

REGULAR EXPRESSIONS: NEW RESULTS AND OPEN PROBLEMS^{1,2}

KEITH ELLUL, BRYAN KRAWETZ, JEFFREY SHALLIT, and MING-WEI WANG

Department of Computer Science, University of Waterloo

Waterloo, Ontario, Canada N2L 3G1

*e-mail: {kbellul,bakrawet,m2wang}@math.uwaterloo.ca
shallit@uwaterloo.ca*

ABSTRACT


Regular expressions have been studied for nearly 50 years, yet many intriguing problems about their descriptive capabilities remain open. In this paper we sketch some new results and discuss what remains to be solved.

Keywords: Regular expression, finite automaton

1. Introduction

The class of regular languages is one of the most important and best-understood classes of languages in computer science. A regular language can be represented in several different ways (without trying to be exhaustive):

- as the language accepted by
 - a deterministic finite automaton (DFA);
 - a nondeterministic finite automaton (NFA); or
 - a nondeterministic finite automaton with ϵ -transitions (NFA- ϵ);
- as the language specified by
 - a regular expression (RE), allowing the operations of union (+), concatenation (typically represented implicitly by juxtaposition), and Kleene closure (*) [29, 41, 10];
 - a generalized regular expression (GRE), allowing the additional operations of intersection (\cap) and complement (\neg) [41].

 ¹By a fatal mistake by the journal in Issue 2/3 of Vol. 9 (2004) of JALC at pages 233–256 a wrong version of the article has been published, the presented article is the correct one.

²Full version of an invited lecture presented at the 4th Workshop on *Descriptive Complexity of Automata, Grammars and Related Structures* (London, Ontario, Canada, August 21–24, 2002).

The pioneering paper of Meyer and Fischer in 1971 [42] compared the relative descriptonal complexity of these models and others. Since then, many papers on descriptonal complexity of regular languages have been published. Most of these focussed on the state complexity $sc(L)$ for certain languages L , where by state complexity we mean the smallest number of states in a DFA accepting a given language; see, for example, [52]. Less attention has been given to the number of states in NFA's [27, 26], and even less attention has been given to the size of the shortest regular expression for a given language. In this paper, we focus on this last measure.

There does not seem to be any uniform agreement on how to measure the size of a regular expression over an alphabet Σ . One obvious measure is the *ordinary length*: the total number of symbols, including parentheses ([2, p. 396], [25]). For example, the regular expression $(0+10)^*(1+\epsilon)$ has ordinary length 12. Note there is no explicit symbol for concatenation in such a scheme. Another measure is based on the length of the expression converted to (parenthesis-free) reverse polish form, using an explicit operator \bullet for concatenation [53]. Thus the previous regular expression would be written $010\bullet+*1\epsilon+\bullet$ and would have *reverse polish length* 10. We denote reverse polish length of an expression E by $|rpn(E)|$. Evidently reverse polish length is the same as the number of nodes in the syntax tree for the expression.

However, the most useful measure in practice seems to be the total number of *alphabetic symbols*, counted with multiplicity [41, 44, 18, 33]. By an alphabetic symbol we mean an element of Σ , ignoring all operations, parentheses, and the special symbols ϵ and \emptyset . Under this measure, for the expression $(0+10)^*(1+\epsilon)$ would have length 4. We denote the number of alphabetic symbols of an expression E by $|\text{alph}(E)|$.

It does not seem to have been previously observed that these three measures are essentially identical, up to a constant multiplicative factor. We say “essentially” because one can always artificially inflate the ordinary length of a regular expression by adding arbitrarily many multiplicative factors of ϵ , additive factors of \emptyset , etc. In order to avoid such trivialities, we define what it means for a regular expression to be *collapsible*, as follows:

Definition 1 Let E be a regular expression over the alphabet Σ , and let $L(E)$ be the language specified by E . We say E is *collapsible* if any of the following conditions hold:

1. E contains the symbol \emptyset and $|E| > 1$;
2. E contains a subexpression of the form FG or GF where $L(F) = \{\epsilon\}$;
3. E contains a subexpression of the form $F+G$ or $G+F$ where $L(F) = \{\epsilon\}$ and $\epsilon \in L(G)$.

Otherwise, if none of the conditions hold, E is said to be *uncollapsible*.

Note that an expression such as $a+a$ is uncollapsible by our definition, although it can be simplified to just a . However, it is known that the regular expression identities are not finitely axiomatizable (even over a unary alphabet) [13, 1]. Thus it is not

realistic to expect that more rules like the ones given above could achieve ultimate simplification.

Definition 2 If E is an uncollapsible regular expression such that

1. E contains no superfluous parentheses; and
2. E contains no subexpression of the form F^{**} .

then we say E is *irreducible*.

Note that a minimal regular expression for E is uncollapsible and irreducible, but the converse does not necessarily hold.

We now prove the following theorem relating ordinary length, alphabetic length, and reverse polish length of a regular expression.

Theorem 3 Let E be a regular expression over Σ . Then we have

- (a) $|\text{alph}(E)| \leq |E|$;
- (b) If E is irreducible and $|\text{alph}(E)| \geq 1$, then $|E| \leq 11|\text{alph}(E)| - 4$;
- (c) $|\text{rpn}(E)| \leq 2|E| - 1$;
- (d) $|E| \leq 2|\text{rpn}(E)| - 1$;
- (e) $|\text{alph}(E)| \leq \frac{1}{2}(|\text{rpn}(E)| + 1)$;
- (f) If E is irreducible and $|\text{alph}(E)| \geq 1$, then $|\text{rpn}(E)| \leq 7|\text{alph}(E)| - 2$.

We prove (b), leaving the rest to the reader. We need the following lemmas:

Lemma 4 If E is a regular expression without alphabetic symbols, then $L(E) = \{\epsilon\}$ or $L(E) = \emptyset$.

Proof. Clear. □

We now state a lemma due to Ilie & Yu [25]. For a string w we define $|w|_\epsilon$ to be the number of occurrences of the symbol ϵ in w .

Lemma 5 Let E be an uncollapsible regular expression over Σ containing at least one alphabetic symbol. Then $|E|_\epsilon \leq |\text{alph}(E)|$. If equality occurs, then $\epsilon \in L(E)$.

Proof. By induction on the height of the expression tree induced by E . If this height is 0, then, since E contains at least one alphabetic symbol, we have $E = a$ for some $a \in \Sigma$, and the result clearly follows.

Now assume the result is true for all uncollapsible E whose corresponding expression tree has height $< h$; we prove it for all E with an expression tree of height h .

If $E = F^*$ then F is uncollapsible. By induction the desired conclusions hold for F , and hence trivially for E .

If $E = FG$, then since E is uncollapsible, F and G must also be uncollapsible. By the definition of “uncollapsible”, neither F nor G evaluate to ϵ or \emptyset . Hence F and

G contain at least one alphabetic symbol. Then by induction $|F|_\epsilon \leq |\text{alph}(F)|$ and $|G|_\epsilon \leq |\text{alph}(G)|$. Hence $|E|_\epsilon \leq |\text{alph}(E)|$. Furthermore, if $|E|_\epsilon = |\text{alph}(E)|$, then $|F|_\epsilon = |\text{alph}(F)|$ and $|G|_\epsilon = |\text{alph}(G)|$. Hence $\epsilon \in L(F)$ and $\epsilon \in L(G)$, and hence $\epsilon \in L(E)$.

Finally, if $E = F + G$, then again F and G are uncollapsible. First we consider the case where at least one of F, G contains no alphabetic symbol. If both contain no alphabetic symbol, then E is reducible, so we assume without loss of generality that F contains no alphabetic symbol and G does. Then F evaluates to ϵ and by induction $|G|_\epsilon \leq |\text{alph}(G)|$. If $|G|_\epsilon = |\text{alph}(G)|$, then $\epsilon \in L(G)$, and so E is reducible. Thus $|G|_\epsilon < |\text{alph}(G)|$ and hence $|E|_\epsilon \leq |\text{alph}(E)|$. Furthermore $\epsilon \in L(E)$.

If both F and G contain alphabetic symbols, then by induction the number of occurrences of ϵ in F (respectively, G) is \leq the number of alphabetic symbols in F (respectively, G). Hence the same inequality holds for E . Furthermore, if the number of occurrences of ϵ in E equals the number of alphabetic symbols in E , then the same must be true for F and G . Hence $\epsilon \in L(F)$ and $\epsilon \in L(G)$, and hence $\epsilon \in L(E)$. \square

We can now prove Theorem 3 (b).

Proof. Let E be an irreducible regular expression containing n alphabetic symbols, for $n \geq 1$. By Lemma 5, there are at most n occurrences of ϵ in E . Now consider the expression tree for E . Disregarding any occurrences of Kleene $*$, the tree is a binary tree with $\leq 2n$ leaves, and hence has at most $2n - 1$ internal nodes corresponding to occurrences of $+$. It remains to count parentheses and stars. In the worst case every internal node gives rise to two parentheses, which gives $\leq 4n - 2$ parentheses. Finally, each internal node and non- ϵ node could have an associated Kleene $*$, which gives $\leq 3n - 1$ stars. Adding these, we get $2n + (2n - 1) + (4n - 2) + (3n - 1) \leq 11n - 4$. \square

Remark: Ilie & Yu [25] proved a stronger version of inequality (f).

It may be worth noting that there exist irreducible regular expressions with n alphabetic symbols of length $10n - 4$. For example, for $n = 3$ one such expression is

$$(((a_1 + \epsilon)^* + (a_2 + \epsilon)^*)^* + (a_3 + \epsilon)^*)^*.$$

Similar expressions were given by Ilie & Yu [25].

For generalized regular expressions, there is no bound analogous to those of Theorem 3 (b) and (f), as can be seen by considering an expression of the form

$$(\neg\epsilon)(\neg\epsilon)(\neg\epsilon)\cdots(\neg\epsilon)$$

which has no alphabetic symbols at all. For generalized regular expressions, therefore, we must use either the measures of ordinary length or reverse polish length.

There are very few techniques known for bounding the size of a regular expression for a given language. One technique uses the following easy observations:

Proposition 6 *Let L be a nonempty regular language.*

- (a) *If the length of the shortest string in L is n , then $|\text{alph}(E)| \geq n$ for any regular expression E with $L(E) = L$.*
- (b) *If further L is finite, and the length of the longest string in L is n , then $|\text{alph}(E)| \geq n$ for any regular expression E with $L(E) = L$.*

Another technique uses Theorem 10 given below, which states that given an RE E with $|\text{alph}(E)| = n$, there exists an NFA M with at most $n + 1$ states such that $L(M) = L(E)$. Hence a lower bound on the number of states in an NFA accepting L implies a similar lower bound on the length of an equivalent regular expression. To obtain lower bounds on the number of states in an NFA we may use the following result of Birget [7] (rediscovered in a weaker form by Glaister & Shallit [21]):

Theorem 7 *Let L be a regular language. If there exist t pairs of strings $\{(x_1, y_1), \dots, (x_t, y_t)\}$ such that*

- (i) $x_i y_i \in L$ for $1 \leq i \leq t$;
- (ii) $x_i y_j \notin L$ or $x_j y_i \notin L$ for all i, j with $1 \leq i < j \leq t$;

then any NFA accepting L must have at least t states.

Finally, there is the method of Ehrenfeucht & Zeiger [18], which, while powerful, seems restricted in application to finite automata over large alphabets in which each transition is labeled with a unique symbol.

2. Some Examples

In this section we consider some simple families of examples and obtain regular expressions for them, some provably optimal.

Sometimes short regular expressions can be found through an analogue of Horner's rule for evaluating polynomials.

Example 1 Consider the language $R_n := \{0, 0^2, 0^3, \dots, 0^n\}$. We can, of course, specify this language with a regular expression of the form $0 + 00 + 000 + \dots + 0^n$; such a regular expression is of length $\Theta(n^2)$. However, the regular expression $0 + 0(0 + 0(0 + 0(\dots)))$ is of length $\Theta(n)$. Another way to say this is to define $r_1 := 0$, $r_2 := 0 + 00$, and $r_{n+1} := 0 + 0(r_n)$ for $n \geq 2$. This immediately gives $|r_n| = 5n - 6$ for $n \geq 2$.

Example 2 Similarly, consider $S_n := \{0^i 1^i : 0 \leq i \leq n\}$. If we define $s_0 := \epsilon$, $s_1 := \epsilon + 01$, and $s_{n+1} = \epsilon + 0(s_n)1$ for $n \geq 1$, then it is clear that $L(s_n) = S_n$ and furthermore $|s_n| = 6n - 2$ for $n \geq 1$.

Example 3 Consider $T_n := \{0^i 1^j : 0 \leq i \leq j \leq n\}$. If we define $t_0 := \epsilon$, $t_1 := \epsilon + 1 + 01$, and $t_{n+1} = (0 + \epsilon)(t_n)1 + \epsilon$, then $L(t_n) = T_n$ and $|t_n| = 10n - 4$ for $n \geq 1$.

By the technique of the longest string in Proposition 6 (b), each of the bounds in Examples 1–3 is optimal, up to a constant factor.

Sometimes divide-and-conquer is a useful technique for constructing regular expressions. The following example was obtained in discussions with J. Karhumäki, and we thank him for allowing us to reproduce it here.

Example 4 Let $\Sigma = \{0, 1\}$. For a string $w = a_1 a_2 \cdots a_n \in \Sigma^n$, define $\text{omit}(w) = \Sigma^n - \{w\}$. Then for $n \geq 2$ we have

$$\text{omit}(a_1 a_2 \cdots a_n) = \Sigma^{\lfloor \frac{n}{2} \rfloor} \text{omit}(a_{\lfloor \frac{n}{2} \rfloor + 1} \cdots a_n) + \text{omit}(a_1 \cdots a_{\lfloor \frac{n}{2} \rfloor}) a_{\lfloor \frac{n}{2} \rfloor + 1} \cdots a_n.$$

Thus, for example,

$$\text{omit}(1111) = (0 + 1)(0 + 1)((0 + 1)0 + 01) + ((0 + 1)0 + 01)11.$$

This recursively-defined regular expression has $O(n \log n)$ alphabetic symbols.

Example 5 The *binomial language* $B(n, k)$ is defined as follows:

$$B(n, k) = \{x \in \{0, 1\}^n : |x|_1 = k\}.$$

Thus, for example, $B(4, 2) = \{0011, 0101, 0110, 1001, 1010, 1100\}$.

We can find a (not necessarily optimal) regular expression $E(n, k)$ for $B(n, k)$ by divide and conquer. We split a typical string in half; if the first half contains i 1's, then the second half must contain $k - i$ 1's. This gives the following recursive definition for $E(n, k)$:

$$E(n, k) = \begin{cases} 0^n, & \text{if } k = 0; \\ \tilde{E}(n, n - k), & \text{if } k > \frac{n}{2}; \\ \sum_{0 \leq i \leq k} E(\lfloor \frac{n}{2} \rfloor, i) E(\lceil \frac{n}{2} \rceil, k - i), & \text{if } 0 < k \leq \frac{n}{2}. \end{cases}$$

Here \tilde{E} , where E is a regular expression, changes each 0 to 1 and vice versa, and leaves other characters unchanged. For example, $E(4, 2) = 0011 + (01 + 10)(01 + 10) + 1100$.

Let $L(n, k) = |\text{alph}(E(n, k))|$, the alphabetic length of $E(n, k)$. Then

Theorem 8 For each fixed $k \geq 0$, we have $L(n, k) = O(n(\log n)^k)$, where the implied constant depends on k .

Proof. We begin by proving the inequality

$$L(2^n, k) \leq 2^n \binom{n + k}{k} \tag{1}$$

for $n \geq 0$ and $0 \leq k \leq 2^n$. We prove this by induction on $n + k$. The base case is $n = k = 0$. Then $L(2^n, k) = L(1, 0) = 1 \leq 2^0 \binom{0}{0}$.

Now assume the result is true for $n + k < N$; we prove it for $n + k = N$. If $k > 2^{n-1}$ then $k' = 2^n - k < 2^n - 2^{n-1} = 2^{n-1} < k$. Hence

$$L(2^n, k) = L(2^n, k') \leq 2^n \binom{n + k'}{k'} \leq 2^n \binom{n + k}{k}.$$

Otherwise $k \leq 2^{n-1}$. Then from the recursive definition for $E(n, k)$, we have

$$\begin{aligned} L(2^n, k) &= \sum_{0 \leq i \leq k} (L(2^{n-1}, i) + L(2^{n-1}, k - i)) \\ &= 2 \sum_{0 \leq i \leq k} L(2^{n-1}, i) \\ &\leq 2 \sum_{0 \leq i \leq k} 2^{n-1} \binom{n-1+j}{j} \\ &= 2^n \binom{n+k}{k}. \end{aligned}$$

Now, letting $n' = \lceil \log_2 n \rceil$ and using Eq. (1), we have

$$\begin{aligned} L(n, k) &\leq L(2^{n'}, k) \\ &\leq 2^{n'} \binom{n'+k}{k} \\ &\leq 2^{n'} \frac{(n'+k)^k}{k!} \\ &\leq 2n \frac{((\log_2 n) + k + 1)^k}{k!} \\ &= O(n(\log n)^k). \end{aligned} \quad \square$$

As an aside, the numbers $L(n, k)$ have some very interesting combinatorial properties, which we summarize here:

- Theorem 9** (a) $L(n, k) = L(\lfloor \frac{n}{2} \rfloor, k) + L(\lceil \frac{n}{2} \rceil, k) + L(n, k - 1)$ for $1 \leq k \leq \frac{n}{2}$.
 (b) $L(2n, n) = L(2n, j) + L(2n, n - j - 1)$ for $0 \leq j < n$.
 (c) $L(4n, 2n) = 2(n + 1)L(2n, n)$ for $n \geq 1$.
 (d) $L(2^{n+1}, 2^n) = 2^{n+2}(2^0 + 1)(2^1 + 1) \cdots (2^{n-1} + 1)$ for $n \geq 1$.
 (e) $L(2^{n+1}, 2^n) = \Theta(2^{\frac{n^2}{2} + \frac{n}{2}})$.

Proof. (a) From the definition we have

$$L(n, k) = \sum_{0 \leq i \leq k} \left(L(\lfloor \frac{n}{2} \rfloor, i) + L(\lceil \frac{n}{2} \rceil, k - i) \right)$$

and

$$L(n, k - 1) = \sum_{0 \leq i \leq k-1} \left(L(\lfloor \frac{n}{2} \rfloor, i) + L(\lceil \frac{n}{2} \rceil, k - 1 - i) \right).$$

Subtracting the second from the first, we get

$$L(n, k) - L(n, k - 1) = L(\lfloor \frac{n}{2} \rfloor, k) + L(\lceil \frac{n}{2} \rceil, k).$$

(b) We have

$$\begin{aligned}
& L(2n, j) + L(2n, n - j - 1) \\
&= \sum_{0 \leq i \leq j} (L(n, i) + L(n, j - i)) + \sum_{0 \leq i \leq n-j-1} (L(n, i) + L(n, n - j - i - 1)) \\
&= \sum_{0 \leq i \leq j} L(n, i) + \sum_{0 \leq i \leq n-j-1} L(n, j + i + 1) + \sum_{0 \leq i \leq n-j-1} L(n, i) \\
&\quad + \sum_{0 \leq i \leq j} L(n, n - j + i) \\
&= \sum_{0 \leq i \leq j} L(n, i) + \sum_{j+1 \leq i \leq n} L(n, i) + \sum_{0 \leq i \leq n-j-1} L(n, i) + \sum_{n-j \leq i \leq n} L(n, i) \\
&= 2 \sum_{0 \leq i \leq n} L(n, i) \\
&= L(2n, n).
\end{aligned}$$

(c) From (b) we have

$$L(2n, j) + L(2n, n - j - 1) = L(2n, j) + L(2n, n + j + 1) = L(2n, n).$$

Hence

$$L(4n, 2n) = 2 \sum_{0 \leq i \leq 2n} L(2n, i) = 2(n+1)L(2n, n).$$

(d) Iterate (c).

(e) From (d) we have

$$2^{n+2}2^{0+1+\dots+n-1} \leq L(2^{n+1}, 2^n) \leq 2^{n+2}2^{0+1+\dots+n-1}\alpha$$

where $\alpha = (1 + 2^{-0})(1 + 2^{-1})(1 + 2^{-2}) \dots \doteq 4.768462$. □

It is possible to prove that our expressions for $E(n, 0)$ and $E(n, 1)$ are optimal, up to a constant multiplicative factor, but we do not know this for $E(n, k)$, $k \geq 2$.

3. Conversion Problems

In this section we consider problems about converting from one method of representing a regular language to another.

Converting from a regular expression to a DFA or NFA has been well-studied. We define an NFA to be *non-returning* if there are no transitions entering the initial state. Then we have the following theorem ([33, 34],[22, Thm. 16]).

Theorem 10 *Let E be a regular expression with $|\text{alph}(E)| = n$. Then there exists a non-returning NFA accepting $L(E)$ with $\leq n + 1$ states, and a DFA accepting E with $\leq 2^n + 1$ states.*

It is easy to see that this upper bound for RE to NFA conversion is tight, even in the unary case (as can be seen by considering the regular expression specifying any single word of length n). Surprisingly, however, it does not seem to be known if the upper bound for RE to DFA conversion is tight; most likely it is not.³ The regular expression $E_r := (0 + (01^*)^{r-1}0)^*$ has $2r$ alphabetic symbols and Leung [35] proved that the minimal DFA for $L(E_r)$ has 2^r states. This implies a worst case lower bound of $2^{\frac{r}{2}}$ for RE to DFA conversion.

However, this bound can be improved somewhat by a simple variation on Leung's expression.

Theorem 11 *Let F_r denote the regular expression*

$$(0^*(01^*)^{r-1}0)^*.$$

Then this expression has $2r$ alphabetic symbols and the minimal DFA for $L(F_r)$ has $2^r + 2^{r-2}$ states. This gives a worst case lower bound of $\frac{5}{4} \cdot 2^{\frac{r}{2}}$ for RE to DFA conversion.

Proof. We give a sketch of the proof. First, we can create an NFA $M_r = (Q, \Sigma, \delta, q_0, F)$ that accepts $L(F_r)$, as follows:

$$\begin{aligned} Q &= \{q_0, q_1, \dots, q_r, q_{r+1}\} \\ \Sigma &= \{0, 1\} \\ F &= \{q_0, q_{r+1}\} \\ \delta(q_0, 0) &= \delta(q_1, 0) = \{q_1, q_2\} \\ \delta(q_0, 1) &= \delta(q_1, 1) = \delta(q_{r+1}, 0) = \delta(q_{r+1}, 1) = \emptyset \\ \delta(q_i, 0) &= \{q_{i+1}\}, \quad 1 < i < r \\ \delta(q_i, 1) &= \{q_i\}, \quad 1 < i \leq r \\ \delta(q_r, 0) &= \{q_0, q_1, q_{r+1}\}. \end{aligned}$$

When we apply the subset construction to this NFA to get a DFA with states being subsets of Q , we find that only certain subsets of Q are reachable: namely, sets of the form

- (a) $\{q_0\}$;
- (b) $\{q_0, q_1, q_{r+1}\} \cup X$, where $X \subseteq \{q_2, q_3, \dots, q_r\}$;
- (c) $\{q_1, q_2\} \cup Y$, where $Y \subseteq \{q_3, q_4, \dots, q_r\}$;
- (d) Z , where $Z \subseteq \{q_2, q_3, \dots, q_r\}$.

Let us see why these states are reachable. For any subset $P \subseteq Q$ define the string $x_P := 0x_r 0x_{r-1} 0 \dots 0x_2 0$ where

$$x_i = \begin{cases} \epsilon, & \text{if } q_i \in P; \\ 1, & \text{otherwise.} \end{cases}$$

Then it is not difficult to see that

³Using a non-standard definition of the length of a regular expression, Leiss [32, 34] proved the upper bound for RE to DFA conversion is tight.

- (a) $\delta(q_0, \epsilon) = \{q_0\}$;
- (b) If $P = \{q_0, q_1, q_{r+1}\} \cup X$, where $X \subseteq \{q_2, q_3, \dots, q_r\}$, then $\delta(q_0, 0^{r+1}x_P) = P$;
- (c) If $P = \{q_1, q_2\} \cup Y$, where $Y \subseteq \{q_3, q_4, \dots, q_r\}$, then $\delta(q_0, 0^{r+1}1x_P) = P$;
- (d) If $P \subseteq \{q_2, q_3, \dots, q_r\}$, then $\delta(q_0, 0^{r+1}1x_P1) = P$.

Furthermore, all of these subsets are pairwise distinguishable (in the sense of [24, p. 68]), with the exception that $\{q_0\}$ is equivalent to $\{q_0, q_1, q_{r+1}\}$. The total number of states needed is therefore $2^{r-1} + 2^{r-2} + 2^{r-1} = \frac{5}{4}2^r$. \square

Open Problem 1 What is the worst case in RE to DFA conversion for alphabet size ≥ 2 ?

One can also consider variations on Open Problem 1 involving restricted classes of regular expressions. For example, consider regular expressions of the form

$$(w_1 + w_2 + \dots + w_j)^*,$$

where each w_i is a word. Over a unary alphabet, the deterministic state complexity of the corresponding language is bounded by $O(n^2)$, where $n = \sum_{1 \leq i \leq j} |w_i|$. This follows immediately from classical results on the so-called Frobenius problem [9] which asks, given a list of j integers a_1, a_2, \dots, a_j with $\gcd(a_1, a_2, \dots, a_j) = 1$, find the largest integer not representable as a non-negative integer linear combination of the a_i .

Currently, only the trivial upper bound of 2^n is known for the deterministic state complexity in the case of non-unary alphabets. Finding a tight upper bound can be thought of as the non-commutative generalization of the Frobenius problem.

For a long time, it was thought that the true state complexity of this problem was $O(n^2)$. Recently, however, the third author found the following class of examples that achieves $2^{\Omega(\sqrt{n})}$.

Let t be an integer ≥ 2 , and define strings as follows:

$$\begin{aligned} y &:= 01^{t-1}0 \\ x_i &:= 1^{t-i-1}01^{i+1}, \quad 0 \leq i \leq t-2 \end{aligned}$$

Let $S_t := \{0, x_0, x_1, \dots, x_{t-2}, y\}$. Thus, for example,

$$S_6 := \{0, 1111101, 1111011, 1110111, 1101111, 1011111, 0111110\}.$$

Theorem 12 S_t^* has state complexity $3t2^{t-2} + 2^{t-1}$.

The proof of this theorem is rather complicated, so we just sketch a proof of the following slightly weaker result:

Theorem 13 $\text{sc}(S_t^*) \geq 2^{t-2}$.

Proof. First, we create an NFA $M_t = (Q, \{0, 1\}, \delta, p_0, F)$ that accepts S_t^* . This NFA has $3t - 1$ states

$$Q = \{p_0, p_1, \dots, p_t, q_1, q_2, \dots, q_{t-1}, r_1, r_2, \dots, r_{t-1}\}$$

with only one final state $F = \{p_0\}$. The transition function δ is defined as follows:

$$\begin{aligned} \delta(p_0, 0) &= \{p_0, p_1\} \\ \delta(p_i, 1) &= \{p_{i+1}\}, \quad 1 \leq i \leq t-1 \\ \delta(p_n, 0) &= \{p_0\} \\ \delta(p_0, 1) &= \{q_1\} \\ \delta(q_i, 1) &= \{q_{i+1}\}, \quad 1 \leq i \leq t-2 \\ \delta(q_i, 0) &= \{r_i\}, \quad 1 \leq i \leq t-1 \\ \delta(r_i, 1) &= \{r_{i+1}\}, \quad 1 \leq i \leq t-2 \\ \delta(r_{n-1}, 1) &= \{p_0\} \end{aligned}$$

For example, the NFA M_6 is illustrated in Figure 1.

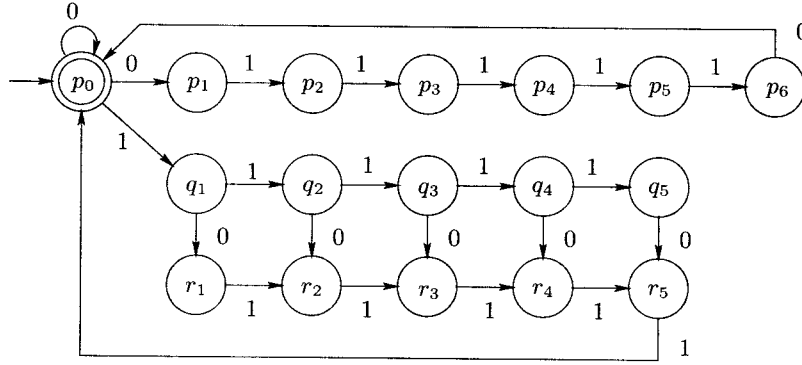


Figure 1: The NFA M_6

Let T be any subset of $\{r_1, r_2, \dots, r_{t-2}\}$, and write $T = \{r_{i_1}, r_{i_2}, \dots, r_{i_j}\}$ for j indices $1 \leq i_1 < i_2 < \dots < i_j \leq t-2$. We claim that the 2^{t-2} strings

$$y \ x_{t-2}y \ x_{t-3}x_{t-2}y \ x_{t-4}x_{t-3}x_{t-2}y \ \cdots \ x_1x_2 \cdots x_{t-2}y \ x_{i_1}x_{i_2} \cdots x_{i_j}y$$

are pairwise inequivalent under the Myhill-Nerode equivalence relation.

To show this, we first argue that any subset of states of the form $T' := \{p_0, r_{t-1}\} \cup T$, where T is as in the previous paragraph, is reachable from p_0 . We claim that the following path reaches T' :

$$\begin{aligned} \{p_0\} &\xrightarrow{y} \{p_0, r_{t-1}\} \xrightarrow{x_{t-2}y} \{p_0, r_{t-2}, r_{t-1}\} \xrightarrow{x_{t-3}x_{t-2}y} \{p_0, r_{t-3}, r_{t-2}, r_{t-1}\} \\ &\xrightarrow{x_{t-4}x_{t-3}x_{t-2}y} \cdots \xrightarrow{x_1x_2 \cdots x_{t-2}y} \{p_0, r_1, r_2, \dots, r_{t-1}\} \\ &\xrightarrow{x_{i_1}x_{i_2} \cdots x_{i_j}y} \{p_0, r_{i_1}, r_{i_2}, \dots, r_{i_j}, r_{t-1}\}. \end{aligned}$$

Finally, we argue that each of these subsets of states is inequivalent. This is because given two distinct such subsets, say T' and T'' , there must be an r_i , $1 \leq i \leq t-2$, that is contained in one (say T') but not the other. Then reading the string 1^{t-i} takes T' to p_0 , but not T'' . \square

Since the alphabetic length of the strings in S_t is $n = t^2 + t + 1$, it now follows that this example has state complexity $2^{\Omega(\sqrt{n})}$.

Let us now turn to the unary case of RE to DFA conversion. Define

$$g(n) := \max_{\sum n_i \leq n} \text{lcm}(n_1, n_2, \dots).$$

It is known that $g(n) = e^{\sqrt{n \log n}(1+o(1))}$ (see, for example, [43]).

Theorem 14 *Let E be a regular expression over a unary alphabet with $|\text{alph}(E)| = n$. Then there exists a DFA accepting E with $\leq g(n) + (n-1)^2 + 2$ states. Furthermore for each n there exists a regular expression E with $|\text{alph}(E)| = n$ such that the smallest equivalent DFA has $g(n)$ states.*

Note that $e^{\sqrt{n \log n}}$ grows *much* more rapidly than $(n-1)^2 + 2$, so the upper and lower bounds are (relatively) quite close.

Proof. By Theorem 10 we know there is a non-returning NFA of $n+1$ states accepting $L(E)$. By a result of Mandl [39] this means there is a DFA with $g(n) + (n-1)^2 + 2$ states accepting $L(E)$.

For the lower bound, we first find the partition $n = n_1 + n_2 + \dots + n_t$ which maximizes $g(n)$. By a well-known result [43, p. 501] we may assume the n_i are powers of distinct primes, say $n_i = p_i^{e_i}$ for $1 \leq i \leq t$. Now let $E = (a^{n_1})^* + \dots + (a^{n_t})^*$; this has n alphabetic symbols.

It is easy to see that there is a DFA with $n_1 n_2 \dots n_t = g(n)$ states accepting $L(E)$. Such an automaton is cyclic, and we may assume the states are numbered $q_0, q_1, \dots, q_{g(n)-1}$. To see it is minimal, it suffices to show that for each i, j with $0 \leq i < j < g(n)$ the states q_i and q_j are distinguishable in the sense of [24, p. 68].

Since $0 < j-i < g(n)$, there exists at least one n_k such that $n_k \nmid j-i$. By the Chinese Remainder Theorem, there exists t such that $t+j \equiv 0 \pmod{n_k}$ but $t+i \equiv 1 \pmod{n_l}$ for all $l \neq k$. Now $t+i \equiv i-j \pmod{n_k}$, and so $t+i \not\equiv 0 \pmod{n_k}$. It follows that q_i and q_j are distinguishable states, being distinguished by the string a^t . \square

We now turn to the converse problem, which has not received as much attention: converting from a DFA or NFA to an RE. For unary languages, we have the following easy result:

Theorem 15 *If L is a unary regular language, accepted by a DFA M with n states, then it is specified by an RE of size $O(n)$. Furthermore, there exist infinitely many regular languages for which the smallest corresponding regular expression is of size $\Omega(n)$.*

Proof. We may assume without loss of generality that the transition diagram of M is connected. This transition diagram has a “tail” of $t \geq 0$ states and a “cycle” of $c \geq 1$ states, with $n = t + c$. It follows that there exist sets $A \subseteq \{\epsilon, a, \dots, a^{t-1}\}$ and $B \subseteq \{\epsilon, a, \dots, a^{c-1}\}$ such that

$$L(M) = A + Ba^t(a^c)^* \quad (2)$$

See, for example, [45]. Now, using the analogue of Horner’s rule mentioned above, we can generate A with a regular expression of length $O(t + 1)$, and B with a regular expression of length $O(c)$.

For the lower bound, consider the regular language $\{a^{n-1}\}$. This is accepted by a DFA with $n + 1$ states, and specified by a regular expression of length $n - 1$. No smaller regular expression will suffice, by Proposition 6. \square

Theorem 16 *If L is a unary regular language, accepted by an NFA with n states, then it is specified by a regular expression E with $\text{alph}(E) \leq 2n^2 + 4n$.*

Proof. A theorem of Chrobak [12] says that for any unary NFA, there exists an equivalent NFA in Chrobak normal form (where there is a “tail” of at most $n^2 + n$ states, ending with a single nondeterministic state, followed by cycles with at most n states in total). Using the analogue of Horner’s rule mentioned above in § 2 this can be converted into a regular expression with the stated length. \square

When implemented, this method appears to give results that are significantly smaller than those produced by Grail [40].

Open Problem 2 Does there exist an infinite family of unary regular languages such that the blow-up from number of states in an NFA to length of a regular expression is quadratic?

Note that an $O(n/\log n)^n$ upper bound is known for the number of distinct languages accepted by unary NFA’s with n states [47]. If a matching lower bound could be proved, this would show that there are unary regular languages, accepted by an NFA with n states, that require $\Omega(n \log n)$ -length regular expressions.

For languages over arbitrary alphabets we have the following upper bound, essentially due to McNaughton & Yamada [41]:

Theorem 17 *If L is a regular language, accepted by a DFA or NFA $M = (Q, \Sigma, \delta, q_1, F)$ where $|Q| = n$ and $|\Sigma| = k$, it can be specified by an RE E such that $|\text{alph}(E)| \leq nk4^n$.*

Proof. We use the McNaughton-Yamada algorithm [41, 24]. We assume $Q = \{q_1, q_2, \dots, q_n\}$. The algorithm defines a sequence of regular expressions $\alpha_{i,j}^{(l)}$ which specify all words taking the automaton from state q_i to state q_j without passing

through (i.e., both entering and leaving) a state numbered higher than q_l . Then we have

$$\alpha_{i,j}^{(l)} := \alpha_{i,j}^{(l-1)} + \alpha_{i,l}^{(l-1)} \alpha_{l,l}^{(l-1)*} \alpha_{l,j}^{(l-1)}.$$

If we define $T_l = \max_{i,j} |\text{alph}(\alpha_{i,j}^{(l)})|$ then clearly $T_l \leq 4T_{l-1}$. Since $T_0 \leq k$, it follows that $T_n \leq k4^n$. Now $L(M)$ can be specified by the union of the $\alpha_{1,i}^{(n)}$ for all i with $q_i \in F$. The bound follows. \square

Although this upper bound seems large, other methods of NFA to RE conversion (such as state elimination) seem to generate even larger upper bounds.

The preceding upper bound can be improved in certain special cases. For example, if the transition diagram of the NFA has no long paths, we can get a better bound, as the following theorem shows.

Theorem 18 *Let G be a edge-labeled directed graph with n vertices and outdegree bounded above by D . Suppose the number of edges in a longest simple path (not repeating vertices or edges) in G that starts with vertex u is at most k . (By the length of the path we mean the number of edges.) Then for all vertices v , there is a regular expression E denoting all walks from u to v with $|\text{alph}(E)| = O(D^{k+1}n^k)$.*

Proof. Fix a vertex s in G . We compute a set of regular expressions $t_{s,q}$ corresponding to walks from s to vertices q of G . The procedure is recursive. First remove all edges into s , and call the new graph G' .

Next suppose there is a directed edge $s \rightarrow p$ labeled a , with $p \neq s$. Let H' be the subgraph of G' containing all vertices reachable from p in G' . Since we have removed all edges into s , the longest simple starting with p in H' is of length $\leq k-1$. We now recursively construct expressions $r_{p,j}$ which are the regular expressions corresponding to walks from p to vertex j in H' . We do this for each $p \neq s$ and edge $s \rightarrow p$.

Now we use the $r_{p,j}$ to construct expressions for walks from s to vertices j in G . Let j_1, j_2, \dots, j_l be the vertices in G distinct from s having an edge to s in G , labeled a_1, a_2, \dots, a_l respectively. First, we construct regular expressions $t_{s,s}$ corresponding to all walks from s to itself. Let s have b self-loops with labels c_1, \dots, c_b . Let p_1, p_2, \dots, p_f be vertices distinct from s having an edge from s with labels d_1, d_2, \dots, d_f , respectively. Then we define

$$t_{s,s} := \left(\left(\sum_{1 \leq i \leq b} c_i \right) + \sum_{1 \leq i \leq f} \left(d_i \sum_{1 \leq m \leq l} r_{p_i, j_m} a_m \right) \right)^*. \quad (3)$$

Finally, we construct $t_{s,q}$ for $q \neq s$. We define

$$t_{s,q} := t_{s,s} + \sum_{1 \leq i \leq f} d_i r_{p_i, q}. \quad (4)$$

Correctness is left to the reader.

It remains to estimate the size of the expressions $t_{s,s}$ and $t_{s,q}$. Let $g(k)$ denote the maximum number of alphabetic symbols in any $t_{s,q}$ over all pairs $s, q \in G$ (including

the case where $s = q$), over all G with outdegree bounded by D , such that the length of the longest simple path starting from s to a vertex in G is $\leq k$. Then we have, using (3) and the inequalities $b \leq D$, $f \leq D$, and $l \leq n - 1$, that $|\text{alph}(t_{s,s})| \leq D + D(1 + (n-1)(g(k-1) + 1))$. Furthermore $|\text{alph}(t_{s,q})| \leq D(1 + g(k-1)) + |\text{alph}(t_{s,s})|$. It follows that $g(k) \leq D(n+2) + Dng(k-1)$. Now a simple induction shows

$$g(k) \leq (Dn)^k g(0) + D(n+2) \frac{(Dn)^k - 1}{Dn - 1}.$$

Since $g(0) \leq D$, the result follows. \square

We observe that if $k = o(n/\log n)$, then this bound is superior to that in Theorem 17.

Another improvement on the upper bound in Theorem 17 arises from other special classes of transition diagrams.

Theorem 19 *Let $M = (Q, \Sigma, \Delta, q_0, F)$ be an NFA with r states, such that its transition diagram can be drawn in the plane with no edges crossing. Then there exists a regular expression E for $L(M)$ with $|\alpha(E)| \leq e^{O(\sqrt{r})}$.*

Proof. The basic idea is divide-and-conquer. Instead of computing the regular expression for $L(M)$ by state elimination or the dynamic programming approach of McNaughton-Yamada, which are iterative methods working one state a time, we divide the transition diagram for M into two pieces and work on each piece separately. This is similar to “nonserial dynamic programming” [37].

We use the following well-known theorem of Lipton & Tarjan [36]: if G is an undirected planar graph with r vertices, then the vertices of G can be written as the (not necessarily disjoint) union of two sets A and B such that

- (a) there are no edges from $A - B$ to $B - A$,
- (b) $|A - B|, |B - A| \leq 2r/3$, and
- (c) $|A \cap B| \leq \sqrt{8r}$.

(Djidjev [16] improved the “8” in part (c) to “6”, and Alon, Seymour, & Thomas [4] further improved this to $9/2$.) We apply this theorem to the underlying undirected graph of the transition diagram for M for all sufficiently large M . (For small M we use the McNaughton-Yamada algorithm instead.)

Here is the idea: we find a planar partition of the vertices of the transition diagram for M into A and B . Let $C = A \cap B$, $t = |C|$, and $C = \{c_1, c_2, \dots, c_t\}$. Now suppose q_u, q_v are both vertices of A ; if $q_u \in A$ and $q_v \in B - A$ a similar argument applies.

We can now decompose any walk connecting q_u to q_v as follows: either we go from q_u to q_v without leaving A , or the walk eventually leaves A and enters $B - A$. If the latter, it must leave A through a vertex of C , since there are no edges connecting $A - B$ to $B - A$. Now the walk is in $B - A$. Since $q_v \in A$, at some point the walk must leave $B - A$ and return to A , and it can do so only through a vertex of C . This “ping-pong” between A and $B - A$ can occur arbitrarily many times. Eventually we leave $B - A$ and return to A , and finally we enter q_v .

We can now create an NFA $M' = (Q', \Delta, p_0, \delta, F)$ that encodes this walk in its transitions, with a single symbol representing a sequence of transitions for M . This NFA has $2t + 3$ states, and is defined as follows:

$$\begin{aligned} Q' &= \{p_0, p_1, \dots, p_t, p'_1, \dots, p'_t, f_1, f_2\}, \\ \Delta &= \{\langle c_i, c_j, A \rangle, \langle c_i, c_j, B - A \rangle : 1 \leq i, j \leq t\} \\ &\quad \cup \{[u, c_i, A], [c_i, v, A] : 1 \leq i \leq t\} \\ &\quad \cup \{[u, v, A]\}, \\ \delta(p_0, [u, v, A]) &= f_1, \\ \delta(p_0, [u, c_i, A]) &= p_i, \quad 1 \leq i \leq t, \\ \delta(p_i, \langle c_i, c_j, B - A \rangle) &= p'_j, \quad 1 \leq i, j \leq t, \\ \delta(p'_i, [c_i, c_j, A]) &= p_j, \quad 1 \leq i, j \leq t, \\ \delta(p'_i, [c_i, v, A]) &= f_2, \\ F &= \{f_1, f_2\}. \end{aligned}$$

Here the symbol $[x, y, S]$ is intended to represent all walks from x to y in the transition diagram of M , such that all internal vertices lie in S . The symbol $\langle x, y, S \rangle$ is similar, but adds the extra condition that this walk has at least one vertex in S .

Since $|Q'| = 2t + 3$ and $|\Sigma'| = 2t^2 + 2t + 1$, it follows from the proof Theorem 17 that we can construct a regular expression E for $L(M')$ with at most $1 + (2t^2 + 2t + 1)4^{2t+3}$ alphabetic symbols. (The “1” comes from the path from p_0 to f_1 , and the other term corresponds to the path from p_0 to f_2 .) Now for each symbol of the form $[x, y, S]$ we recursively determine regular expressions specifying the labels of walks from x to y with internal nodes in S . We then substitute these regular expressions for $[x, y, S]$ in E .

A slightly different procedure is needed for symbols of the form $\langle c_i, c_j, B - A \rangle$. For these we create a digraph G with vertices $(B - A) \cup \{c_1, \dots, c_t\} \cup \{c'_1, \dots, c'_t\}$ and labeled edges those induced by the subgraph $B - A$, together with edges

$$\{c_i \xrightarrow{a} q : q \in B - A\}$$

and

$$\{q \xrightarrow{a} c'_i : \delta(q, a) = c_i\}.$$

We then use the procedure of Theorem 17 to find a regular expression for all walks from c_i to c'_j in G . This digraph has $|B| + |C| \leq 2r/3 + 2\sqrt{8r}$ states. Again, we substitute these regular expressions for $\langle c_i, c_j, B - A \rangle$ in E .

If we let $T(n)$ denote the length of the resulting regular expression denoting all walks between two vertices in an NFA with n states, we get the recurrence relation

$$T(n) \leq 4^{O(\sqrt{n})} T(2n/3 + 2\sqrt{8n}),$$

which has the solution $T(n) \leq e^{O(\sqrt{n})}$. Since the expression we construct represents all paths going from q_u to q_v , it follows that we can construct such an expression that represents all paths from the original start state to the original final states. Then the resulting regular expression specifies $L(M)$, and has size $\leq nT(n) = e^{O(\sqrt{n})}$. \square

Not much seems to be known about the class of languages possessing planar DFA's. It is known that inherently nonplanar DFA's exist (i.e., DFA's for which no planar DFA accepts the same language) [8].

We note that the $e^{O(\sqrt{n})}$ upper bound also holds for any family of NFA's whose transition diagrams are of bounded genus [20], or exclude any fixed complete graph K_i as a minor [3].

As an example, consider the languages

$$L_n = \{x \in \{a, b\}^* : |x|_a \equiv |x|_b \equiv 0 \pmod{n}\}$$

for $n \geq 1$. The language L_n can be accepted by an NFA with n^2 states, as illustrated in Figure 2 for the case $n = 4$.

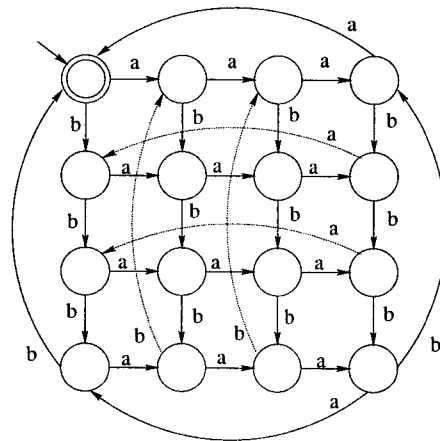


Figure 2: NFA for the language $L_4 = \{x \in \{a, b\}^* : |x|_a \equiv |x|_b \equiv 0 \pmod{4}\}$

Since this transition diagram can be embedded in a torus (which is of genus 2), it follows that there are regular expressions of size $e^{O(n)}$ for L_n .

It seems like divide-and-conquer, combined with a heuristic graph separator algorithm (e.g., [46]), or an approximation algorithm (e.g., [19]) would be a powerful technique for creating relatively short regular expressions for arbitrary automata in software packages such as Grail [49].

Finally, the upper bound of Theorem 17 can be improved in the case of finite languages, as we will prove in a corollary of the result below.

Theorem 20 *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA or NFA with t states, over an alphabet of size $k = |\Sigma|$. Write $Q = \{q_0, q_1, \dots, q_{t-1}\}$. Then for all $q_i, q_j \in Q$ and $n \geq 1$ there is a regular expression E denoting all walks of length n from q_i to q_j with $|\text{alph}(E)| < k(t+1)n^{(\log_2 t)+1}$. The same bound holds for a regular expression denoting all walks of length $\leq n$.*

We need the following lemma:

Lemma 21 Suppose $V(n) = s(V(\lfloor \frac{n}{2} \rfloor) + V(\lceil \frac{n}{2} \rceil))$ for $n \geq 2$. Then

$$V(n) = V(1)(2bs^{a+1} + (2^a - b)s^a)$$

for all $n \geq 1$, where $n = 2^a + b$, $0 \leq b < 2^a$.

Proof. By induction on n . The result is clearly true for $n = 1$. Now assume it is true for all $n' < n$. We prove it for n . By induction we have

$$\begin{aligned} V(\lfloor \frac{n}{2} \rfloor) &= V(2^{a-1} + \lfloor \frac{b}{2} \rfloor) \\ &= V(1)(2\lfloor \frac{b}{2} \rfloor s^a + (2^{a-1} - \lfloor \frac{b}{2} \rfloor)s^{a-1}). \end{aligned}$$

Similarly,

$$V(\lceil \frac{n}{2} \rceil) = V(1)(2\lceil \frac{b}{2} \rceil s^a + (2^{a-1} - \lceil \frac{b}{2} \rceil)s^{a-1}).$$

Now the result follows from $V(n) = s(V(\lfloor \frac{n}{2} \rfloor) + V(\lceil \frac{n}{2} \rceil))$. \square

Now we can prove Theorem 20:

Proof. We use an old idea that can be found, for example, Burks & Wang [11], Glushkov [22, Thm. 22], and Ehrenfeucht & Zeiger [18].

Create a $t \times t$ matrix $M = (m_{i,j})_{0 \leq i,j < t}$ such that $m_{i,j}$ is a regular expression specifying all labeled edges taking A from state q_i to state q_j . We can now define a multiplication for such matrices, where ordinary addition is replaced by $+$ and ordinary multiplication by concatenation. More precisely, if $N = (n_{i,j})_{0 \leq i,j < t}$, then $MN = (u_{i,j})_{0 \leq i,j < t}$, where

$$u_{i,j} = (m_{i,1})(n_{1,j}) + \cdots + (m_{i,t-1})(n_{t-1,j}). \quad (5)$$

Then it is easy to see that if $M^k = (m_{i,j}^{(k)})_{0 \leq i,j < t}$, then $m_{i,j}^{(k)}$ is a regular expression specifying the labels of all length- k walks from q_i to q_j .

Now define $S(n) = \max_{0 \leq i,j < t} |\text{alph}(m_{i,j}^{(n)})|$. From Eq. (5) we see that

$$S(n) \leq t(S(\lfloor \frac{n}{2} \rfloor) + S(\lceil \frac{n}{2} \rceil)).$$

Now use Lemma 21 with $V(1) \leq k$ and $s = t$. We get

$$\begin{aligned} S(n) &\leq k(2bt^{a+1} + (2^a - b)t^a) \\ &< k(nt^{1+\log_2 n} + nt^{\log_2 n}) \\ &= kn(t+1)t^{\log_2 n} \\ &= kn(t+1)n^{\log_2 t} \\ &= k(t+1)n^{(\log_2 t)+1}, \end{aligned}$$

as desired. Here we have used the inequalities $b < \frac{n}{2}$, $2^a - b \leq n$, and $a \leq \log_2 n$.

Now consider a regular expression for all strings of length $\leq n$. Our analysis above can then be repeated without change, except for the following: we add ϵ to each entry on the diagonal of M . \square

Corollary 22 *If A is a DFA or NFA with r states over a k -letter alphabet, and $L(A)$ is finite, then there is a regular expression E specifying $L(A)$ with*

$$|\text{alph}(E)| \leq kr(r+1)(r-1)^{(\log_2 r)+1}.$$

Proof. Such a machine accepts no strings of length $\geq r$. Use Theorem 20 with $t = r$ and $n = r - 1$. Finally, the strings accepted by A correspond to paths from the initial state to all final states, so we get an extra multiplicative factor of r . \square

After this abundance of upper bounds, we state the following

Open Problem 3 Do there exist a constant c , a fixed alphabet Σ , and a family of languages L_n ($n \geq 1$) defined over Σ , accepted by DFA's (resp. NFA's) with $f(n)$ states, such that the smallest corresponding regular expression has size $2^{cf(n)}$?

Trivial lower bounds follow from basic enumeration results for the languages accepted by DFA's and NFA's [17].) Namely, there exist DFA's over an alphabet of size 2 or greater where the corresponding regular expressions have length $\Omega(n)$, and there exist NFA's over an alphabet of size 2 or greater where the corresponding regular expressions have length $\Omega(n^2)$. Ehrenfeucht & Zeiger [18] gave a family of examples with n states achieving exponential blow-up, but their alphabet size grew quadratically with n . Only in 2000 was any nontrivial lower bound proved for fixed alphabet size: building on the approach of Ehrenfeucht & Zeiger, Waizenegger [51] gave an example of an NFA over a 4-letter alphabet such that the smallest corresponding regular expression has at least $\Omega(2^{\sqrt[3]{n}})$ alphabetic symbols.

In particular, it would be interesting to determine the size of the minimum regular expressions for the following languages:

- $\overline{\Sigma^* w \Sigma^*}$ for $w \in \{0, 1\}^*$. This language can be accepted by a DFA with $|w| + 1$ states. For certain w (e.g., $w = 0^n$), there are short regular expressions but for other w (e.g., the finite Fibonacci words [6]), the expressions appear to be long.
- The language of strings over $\{0, 1\}$ representing numbers in base 2 divisible by n . This language can be accepted by $e + r$ states, where $n = 2^e \cdot r$, r odd. Regular expressions for this language seem large, particularly when n is a prime.

New techniques seem to be needed here. Perhaps techniques from circuit complexity may be helpful. As an example, consider the language

$$L_n = \{w \in \{0, 1\}^* : |w| = n, |w|_1 \text{ is even}\}.$$

There is a DFA of size $2n + 1$ that accepts L_n . We show

Theorem 23 *The minimal regular expression for L_n is of size $\Omega(n^2)$.*

Proof. To prove this lower bound we show how to transform a regular expression for L_n into a boolean formula for L_n of the same size. Our lower bound then follows from Khrapchenko's $\Omega(n^2)$ lower bound for the boolean formula size of L_n [28, 54].

The transformation we define below can be applied to any regular expression that specifies a binary language where all words in the language have the same length.

The transformation works as follows. Let R be a regular expression for a language L where all words in L have length n . Note that since words in L have the same length, each alphabetic symbol in R matches exactly one position of a word in L . We will use this fact below. We go through the regular expression R and do the following:

1. If we encounter an alphabetic symbol a and a matches position i , then we replace a by x_i if $a = 1$ and by $\neg x_i$ if $a = 0$.
2. If we encounter $+$ (union), then we replace $+$ by \vee (OR).
3. If we encounter \cdot (concatenation), then we replace \cdot by \wedge (AND).
4. We leave the parenthesis unchanged.

Let the resulting boolean formula be $B = \chi(R)$. We claim that

Lemma 24 (a_1, a_2, \dots, a_n) , $a_i \in \{0, 1\}$ is a satisfying assignment for B if and only if $a_1 a_2 \dots a_n \in L$.

Proof. We proceed by induction on the alphabetic length of R .

Base case: $|R| = 1$. In this case R contains exactly one alphabetic symbol. The lemma follows immediately from the definition of B .

Inductive case:

1. $R = R' + R''$. Let $B' = \chi(R')$ and $B'' = \chi(R'')$. From the definition, we have

$$B = \chi(R) = \chi(R') \vee \chi(R'') = B' \vee B''.$$

By definition, (a_1, \dots, a_n) is a satisfying assignment for B if and only if it is a satisfying assignment for B' or B'' . By induction, (a_1, \dots, a_n) is a satisfying assignment for B' or B'' if and only if $a_1 \dots a_n \in L(R') \cup L(R'')$. So the lemma is true in this case.

2. $R = R'R''$. Let $B' = \chi(R')$ and $B'' = \chi(R'')$. From the definition, we have

$$B = \chi(R) = \chi(R') \wedge \chi(R'') = B' \wedge B''.$$

By definition, (a_1, \dots, a_n) is a satisfying assignment for B if and only if it is a satisfying assignment for B' and B'' . By construction, there exists i such that B' is a boolean formula on the variables (x_1, \dots, x_i) and B'' is a boolean formula on the variables (x_{i+1}, \dots, x_n) . So by induction (a_1, \dots, a_i) is a satisfying assignment for B' if and only if $a_1 \dots a_i \in L(R')$ and (a_{i+1}, \dots, a_n) is a satisfying assignment for B'' if and only if $a_{i+1} \dots a_n \in L(R'')$. Hence (a_1, \dots, a_n) is a satisfying assignment for B if and only if $a_1 \dots a_n \in L$. So this case is true and we are done. \square

To complete the proof of Theorem 23, note that the standard definition of size of $B = \chi(R)$ equals the alphabetic size of R . It follows that $|R| = |B| = \Omega(n^2)$. \square

One can also ask similar questions for converting from a DFA or NFA to a GRE, but no nontrivial lower bounds appear to be known here.

We now turn to another question about GRE's. What is maximum possible size blow-up in going from a GRE to RE? (As mentioned before, to avoid trivialities, we must use the reverse polish length or ordinary length when measuring the size of the GRE.)

Surprisingly, the blow-up is not even elementary. (By "elementary", we mean a function of the form

$$2^{2^{\dots 2^n}}$$

where the number of 2's is bounded.) This seems to be a "folklore" result apparently first published by Dang [14]. (For a related result, see [48].)

Theorem 25 *The worst-case size blow-up from GRE to RE is not elementary.*

Here is an argument, suggested to us by Albert Meyer (personal communication):

Proof. Suppose every GRE R of size n has an equivalent RE R' having $\leq f(n)$ alphabetic symbols, where $f(n)$ is an elementary function of the form $2^{2^{\dots 2^n}}$ for some finite number of levels of exponents.

Now by Theorem 10, R' can be converted to an equivalent DFA having at most $\leq 2^{f(n)+1}$ states. But, as is well-known, a DFA M of t states accepts the empty language if and only if it accepts no string of length $< t$ [24, pp. 63–64]. Hence we would have the following algorithm A for determining if $L(R) = \emptyset$: for each string x of length $\leq 2^{f(n)+1}$, determine if $x \in L(R)$. If the answer is no for all these x , return "yes"; otherwise, return "no". Since we can decide if $x \in L(R)$ in time polynomial in $x + |R|$ ([24, pp. 75–76]; [23, 31]) the running time of algorithm A is elementary. But this contradicts a well-known result of Stockmeyer that there is no algorithm that runs in elementary time which decides if the language specified by a GRE is empty. ([2, p. 422], [50]). \square

For the unary case, however, the situation is somewhat different. Let $sc(L)$ denote the deterministic state complexity of a regular language L , i.e., the number of states in a minimal DFA for L . Yu, Zhuang, & Salomaa [52] proved that, for unary languages L_1 and L_2 we have

$$\begin{aligned} sc(L_1 L_2) &\leq sc(L_1) sc(L_2) \\ sc(L_1 \cup L_2) &\leq sc(L_1) sc(L_2) \\ sc(L_1 \cap L_2) &\leq sc(L_1) sc(L_2) \\ sc(\overline{L_1}) &= sc(L_1) \\ sc(L_1^*) &\leq (sc(L_1) - 1)^2 + 1. \end{aligned}$$

(For further results along these lines, see [45].) From this, a result of Dang [15] easily follows:

Theorem 26 For an generalized regular expression over a unary alphabet $\{a\}$, define its “refined length” as follows:

$$\begin{aligned} \text{rlen}(r_1 r_2) &= \text{rlen}(r_1) + \text{rlen}(r_2) \\ \text{rlen}(r_1 + r_2) &= \text{rlen}(r_1) + \text{rlen}(r_2) \\ \text{rlen}(r_1 \cap r_2) &= \text{rlen}(r_1) + \text{rlen}(r_2) \\ \text{rlen}(\neg r_1) &= \max(\text{rlen}(r_1), 1) \\ \text{rlen}(r_1^*) &= 2\text{rlen}(r_1) \\ \text{rlen}(a) &= 1 \\ \text{rlen}(\epsilon) &= \text{rlen}(\emptyset) = 0. \end{aligned}$$

(Note this definition of length is non-standard.) Then $\text{sc}(L(r)) \leq 3^{\text{rlen}(r)}$.

4. Operations on Regular Languages

In this section we examine how various operations involving regular languages affect the length of the regular expression representing them.

In particular, what is maximum blow-up in going from an RE for L to the shortest possible RE for $\bar{L} = \Sigma^* - L$? For the unary case optimal results are known.

Theorem 27 If E is a regular expression with $|\text{alph}(E)| = n$ specifying a unary regular language L , then there exists a regular expression E' of length $e^{O(\sqrt{n \log n})}$ specifying \bar{L} . Furthermore, there exist infinitely many regular languages for which this bound is achieved.

Proof. Convert the RE to an NFA using the usual method; convert the NFA to a DFA using the subset construction, interchange accepting and non-accepting states, and convert the resulting DFA to an RE. The first, third, and fourth steps involve only a linear blow-up, while the second can increase the size of the resulting DFA by $e^{\sqrt{n \log n}}$, as is well-known (e.g., [12, 39, 38]). This gives a bound of $e^{O(\sqrt{n \log n})}$.

This upper bound can be achieved using an expression like

$$\epsilon + r_1(00)^* + r_2(000)^* + r_4(00000)^* + \cdots + r_{p-1}(0^p)^*$$

where r_n is the regular expression defined above in Section 2, and p is the largest prime $\leq n$. Using a result from [5], this regular expression is of length $t = O(n^2/(\log n))$. However, the regular expression for the complement is of length $e^{O(\sqrt{t \log t})}$, since the shortest string in the complement is $0^{p_1 p_2 \cdots p_k}$, where $p_1, p_2, \dots, p_k = p$ are the primes $\leq n$, and by the prime number theorem we have $p_1 p_2 \cdots p_k = e^{n(1+o(1))}$. \square

How about the case of larger alphabets? For the upper bound, the best result we currently know is doubly-exponential. First convert the RE to a DFA, interchange accepting and non-accepting states; and, finally, convert back to an RE. This gives an upper bound of the form c^{2^n} for a constant c .

For a lower bound, we have the following examples.

Theorem 28 Define $E_n = (0 + 1)^*0(0 + 1)^{n-1}0(0 + 1)^*$. Then

- (a) E_n is a regular expression with $|\text{alph}(E_n)| = 2n + 4$;
- (b) $L(E_n)$ is accepted by a minimal DFA with $2^n + 1$ states;
- (c) $L(E_n)$ is accepted by an NFA with $n + 2$ states;
- (d) $\overline{L(E_n)}$ is not accepted by any NFA with $< 2^n$ states;
- (e) $\overline{L(E_n)}$ is not specified by any regular expression E with $|\text{alph}(E)| < 2^n - 1$.

Proof. (a) Clear.

(b) follows because we can accept $L(E_n)$ with $2^n + 1$ states by using a state labeled with every string of length n , indicating the last n symbols seen, plus one more state for the accepting state once we detect $0(0 + 1)^{n-1}0$. The initial state is $[111 \cdots 1]$.

To see this is minimal, we use the Myhill-Nerode theorem. Note that for the states labeled with strings, say w and w' , we must have w and w' differ in some letter. Without loss of generality, assume w has a 0 in position i but w' does not. Now append enough 1's and then a 0 to make the 0 in position i occur n symbols from the added 0. Now w will be accepted and w' won't be. To see the additional accepting state is distinguishable from all the labeled states, note that ϵ distinguishes them.

(c) Easy. Use a state that loops on everything, followed by a transition on 0, followed by n transitions on everything, followed by a transition on 0, followed by a state that loops on everything.

(d) Use Birget's theorem (Theorem 7). If $w \in \{0, 1\}^*$, let \tilde{w} be the string obtained from w by changing every 0 to a 1 and vice-versa. In Birget's theorem let S be the set of pairs

$$\{(w, \tilde{w}) : w \in \{0, 1\}^n\}.$$

Clearly $w\tilde{w} \in \overline{L(E_n)}$, since any two symbols that are n positions apart are different. Now if $w \neq x$ then (say) there is a 0 in a position in w that has a 1 in the corresponding position of x . Then $w\tilde{x} \in L(E_n)$ since there are two 0's n positions apart.

(e) If $\overline{L(E_n)}$ were specified by a regular expression with $2^n - 1$ symbols there would be an NFA for $\overline{L(E_n)}$ with 2^n states by Theorem 10, a contradiction. \square

This leads to

Open Problem 4 What is the maximum achievable blow-up in going from a regular expression for L to a shortest regular expression for \overline{L} ?

Here is an interesting example over a class of growing alphabets. Define $\Sigma_n := \{1, 2, \dots, n\}$ and define $P_n := \{w \in \Sigma^* : \text{for all } i, 1 \leq i \leq n, i \text{ occurs exactly once in } w\}$. In other words, P_n is the language of all strings representing permutations of $\{1, 2, \dots, n\}$. For example, $P_3 = \{123, 132, 213, 231, 312, 321\}$.

Theorem 29 No NFA with $< 2^n$ states can accept P_n . No regular expression with $< 2^{n-1}$ symbols can specify P_n . However, there is a regular expression of length $O(n^2)$ that can specify $\overline{P_n}$.

Proof. To see this, use Birget's theorem (Theorem 7). Let the set of pairs be $\{(w, w') : w \text{ is a word formed by concatenating the symbols of any subset } S \text{ of } \Sigma_n, \text{ and } w' \text{ is a word formed by concatenating the symbols of } \Sigma_n - S\}$. For example, for $n = 3$ we can take $\{(\epsilon, 123), (1, 23), (2, 13), (3, 12), (12, 3), (13, 2), (23, 1), (123, \epsilon)\}$. Then by the theorem in that paper any NFA accepting P_n must have as many states as there are pairs, so at least 2^n .

Now we claim no regular expression with $< 2^{n-1}$ symbols can generate P_n . For if so, by the usual method of converting a regular expression to an NFA there would be an NFA with $< 2^n$ symbols accepting P_n , a contradiction.

However, there is a regular expression for $\overline{P_n}$ with $O(n^2)$ symbols. We just give it by example for $n = 4$:

$$\begin{aligned} & (1 + 2 + 3)^* + (1 + 2 + 4)^* + (1 + 3 + 4)^* + (2 + 3 + 4)^* \\ & + (1 + 2 + 3 + 4)^*(1(1 + 2 + 3 + 4)^*1 + 2(1 + 2 + 3 + 4)^*2 \\ & + 3(1 + 2 + 3 + 4)^*3 + 4(1 + 2 + 3 + 4)^*4)(1 + 2 + 3 + 4)^* \end{aligned}$$

□

This example is particularly interesting because we can prove a similar result for context-free representations of P_n .

Theorem 30 *Let $G = (V, \Sigma, P, S)$ be a grammar in Chomsky normal form generating P_n . Then $|V| = \Omega(n^{-3/2}d^n)$, where $d = 3/2^{2/3} \doteq 1.89$.*

First, we need the following lemma:

Lemma 31 *If $S \Longrightarrow^* w$ is a derivation of a word in some CNF grammar G , and $|w| > 1$, then there exists a variable A participating in the derivation,*

$$S \Longrightarrow^* \alpha A \beta \Longrightarrow^* w$$

such that if y represents the yield of A , then $|w|/3 \leq |y| < 2|w|/3$.

Proof. Consider the derivation tree T corresponding to the derivation. The yield of S is of length $|w|$. Now trace a path from S down to a leaf, at each point choosing the variable with the larger yield. Eventually we reach a variable with yield 1; since $|w| > 1$ the yield y of this variable satisfies $|y| < 2|w|/3$. Thus there must be a variable, call it A , for which its yield y satisfies $|y| < 2|w|/3$, but for which its parent variable B has yield z satisfying $|z| \geq 2|w|/3$. We claim A is the desired variable. For since we always chose the variable with the larger yield, we must have $|y| \geq |w|/3$. □

Now we can prove Theorem 30.

Proof. Now let P_n be the language of all permutations of $\{1, 2, \dots, n\}$, $n > 1$, and let G be a Chomsky normal form grammar generating P_n . Without loss of generality assume all variables are useful (participate in some derivation of some word in P_n).

First we note that if C is a variable of G , then

- (i) all strings generated by C are of the same length

- (ii) every string generated by C is a permutation of all the other strings generated by C .

For if (i) were not true then G would generate a string of length other than n , and if (ii) were not true then G would generate some string that is not a permutation of $\{1, 2, \dots, a_n\}$.

Choose any word $w \in P_n$. We show how to associate with w a pair (A, k) where A is a variable in G and k is an integer from 0 to $n - 1$ representing a position in w . From the Lemma there exists a variable A such that A generates a subword y of w with $|w|/3 \leq |y| < 2|w|/3$. (There may be as many as 3 such; just pick one.) This subword y occurs at a uniquely defined position k within w . Associate w with the pair (A, k) .

Now consider all the pairs (A, k) so assigned. How many words can be associated with a fixed pair (A, k) ? Now A generates words of a fixed length r , and by the argument above $n/3 \leq r < 2n/3$. So there are $r!$ different possibilities for the words generated by A . Furthermore, there are $n - r$ remaining letters once the word generated by A is removed. So there are at most $r!(n - r)!$ possible words associated with (A, k) . But there are $n!$ words in all, so there are at least $n!/(r!(n - r)!) = \binom{n}{r}$ distinct pairs (A, k) . But k can take n different values, so the grammar must contain at least $(n - 1)!/(r!(n - r)!)$ different variables A . Since the binomial coefficients $\binom{n}{k}$ increase monotonically from $k = 0$ to $k = \lfloor \frac{n}{2} \rfloor$, this bound is minimized in the range $n/3 \leq r < 2n/3$ at the extreme points, which gives us the bound $\frac{1}{n} \binom{n}{\lfloor n/3 \rfloor}$. Asymptotically, this bound is equal to $\Omega(n^{-3/2}d^n)$, where $d = 3/2^{2/3}$, using Stirling's formula. \square

We now turn to intersection. Given RE's E_1, E_2 , with m and n alphabetic symbols, respectively, what is the shortest regular expression for $L(E_1) \cap L(E_2)$? By converting to an NFA, forming the direct product, and converting back to an RE, we get an upper bound of the form $c^{(m+1)(n+1)}$. However, we do not currently know an example where the blow-up exceeds cmn .

Open Problem 5 What is the maximum achievable blow-up in going from regular expressions for L_1, L_2 to a shortest regular expression for $L_1 \cap L_2$?

In the unary case we can get an upper bound of $O((mn)^2)$ using Chrobak normal form. Can this be achieved?

There are some interesting variations on these problems. Zhivko Nedev of Wilfrid Laurier University (personal communication, June 2001), asked, what is the worst-case in going from a regular expression R to a regular expression for $L(R) \cap \Sigma^n$? From Theorem 20, we know that we can find a regular expression E for $L(R) \cap \Sigma^n$ with $|\text{alph}(E)| \leq k(r + 1)(r + 2)n^{\log_2(r+1)+1}$, where $k = |\Sigma|$ and $r = |\text{alph}(R)|$. A similar bound holds for $L(R) \cap \Sigma^{\leq n}$.

5. Shortest String not Specified by a Regular Expression

Suppose we have a regular expression E with $|\text{alph}(E)| = n$ over a finite alphabet Σ with $L(E) \neq \Sigma^*$. How long can the shortest string *not* specified by E be?

An upper bound of 2^n can be obtained as follows: first, convert E to a DFA of at most $2^n + 1$ states. Next, interchange accepting and non-accepting states, and look for the shortest string accepted. This is of length at most 2^n . Can this bound be achieved?

We can get 2^{cn} as follows: we create a regular expression that represents all strings except the numbers $1, 2, \dots, 2^n - 1$ (suitably encoded) listed in increasing order, separated by a special delimiter symbol $\#$. To verify that a string is not of this form, we only need verify either that it is syntactically incorrect, or that it contains a substring of the form $\#x\#w\#$ where w does not denote a binary number that is 1 more than that denoted by x . (For a similar construction, see Kumar [30].)

To make life even easier, we will assume that the representations for n also include those for $n - 1$. We do this by treating the base-2 representations of both numbers in parallel, writing one expansion above the other and using the alphabet $\{0, 1\} \times \{0, 1\}$. Thus for $n = 3$ the only string not specified will be

$$\begin{array}{cccccccc} \# & 000 & 001 & 010 & 011 & 100 & 101 & 110 \\ \# & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{array} \#.$$

Note that the strings in the top row encode the integers from 0 to $2^3 - 2$, inclusive, while the strings in the bottom row encode the integers from 1 to $2^3 - 1$, inclusive.

We now recode this over the alphabet $\{a, b, c, d, e\}$, as follows:

$$\begin{array}{ll} 0 & \rightarrow a; & 0 & \rightarrow b; \\ 1 & \rightarrow c; & 1 & \rightarrow d; \\ \# & \rightarrow e. \end{array}$$

Thus, for $n = 3$ the only string not specified will be

$$eaabeabceadbcbcedabedbcddbe.$$

Now we construct the necessary regular expressions. We use the abbreviation $\Sigma = \{a, b, c, d, e\}$.

1. E_1 : the first $n + 2$ symbols are not $ea^{n-1}be$. For $n = 4$ this is

$$\begin{aligned} & \epsilon + e(\epsilon + a(\epsilon + a(\epsilon + a(\epsilon + b)))) \\ & + ((\epsilon + eaaab)(a + b + c + d) \\ & + e(\epsilon + a(\epsilon + a))(b + c + d + e)eaaa(a + c + d + e))\Sigma^*. \end{aligned}$$

We have $|\text{alph}(E_1)| = 4n + 18$.

2. E_2 : the last $n + 2$ symbols are not $ed^{n-1}be$. For $n = 4$ this is

$$\begin{aligned} &\epsilon + (\epsilon + (\epsilon + (\epsilon + (\epsilon + d)d)d)b)e \\ &+ \Sigma^*((a + b + c + d)(\epsilon + dddbe) \\ &+ (a + c + d + e)e + (a + b + c + e)(\epsilon + (\epsilon + d)d)be). \end{aligned}$$

We have $|\text{alph}(E_2)| = 3n + 20$.

3. E_3 : the string has two consecutive e 's: $\Sigma^*ee\Sigma^*$. We have $|\text{alph}(E_3)| = 12$.
 4. E_4 : the string has a block of more than n consecutive digits: $\Sigma^*(a + b + c + d)^{n+1}\Sigma^*$. We have $|\text{alph}(E_4)| = 4n + 14$.
 5. E_5 : the string has a block of length $< n$ between two e 's: $\Sigma^*e(a + b + c + d + \epsilon)^{n-1}e\Sigma^*$. We have $|\text{alph}(E_5)| = 4n + 8$.
 6. E_6 : the string has a block of the form $e_y^x e$ where $[y]_2 \neq [x]_2 + 1$:

$$\Sigma^*e(a + d) * (c + e + bc^*(a + b + d))\Sigma^*.$$

We have $|\text{alph}(E_6)| = 20$.

7. E_7 : the string has a block of the form $e_y^x e_{z'}^{y'}$ where $y \neq y'$:

$$\Sigma^*((a + c)\Sigma^n(c + d) + (b + d)\Sigma^n(a + b))\Sigma^*.$$

We have $|\text{alph}(E_7)| = 10n + 18$.

Our regular expression $E_1 + E_2 + E_3 + E_4 + E_5 + E_6 + E_7$ proves

Theorem 32 *For each $n \geq 3$ there is a regular expression E over the alphabet $\{a, b, c, d, e\}$ with $|\text{alph}(E)| = 25n + 110$ such that the shortest string not specified is of length $(2^n - 1)(n + 1) + 1$.*

We can convert this example to an example over the alphabet $\{0, 1\}$, as follows. First, we apply the morphism sending $a \rightarrow 000$, $b \rightarrow 001$, $c \rightarrow 010$, $d \rightarrow 011$, and $e \rightarrow 111$ to each of the regular expressions E_1, E_2, \dots, E_7 . Next, we supplement the resulting regular expression with two additional ones:

8. E_8 : strings of length divisible by three, containing one of the three excluded triples:

$$((0 + 1)(0 + 1)(0 + 1))^*(100 + 101 + 110)((0 + 1)(0 + 1)(0 + 1))^*.$$

We have $|\text{alph}(E_8)| = 21$.

9. E_9 : strings of length not divisible by three: $((0 + 1)(0 + 1)(0 + 1))^*(0 + 1)(0 + 1 + \epsilon)$. We have $|\text{alph}(E_9)| = 10$.

This gives

Theorem 33 *For each $n \geq 3$ there is a regular expression E over the alphabet $\{0, 1\}$ with $|\text{alph}(E)| = 75n + 361$ such that the shortest string not specified is of length $3(2^n - 1)(n + 1) + 3$.*

Open Problem 6 For a regular expression E over $\{0, 1\}$, define

$$\text{lssns}(E) = \begin{cases} \min\{|x| : x \in \Sigma^* - L(E)\}, & \text{if } L(E) \neq \Sigma^* ; \\ 0, & \text{if } L(E) = \Sigma^*. \end{cases}$$

Define $h(n) = \max_{|\text{alph}(E)|=n} \text{lssns}(E)$. By Theorem 33 we know $h(n) > 2^{n/75}$ for all n sufficiently large, and as above $h(n) \leq 2^n$. Find a closed form for $h(n)$ or better upper and lower bounds.

6. Acknowledgements

We thank Ernst Leiss and Albert Meyer for their helpful discussions regarding generalized regular expressions, and Detlef Wotschke for informing us about the paper of Book and Chandra. We thank Troy Vasiga for a careful reading of a draft. We also thank the referees for many helpful suggestions.

References

- [1] L. ACETO, W. FOKKINK, A. INGÓLFSÐÓTTIR, On a question of A. Salomaa: the equational theory of regular expressions over a singleton alphabet is not finitely based. *Theoret. Comput. Sci.* **209** (1998), 163–178.
- [2] A. V. AHO, J. E. HOPCROFT, J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [3] N. ALON, P. SEYMOUR, R. THOMAS, A separator theorem for nonplanar graphs. *J. Amer. Math. Soc.* **3** (1990), 801–808.
- [4] N. ALON, P. SEYMOUR, R. THOMAS, Planar separators. *SIAM J. Disc. Math.* **7** (1994), 184–193.
- [5] E. BACH, J. SHALLIT, *Algorithmic Number Theory*. The MIT Press, 1986.
- [6] J. BERSTEL, Fibonacci words—a survey. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pp. 13–27. Springer-Verlag, 1986.
- [7] J.-C. BIRGET, Intersection and union of regular languages and state complexity. *Inform. Process. Lett.* **43** (1992), 185–190.
- [8] R. V. BOOK, A. K. CHANDRA, Inherently nonplanar automata. *Acta Informatica* **6** (1976), 89–94.
- [9] A. BRAUER, On a problem of partitions. *Amer. J. Math.* **64** (1942), 299–312.
- [10] J. BRZOZOWSKI, A survey of regular expressions and their applications. *IEEE Trans. Electr. Comput.* **11** (1962), 324–335.
- [11] A. W. BURKS, H. WANG, The logic of automata—Part II. *J. Assoc. Comput. Mach.* **4** (1957), 279–297.
- [12] M. CHROBAK, Finite automata and unary languages. *Theoret. Comput. Sci.* **47** (1986), 149–158. Errata, **302** (2003), 497–498.

- [13] J. H. CONWAY, *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.
- [14] Z. R. DANG, On the complexity of a finite automaton corresponding to a generalized regular expression. *Dokl. Akad. Nauk SSSR* **213** (1973), 26–29. In Russian. English translation in *Soviet Math. Dokl.* **14** (1973), 1632–1636.
- [15] Z. R. DANG, Upper bounds on finite-automaton complexity for generalized regular expressions in a 1-letter alphabet. *Diskretnaya Matematika* **1**(4) (1989), 12–16. In Russian.
- [16] H. N. DJIDJEV, On the problem of partitioning planar graphs. *SIAM J. Algebraic Discrete Methods* **3** (1982), 229–240.
- [17] M. DOMARATZKI, D. KISMAN, J. SHALLIT, On the number of distinct languages accepted by finite automata with n states. *J. Autom. Lang. Comb.* **7** (2002), 469–486.
- [18] A. EHRENFUCHT, P. ZEIGER, Complexity measures for regular expressions. *J. Comput. System Sci.* **12** (1976), 134–146.
- [19] U. FEIGE, R. KRAUTHGAMER, A polylogarithmic approximation of the minimum bisection. In *Proc. 41st Symp. Found. Comput. Sci.*, pp. 105–115. IEEE Press, 2000.
- [20] J. R. GILBERT, J. P. HUTCHINSON, R. E. TARJAN, A separator theorem for graphs of bounded genus. *J. Algorithms* **5** (1984), 391–407.
- [21] I. GLAISER, J. SHALLIT, A lower bound technique for the size of nondeterministic finite automata. *Inform. Process. Lett.* **59** (1996), 75–77.
- [22] V. M. GLUSHKOV, The abstract theory of automata. *Uspekhi. Mat. Nauk* **16**(5) (1961), 3–62. In Russian. English translation in *Russian Math. Surveys* **16** (5) (1961), 1–53.
- [23] S. C. HIRST, A new algorithm solving membership of extended regular expressions. Technical Report 354, Department of Computer Science, University of Sydney, 1989.
- [24] J. E. HOPCROFT, J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [25] L. ILIE, S. YU, Algorithms for computing small NFAs. To appear, Proc. 27th MFCS, 2002.
- [26] T. JIANG, B. RAVIKUMAR, Minimal NFA problems are hard. *SIAM J. Comput.* **22** (1993), 1117–1141.
- [27] T. KAMEDA, P. WEINER, On the state minimization of nondeterministic finite automata. *IEEE Trans. Comput.* **C-19** (1970), 617–627.
- [28] V. M. KHRAPCHENKO, Methods for determining lower bounds for the complexity of π -schemes. *Mat. Zametki* **10** (1972), 83–92. In Russian. English translation in *Math. Notes Acad. Sciences USSR* **10** (1972), 474–479.

- [29] S. C. KLEENE, Representation of events in nerve nets and finite automata. In *Automata Studies*, pp. 3–42. Princeton University Press, 1956.
- [30] K. N. KUMAR, Solution to puzzle 2. *IARCS [Indian Association for Research in Computer Science] Newsletter* **2**(1) (March 1997), 17–18, <http://www.imsc.ernet.in/~iarcs/vol2-1/old-puzzles.ps>
- [31] O. KUPFERMAN, S. ZUHOVITZKY, An improved algorithm for the membership problem for extended regular expressions. In: K. DIKS, W. RYTTER (eds.), *Proc. Math. Found. Comput. Sci.* LNCS 2420, Springer, 2002, pp. 446–458.
- [32] E. LEISS, The complexity of restricted regular expressions. In: *Proc. 1980 Conference on Information Sciences and Systems*. 1980, 204–206.
- [33] E. LEISS, Constructing a finite automaton for a given regular expression. *SIGACT News* **12**(3) (Fall 1980), 81–87.
- [34] E. LEISS, The complexity of restricted regular expressions and the synthesis problem for finite automata. *J. Comput. System Sci.* **23** (1981), 348–354.
- [35] H. LEUNG, Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. *SIAM J. Comput.* **27** (1998), 1073–1082.
- [36] R. J. LIPTON, R. E. TARJAN, A separator theorem for planar graphs. *SIAM J. Appl. Math.* **36** (1979), 177–189.
- [37] R. J. LIPTON, R. E. TARJAN, Applications of a planar separator theorem. *SIAM J. Comput.* **9** (1980), 615–627.
- [38] JU. I. LYUBICH, Estimates for optimal determinization of nondeterministic autonomous automata. *Sibirskii Matematicheskii Zhurnal* **5** (1964), 337–355. In Russian.
- [39] R. MANDL, Precise bounds associated with the subset construction on various classes of nondeterministic finite automata. In: *Proc. 7th Princeton Conference on Information and System Sciences*. 1973, 263–267.
- [40] A. MARTINEZ, Efficient computation of regular expressions from unary NFAs. In *Pre-Proceedings, Descriptive Complexity of Formal Systems (DCFS)*, pp. 174–187. Department of Computer Science, University of Western Ontario, 2002. Technical Report No. 586.
- [41] R. MCNAUGHTON, H. YAMADA, Regular expressions and state graphs for automata. *IRE Trans. Electron. Comput.* **EC-9** (1960), 39–47.
- [42] A. R. MEYER, M. J. FISCHER, Economy of description by automata, grammars, and formal systems. In *Proc. 12th Annual Symposium on Switching and Automata Theory*, IEEE, 1971, pp. 188–191.
- [43] W. MILLER, The maximum order of an element of a finite symmetric group. *Amer. Math. Monthly* **94** (1987), 497–506.
- [44] B. G. MIRKIN, An algorithm for constructing a base in a language of regular expressions. *Engineer. Cybernet.* **5** (1966), 110–116.

- [45] G. PIGHIZZINI, J. SHALLIT, Unary language operations, state complexity, and Jacobsthal's function. *Internat. J. Found. Comp. Sci.* **13** (2002), 145–159.
- [46] D. A. PLAISTED, A heuristic algorithm for small separators in arbitrary graphs. *SIAM J. Comput.* **19** (1990), 267–280.
- [47] C. POMERANCE, J. M. ROBSON, J. SHALLIT, Automaticity II: Descriptive complexity in the unary case. *Theoret. Comput. Sci.* **180** (1997), 181–201.
- [48] J. L. RANGEL, The equivalence problem for regular expressions over one letter is elementary. In *Proc. 15th Ann. Symp. on Switching and Automata Theory*, pp. 24–27. IEEE Press, 1974.
- [49] D. RAYMOND, D. WOOD, *Grail*: a C++ library for automata and expressions. *J. Symbolic Comput.* **17** (1994), 341–350.
- [50] L. J. STOCKMEYER, The complexity of decision problems in automata theory and logic. PhD thesis, MIT, July 1974.
- [51] V. WAIZENEGGER, Über die Effizienz der Darstellung durch reguläre Ausdrücke und endliche Automaten. Diplomarbeit, Fach Informatik, Technische Hochschule Aachen, Germany, 2000.
- [52] S. YU, Q. ZHUANG, K. SALOMAA, The state complexity of some basic operations on regular languages. *Theoret. Comput. Sci.* **125** (1994), 315–328.
- [53] D. ZIADI, Regular expression for a language without empty word. *Theoret. Comput. Sci.* **163** (1996), 309–315.
- [54] U. ZWICK, An extension of Khrapchenko's theorem. *Info. Proc. Letters* **37** (1991), 215–217.

(Received: January 24, 2003; revised: June 17, 2004)