludes the proof of Theorem 3.

Lemma 6. *For every $n \geq 4$, there exists a deterministic finite automaton A with n states, such that the number of states of the minimal deterministic finite automaton for the language $L(A)^{!+}$ is*

$$\sum_{\ell=0}^{n-2}(n+\ell+1) \cdot (n-2)^{\underline{\ell}}.$$

Proof. Let $n \geq 4$ and $A = (Q, \Sigma, \delta, 0, F)$ with input alphabet $\Sigma = \{a, b, c\}$, state set $Q = \{0, 1, \ldots, n-1\}$ and accepting states $F = \{n-1\}$ be the DFA depicted in Fig. 2. The transition function δ is defined as follows:

$$\delta(q,a) = \begin{cases} q+1 & \text{if } 0 \leq q \leq n-3, \\ 0 & \text{if } q = n-2, \\ n-1 & \text{if } q = n-1, \end{cases}$$

$$\delta(q,b) = \begin{cases} 1 & \text{if } q = 0, \\ 0 & \text{if } q \in \{1, n-1\}, \\ q & \text{if } 2 \leq q \leq n-2, \end{cases}$$

and

$$\delta(q,c) = \begin{cases} n-1 & \text{if } q = 0, \\ q & \text{if } 1 \leq q \leq n-2, \\ 0 & \text{if } q = n-1. \end{cases}$$

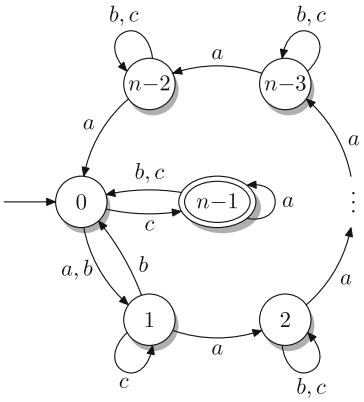

Fig. 2. The n-state DFA A for witnessing the state complexity lower bound for the iterated cut operation.

First notice, that the two mappings $q \mapsto \delta(q, a)$ and $q \mapsto \delta(q, b)$ generate all permutations on the set $Q \setminus F$—see, e.g., [7]. Let $C = (Q_C, \Sigma, \delta_C, q_{0,C}, F_C)$ be the DFA constructed from A by applying the construction for $L^{!+}$. To prove that C needs exactly the claimed number of states, we show that all states, or stacks, of types 1, 3, and 4 from the proof of Lemma 5 are reachable and pairwise distinguishable—note that stacks of type 2 do not appear in C.

Reachability of stacks of type 1 is easy to see: using permutations on the set $Q \setminus F$, which can be realized by reading appropriate words over $\{a, b\}$, every type 1 stack can be transformed into every other type 1 stack of the same size. Moreover, from a type 1 stack of the form $q_1 q_2 \dots q_{\ell-1} 0$ of size $\ell \leq n-2$, with $1 \notin \{q_1, q_2, \dots, q_{\ell-1}\}$, we can reach a stack $q_1 q_2 \dots q_{\ell-1}\, 1\, 0$ of type 1 with size $\ell+1$ by reading cb. Now every type 4 stack $q_1 q_2 \dots q_{\ell-2}\, (n-1)\, 0$ can be reached from the type 1 stack $q_1 q_2 \dots q_{\ell-2}\, 0$ by reading c. From type 4 stacks we can reach type 3 stacks as follows. If the wanted stack has the initial state 0 directly below state $(n-1)$, that is, if it has the form $q_1 q_2 \dots q_{\ell-2}\, 0\, (n-1)$, then we can reach it from $q_1 q_2 \dots q_{\ell-2}\, (n-1)\, 0$ by simply reading c. If the element 0 is not directly below element $(n-1)$, that is, if we want to obtain a stack of the form

$$q_1 q_2 \dots q_{i-1}\, 0\, q_i q_{i+1} \dots q_{\ell-2}\, (n-1),$$

with $i \leq \ell - 2$, then we can reach it from the type 3 stack

$$p_1 p_2 \dots p_{i-1}\, r\, p_i p_{i+1} \dots p_{\ell-2}\, (n-1),$$

by reading $a^{q_{\ell-2}}$, where $p_j = (q_j - q_{\ell-2}) \mod (n-2)$ for $1 \leq j \leq \ell - 2$, and $r = (-q_{\ell-2}) \mod (n-2)$. Notice that $p_{\ell-2} = 0$, so we already know how to reach the latter stack from a type 4 stack.

It remains to prove that every pair of states, or stacks, s_1 and s_2 of C can be distinguished by some input word. Clearly we only have to consider the cases where either both stacks contain the accepting state $n-1$, or none of them does.

We start with the case where $n-1$ does not appear in the stacks. If there is some element q_i that appears in one of the two stacks but not in the other, then we can use a permutation that interchanges q_i and 0, and leaves the other elements stable—if $q_i = 0$, we just take the identity permutation. Now one stack contains element 0 and the other does not, so we can distinguish between those two by reading c. Now assume that both stacks contain the same elements, but differ in their ordering. If the two stacks already differ in their bottom elements, that is, if $s_1 = q_1 t_1$ and $s_2 = q_2 t_2$ with $q_1 \neq q_2$ and appropriate sequences t_1 and t_2, then we use a permutation to interchange q_1 and 0, and obtain stacks $0\,t_1'$ and $q_2' t_2'$, with $q_2' \neq 0$. With a permutation for interchanging q_2' and 2, we obtain stacks $0\,t_1''$ and $2\,t_2''$. These are distinguished by reading cb: the second stack yields a stack containing the element 2, but the stack $0\,t_1''$ yields $0\,1$, which does not contain this element, and we have seen above how to distinguish the stacks in this case.

Next we show that we can also distinguish between stacks s_1 and s_2 that both contain the accepting state $n-1$. Let us consider the lowest (leftmost) position in which the two stacks differ. We have $s_1 = s_0\,q_1\,t_1$ and $s_2 = s_0\,q_2\,t_2$, for appropriate sequences s_0, t_1, t_2 and elements $q_1 \neq q_2$. If at least one of the elements q_1 and q_2 is from the set $\{2, 3, \ldots, n-2\}$, then the stacks obtained from s_1 and s_2 after reading b are still different because every state $q \in \{2, 3, \ldots, n-2\}$ loops on input b and no other b-transitions lead to q. Now there are only three cases remaining, namely $\{q_1, q_2\} = \{0, 1\}$, $\{q_1, q_2\} = \{0, n-1\}$, and $\{q_1, q_2\} = \{1, n-1\}$. In the first two cases, where one of the elements q_1 and q_2 is 0, we can also read input b to get rid of element $n-1$, and again we obtain two different stacks because the transition from 0 to 1 is the only b-transition that leads to state 1. For the remaining case, we may assume $q_1 = 1$ and $q_2 = n-1$, so we have stacks

$$s_1 = s_0\,1\,t_1 \qquad \text{and} \qquad s_2 = s_0\,(n-1)\,t_2,$$

for appropriate sequences s_0, t_1, and t_2, where in particular element 1 does not appear in s_0. Now we read the input word ab. First, after reading a we have stacks

$$s_1' = s_0'\,2\,t_1' \qquad \text{and} \qquad s_2' = s_0'\,(n-1)\,0,$$

where element 2 does not appear in s_0', and thus, not in s_2'. Now by reading the b symbol, stack s_1' yields a stack of the form $s_1'' = s_0''\,2\,t_1''$ that contains element 2, while stack s_2' results in stack $s_2'' = s_0''\,0$ or $s_2'' = s_0''\,0\,1$, depending on whether element 1 appears in s_0''. In either case the stacks s_1'' and s_2'' are different and none of them contains element $n-1$, so they can be distinguished as described earlier. This concludes our proof. □

Recall that the DFA for $L^{!+}$ can be turned into a DFA for $L^{!*}$ by adding a new accepting initial state. Therefore, the upper bound for the state complexity for $L^{!*}$ is by one larger than the bound for $L^{!+}$. In fact, one can see as follows that this bound is also tight by using the same witness automaton as in the

previous proof. Let $q'_{0,C}$ be the new accepting initial state; it has the same outgoing transitions as state $q_{0,C} = 0$. We only need to distinguish state $q'_{0,C}$ from all other accepting states of C, which are the stack states that contain element $(n-1)$. This can simply be done by reading letters a: the successor of the accepting stack state also contains element $(n-1)$, and thus is accepting, but the successor of state 0, and thus also of $q'_{0,C}$, is the non-accepting state 1. Therefore we obtain the following result on the state complexity of the iterated cut $L^{!*}$.

Theorem 7. *Let A be a deterministic finite automaton with n states. Then*

$$1 + (n+1) \cdot \mathsf{F}\left(\begin{matrix} 1, n+2, -n+2 \\ n+1 \end{matrix} \,\middle|\, -1 \right)$$

states are sufficient and necessary in the worst case for a deterministic finite automaton to accept the language $L(A)^{!*}$. The lower bound even holds for automata with ternary input alphabet. □

As in the case of the (non-iterated) cut operation, one can also derive a more precise bound for the state complexity of the iterated cut operation, which depends on the number f of accepting states of the automaton. From the upper bound analysis in the proof of Lemma 5, a bound of

$$\sum_{\ell=0}^{n-f-1} (n-f-1)^{\underline{\ell}} \cdot (n + f \cdot (\ell+1))$$

states can be derived for the case where the initial state $q_{0,A}$ is not accepting. In fact, by some calculations and combinatorial argumentation, one can show that the case, where the initial state *is* accepting, yields the same upper bound. By adapting the automaton from the lower bound proof of Lemma 6, one can obtain witness automata for the more precise bound as follows. Instead of a single accepting state $n-1$ we choose f accepting states $n-1, n-2, \ldots, n-f$, that are connected to a cycle of $n-f$ non-accepting states in a similar fashion as shown in Fig. 2: for each accepting state $n-i$, for $1 \leq i \leq f$, we use another input symbol c_i that switches between states $n-i$ and 0; on input symbol b the states $n-i$ also go to state 0.

Now let us come to the asymptotics of the bounds stated in Theorems 3 and 7 in order to get a better feeling for their size. This can be done by first proving the following upper and lower bounds.

Theorem 8. *The following lower and upper bounds apply:*

$$2 \cdot (n+2) \cdot (n-2)! \leq \sum_{k=0}^{n-2} (n+k+1)(n-2)^{\underline{k}} \leq e \cdot (2n-1) \cdot (n-2)! \qquad \square$$

The upper and the lower bound are quite close, since the former is asymptotically only a factor of e away from the latter. This allows us to show that the bounds provided in Theorems 3 and 7 in the exact number of states asymptotically behave like $(n-1)!$.

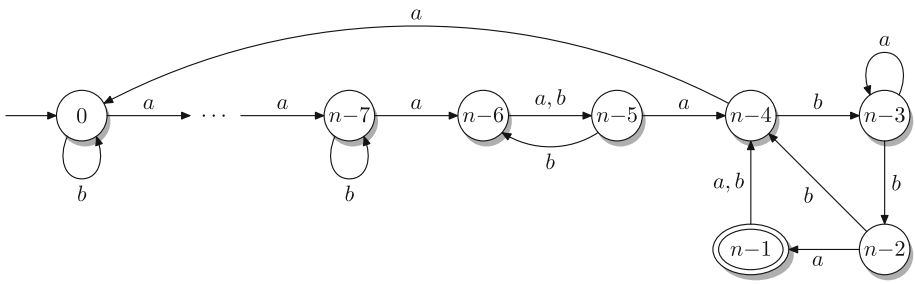

Fig. 3. A DFA A with binary input alphabet $\Sigma = \{a, b\}$, where the minimal DFA for $L(A)^{!+}$ needs at least $\sum_{\ell=1}^{n-3}(n-3)^{\underline{\ell}} > (n-3)!$ states.

Theorem 9. *Let A be a deterministic finite automaton with n states. Then $\Theta((n-1)!)$ states are sufficient and necessary in the worst case for a deterministic finite automaton to accept the language $L(A)^{!+}$. The same holds for a deterministic finite automaton for $L(A)^{!*}$.* □

Notice that the witness automaton from the proof of Lemma 6 uses a ternary input alphabet. Theorem 10 will give a tight bound of $2n-1$ for the state complexity of the iterated cut operation on DFAs with unary input alphabet. So it remains to study the exact bound for automata using a binary alphabet. Here we do not have a tight bound yet. However, we know that already a binary alphabet is enough to provide a huge blow-up in the number of states. Consider the binary n-state DFA A depicted in Fig. 3. One can show that the minimal DFA for the language $L(A)^{!+}$ needs more than $\sum_{\ell=1}^{n-3}(n-3)^{\underline{\ell}} > (n-3)!$ states. The basic idea to prove this is that all stack states with elements from the set $\{0, 1, \ldots, n-4\}$ can be reached: permutations on this set can be obtained with the help of the input words a (cycling through the set) and b^3 (switching elements $n-5$ and $n-6$). New elements can be obtained by reading b^2ab^4 from a stack of the form $s = s'\,(n-4)$, where $n-5$ does not appear in s'.

Now we come to our result on the state complexity of the iterated cut operation on unary languages. The upcoming theorem only considers input automata with at least three states. The reader may convince him- or herself that every minimal two-state DFA A (there are only four possibilities) yields as iterated cut $L(A)^{!*}$ a language that can also be accepted by a DFA with at most two states. Moreover, the iterated cut languages of the single-state languages $\emptyset$ and $\{a\}^*$ are $\{\lambda\}$ and $\{a^*\}$ respectively, and thus are accepted by a two-state, respectively, single-state DFA.

We now state the already mentioned theorem providing the tight bound for the iterated cut in the case of a unary alphabet.

Theorem 10. *Let A be a deterministic finite automaton with $n \geq 3$ states that accepts a unary language. Then $2n-1$ states are sufficient and necessary in the worst case for a deterministic finite automaton to accept the language $L(A)^{!+}$. The same holds for a deterministic finite automaton for $L(A)^{!*}$.* □

Again one can prove a more precise bound of $2n - f$ states, where f is the number of accepting states of A.

5 Conclusions

We have investigated the state complexity of the cut and iterated cut operation for DFAs. In all cases, we obtained tight upper and lower bounds in the exact number of states. Thus, we have solved an open problem stated in [1]. Nevertheless, many open questions and details remain to be worked out:

- Consider the upper and lower bounds for nondeterministic finite automata (NFAs) on cut operations. Can we do better than first determinizing the involved devices and then performing the cut or iterated cut construction for DFAs? Note that for cuts one does not need to determinize the second automaton B in order to construct the (then nondeterministic) automaton for $L(A)\,!\,L(B)$ as in Sect. 2.
- The complexity of decision problems related to the cut and iterated cut operation on finite automata, in particular, on DFAs. An example of such a problem is the following: given a finite automaton A, is $L(A)^{!\,*} = L(A)^*$?
- Succinctness of cut expressions (these are regular expressions that also use the cut operation) compared to DFAs and NFAs were discussed in [1]. There exponential lower bounds for both types of finite state devices were obtained. So what about the succinctness of iterated cut expressions (regular expressions that also use iterated cut) compared to finite automata?

Acknowledgments. Thanks to Rogério Reis for his help doing and verifying some calculations with the computer algebra system MAPLE™.

References

1. Berglund, M., Björklund, H., Drewes, F., van der Merwe, B., Watson, B.: Cuts in regular expressions. In: Béal, M.-P., Carton, O. (eds.) DLT 2013. LNCS, vol. 7907, pp. 70–81. Springer, Heidelberg (2013)
2. Berglund, M., Drewes, F., van der Merwe, B.: Analyzing catastrophic backtracking behavior in practical regular expressions matching. In: Ésik, Z., Fülöp, Z. (eds.) Proceedings of the 14th International Conference on Automata and Formal Languages. EPTCS, vol. 151, pp. 109–123, Szeged, Hungary (2014)
3. Clocksin, W.F., Mellish, C.S.: Programming in Prolog. Springer, Heidelberg (1981)
4. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics: A foundation for Computer Science. Addison-Wesley, Boston (1994)
5. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Boston (1978)
6. Kirrage, J., Rathnayake, A., Thielecke, H.: Static analysis for regular expression denial-of-service attacks. In: Lopez, J., Huang, X., Sandhu, R. (eds.) NSS 2013. LNCS, vol. 7873, pp. 135–148. Springer, Heidelberg (2013)
7. Piccard, S.: Sur les bases du groupe symétrique et les couples de substitutions qui engendrent un groupe régulier. Librairie Vuibert, Paris (1946)