Optimal Lower Bounds on Regular Expression Size Using Communication Complexity

Hermann Gruber and Jan Johannsen

Institut für Informatik, LMU München
Oettingenstr. 67, 80538 München, Germany
{gruberh,jjohanns}@tcs.ifi.lmu.de

Abstract. The problem of converting deterministic finite automata into (short) regular expressions is considered. It is known that the required expression size is $2^{\Theta(n)}$ in the worst case for infinite languages, and for finite languages it is $n^{\Omega(\log \log n)}$ and $n^{O(\log n)}$, if the alphabet size grows with the number of states n of the given automaton. A new lower bound method based on communication complexity for regular expression size is developed to show that the required size is indeed $n^{\Theta(\log n)}$.

For constant alphabet size the best lower bound known to date is $\Omega(n^2)$, even when allowing infinite languages and nondeterministic finite automata. As the technique developed here works equally well for deterministic finite automata over binary alphabets, the lower bound is improved to $n^{\Omega(\log n)}$.

1 Introduction

One of the most basic theorems in formal language theory is that every finite automaton can be converted into an equivalent regular expression, and vice versa [10]. While algorithms accomplishing these tasks have been known for a long time, there has been a renewed interest in these classical problems during the last few years. For instance, new algorithms for converting regular expressions into finite automata outperforming classical algorithms have been found only recently, as well as a matching lower bound of $\Omega(n \cdot (\log n)^2)$ on the minimum number of transitions required by any equivalent nondeterministic finite automaton (NFA). The lower bound is, however, only reachable for growing alphabets, and a better algorithm is known for constant alphabet size, see [20] for the current state of the art.

In contrast, much less is known about the converse direction, namely of converting finite automata into regular expressions. Apart from the fundamental nature of the problem, some applications of converting finite automata into regular expressions lie in control flow normalization, including uses in software engineering such as automatic translation of legacy code [17]. All known algorithms covering the general case of infinite languages are based on the classical ones, which are compared in the survey [18]. The drawback is that all of these (structurally similar) algorithms return expressions of size $2^{O(n)}$ in the worst case, and Ehrenfeucht and Zeiger exhibit a family of languages over a growing alphabet for which

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 273–286, 2008.

this exponential blow-up is inevitable [2]. This leads to the quest for identifying structural restrictions on the underlying transition graph of the given finite automaton that can guarantee a shorter equivalent regular expression [3,16], as well as for heuristics improving the classical algorithms [6,1]. Another possibility is to concentrate on subfamilies of regular languages. For the important special case of unary languages, it has been established that every n-state nondeterministic finite automaton can be converted in polynomial time into an equivalent regular expression of polynomial size [13]. And for finite languages, there exist equivalent regular expressions of size at most $n^{O(\log n)}$ obtained by a classical construction, which is carefully analyzed in [3]. In contrast, results from [2] show that size $n^{\Omega(\log\log n)}$ can be necessary for finite languages—at least for a growing alphabet.

Although there remains a considerable gap between the best known upper bounds and the lower bounds given some 30 years ago, to the best of our knowledge, only little progress has been made on this problem. The most preeminent gap between the upper and lower bounds presented in [2] is in the case of finite languages. There the upper and lower bounds of $n^{O(\log n)}$ and $n^{\Omega(\log\log n)}$, respectively, are essentially the best ones known to date. We close this gap, giving that the blow-up for finite languages is $n^{\Theta(\log n)}$ in the worst case, when switching the representation from a finite automaton to a regular expression. To this end, we develop a new lower bound technique for regular expression size based on communication complexity. Ellul et al. [3] prove a lower bound of $\Omega(n^2)$ on the size of regular expressions for a finite language over the binary alphabet by a reduction to Boolean circuit complexity. We improve this approach by harnessing a technique used to obtain better lower bounds for *monotone* Boolean circuits, using the communication complexity of search problems as introduced by Karchmer and Wigderson [8]. Our approach shows that the lower bound can even be realized for an n-state *deterministic* finite automaton over a *binary* alphabet.

We also show that a family of finite languages (over a growing alphabet) studied by Ehrenfeucht and Zeiger [2] captures the combinatorial core of the conversion problem for finite languages, as these form in some precise sense the hardest languages for this problem. We then use this to obtain a slight improvement of the best known upper bounds on this conversion problem.

2 Preliminaries

2.1 Formal Languages

We assume the reader to be familiar with the basic notions in formal language and automata theory as contained in [7]. In particular, let Σ be an alphabet and Σ^* the set of all words over the alphabet Σ, including the empty word ε. The length of a word w is denoted by $|w|$, where $|\varepsilon| = 0$, and the total number of occurrences of the alphabet symbol a in w is denoted by $|w|_a$. In this paper we mainly deal with a special class of finite languages called homogeneous languages. A finite language $L \subset \Sigma^*$ is *homogeneous* if all words in the language have the same length. In order to fix the notation, we briefly recall the definition of regular expressions and the languages described by them:

Let Σ be an alphabet. The *regular expressions* over Σ and the languages that they denote are defined recursively as follows: $\emptyset$ is a regular expression and denotes the empty language; for $a \in \Sigma \cup \{\varepsilon\}$, a is a regular expression and denotes the language $\{a\}$; if e and f are regular expressions denoting languages E and F, then $(e + f)$, $(e \cdot f)$ and $(e)^*$ are regular expressions denoting the languages $E \cup F$, $E \cdot F$ and E^*, respectively. Finally, the language described by the regular expression E is denoted by $L(E)$.

For convenience, parentheses are sometimes omitted and the product is simply written as juxtaposition. The priority of operators is specified in the usual fashion: product is performed before disjunction, and star before both product and disjunction. We also write sometimes $L_1 + L_2$ to denote the union of the languages L_1 and L_2. The *alphabetic width* (or *size*) $\mathrm{alph}(E)$ of a regular expression E is defined as the total number of occurrences of symbols in Σ in E. For a regular language L we define $\mathrm{alph}(L)$ as the minimum alphabetic width among all regular expressions describing L. As we will be primarily concerned with small regular expressions, we recall the notion of uncollapsible regular expressions [3]:

Let E be a regular expression. We say that E is *uncollapsible* if all of the following conditions hold: If E contains the symbol $\emptyset$, then $E = \emptyset$; the expression E contains no subexpression of the form FG or GF, with $L(F) = \{\varepsilon\}$; if E contains a subexpression of the form $F + G$ or $G + F$ with $L(F) = \{\varepsilon\}$, then $\varepsilon \notin L(G)$; if E contains a subexpression of the form F^*, then $L(F) \neq \{\varepsilon\}$.

The reader might have noticed that we have added a fourth condition not present in the original definition. This condition ensures that the star operator cannot occur in uncollapsible regular expressions describing finite languages. It is easily seen that for every collapsible regular expression, there is an uncollapsible one specifying the same language of at most the same size.

2.2 Communication Complexity

Let X, Y, Z be finite sets and $R \subseteq X \times Y \times Z$ a ternary relation on them. In the search problem R, we have Alice given some input $x \in X$, Bob is given some input $y \in Y$. Initially, no party knows the other's input, and Alice and Bob both want to output some z such that $(x, y, z) \in R$, by communicating as few bits as possible. A *communication protocol* is a binary tree with each internal node v labeled either by a function $a_v : X \to \{0, 1\}$ if Alice transmits at this node, or $b_v : Y \to \{0, 1\}$ if Bob transmits at this node. Each leaf is labeled by an output $z \in Z$. We say that a protocol solves the search problem for relation R if for every input pair $(x, y) \in X \times Y$, walking down the tree according to the functions a_v and b_v leads to a leaf labeled with some z satisfying $(x, y, z) \in R$. The overall number of bits transmitted for a given input pair $(x, y) \in X \times Y$ and a given protocol is then equal to the length of the walk just described; and the maximum length among these walks equals the depth of the tree. The *(deterministic) communication complexity* $D(R)$ is now defined as the minimum depth among all communication protocols solving R, and the *protocol partition number* $C^P(R)$ denotes the minimum number of leaves among all protocols solving the search problem for R.

Of course, there exist protocols whose depth is even linear in the number of leaves, but a standard argument about balancing binary trees shows that every such deep protocol can be transformed into a shallow protocol, see e.g. [11, ch. 2]:

Lemma 1. $\log C^P(R) \geq \frac{1}{3}D(R)$. □

An important fact about these two complexity measures is a close correspondence with the complexity of Boolean circuits and Boolean formulas, respectively. This relation is based on search problems, which we define next in terms of languages.

For a homogeneous language $\emptyset \subset L \subset \Sigma^n$, the *search problem associated with* L is a ternary relation $R_L \subseteq L \times (\Sigma^n \setminus L) \times [n]$ defined by: $(v, w, i) \in R_L$ iff $v_i \neq w_i$. Karchmer and Wigderson established the following connection to circuit complexity [8]: Take $\Sigma = \{0, 1\}$. If we naturally identify each set $L \subseteq \{0, 1\}^n$ with its characteristic n-bit Boolean function, then $D(R_L)$ equals the minimum depth of a Boolean circuit (over the standard basis) computing the characteristic function of L. Moreover, $C^P(R_L)$ equals the minimum number of variable occurrences among all Boolean formulas representing L.

Proving superpolynomial lower bounds on formula size for specific functions turns out to be an extremely difficult open problem. Fortunately, this is no longer true if we consider monotone Boolean formulas [8]. A similar class of search problems can be defined for the latter setup:

Let $(\Sigma, <) = (a_1 < a_2 < \cdots < a_k)$ be an ordered alphabet. This order on Σ is extended componentwise to a partial order on Σ^n. The *upward closure* of a homogeneous language $L \subseteq \Sigma^n$ (w.r.t. this partial order) is defined as the set

$$\uparrow(L) = \{w \in \Sigma^n \mid u \leq w \text{ for some } u \in L\}.$$

A homogeneous language $L \subseteq \Sigma^n$ is called *monotone*, if $L = \uparrow(L)$. For a monotone homogeneous language $\emptyset \subset L \subset \Sigma^n$, the *monotone search problem associated with* L, denoted by R_L^m, is defined by $(v, w, i) \in R_L^m$ iff both $(v, w, i) \in R_L$ and overmore $v_i > w_i$. For $\Sigma = \{0, 1\}$ with $0 < 1$, the measure $C^P(R_L^m)$ equals the minimum formula size among all *monotone* Boolean formulas representing L, and an similar correspondence holds for $D(R_L^m)$ and *monotone* circuit depth [8].

For more background on communication complexity, the reader might want to consult the book [11].

3 A New Lower Bound Technique for Regular Expression Size

The goal of this section is to relate the alphabetic width of a homogeneous language to the protocol partition number of the monotone search problem associated with that language.

Definition 2. *A regular expression E describing a homogeneous language is called a* homogeneous expression*, if none of the symbols $\emptyset$, ε and * occur in E, or $L(E)$ is empty and $E = \emptyset$.*

Lemma 3. *For $n \geq 1$, let $L \subseteq \Sigma^n$ be a homogeneous language. If E is an uncollapsible regular expression describing L, then E is a homogeneous expression.*

Proof. For the case $L(E) = \emptyset$, the statement immediately follows from the definitions. Assume E is uncollapsible and $\emptyset \subset L(E) \subseteq \Sigma^n$. We can rule out that any subexpression F with $L(F) = \emptyset$ occurs in E: Every regular expression denoting the empty language contains the symbol $\emptyset$ at least once. Next, finiteness of the described languages is invariant under the operations $+$ and $\cdot$, but not by the Kleene star: For any regular expression F, the set denoted by F^* is infinite unless $L(F) = \emptyset$ or $\{\varepsilon\}$. We have already ruled out the existence of $\emptyset$ symbols in E. Since E is uncollapsible, it does not contain any subexpression of the form F^* with $L(F) = \{\varepsilon\}$ either. Thus, the language $L(E)$ being finite, E cannot have any subexpression of the form F^*.

Finally, we rule out the possibility that ε occurs in E: As all words in $L(E)$ are of length n, we make the following observation: If E contains a subexpression of the form $F + G$, then there exists $m \leq n$ such that both $L(F)$ and $L(G)$ contain only strings of length m. If $\mathrm{alph}(E) \leq 1$, then clearly E has no ε-subexpression. Assume $\mathrm{alph}(E) > 1$ and ε occurs in E. Since E is uncollapsible, E contains a subexpression of the form $F + \varepsilon$ with $\varepsilon \notin F$ and $F \neq \emptyset$. But then $F + \varepsilon$ describes a set of strings having different lengths, a property which is inherited to E, since E has no subexpressions describing the empty language. □

The next proposition shows that for homogeneous languages, the upward closure operator $\uparrow$ commutes with union and concatenation.

Proposition 4. *For homogeneous languages L_1 and L_2,*

$$\uparrow(L_1) + \uparrow(L_2) = \uparrow(L_1 + L_2) \text{ and } \uparrow(L_1) \cdot \uparrow(L_2) = \uparrow(L_1 \cdot L_2)\,. \qquad \square$$

We establish next that homogeneous monotone languages can be described by regular expressions in some normal form, and that the conversion into this normal form increases the expression size at most by a factor of $|\Sigma|$.

A homogeneous expression is called a *sum* if it uses $+$ as the only operator, i.e. it is of the form $(b_1 + \ldots + b_m)$ for $b_i \in \Sigma$. Let E be a homogeneous expression and F a subexpression of E. The subexpression F is called a *maximal sum* in E if F is a sum, but each subexpression G having F as a proper subexpression is not a sum. Note that the maximal sums in an expression each describe a subset of Σ. For a homogeneous expression E, the number of maximal sums in E is denoted by $\mathrm{s}(E)$. Since any non-redundant sum is of size at most $|\Sigma|$ and contains at least one alphabetical symbol, we get $\mathrm{s}(L) \leq \mathrm{alph}(L) \leq |\Sigma| \cdot \mathrm{s}(L)$ for every homogeneous language L. A homogeneous expression E is called *monotone* if each maximal sum F in E describes a monotone language, that is $L(F) = \uparrow(L(F))$.

Lemma 5. *For each homogeneous expression E over an ordered alphabet Σ, there exists a monotone expression F with $\mathrm{s}(F) = \mathrm{s}(E)$ and $L(F) = \uparrow(L(E))$. In particular, if E describes a monotone language, then $L(F) = L(E)$.*

Proof. The claim is shown by induction on $\mathrm{s}(E)$. In the base case $\mathrm{s}(E) = 1$, E is itself a sum. Let b be the minimal letter occurring in E, and let $b_1, \ldots, b_m$ be

those letters in Σ with $b_i \geq b$. We set $F := (b_1 + \ldots + b_m)$, and we clearly have $L(F) = \uparrow(L(E))$ as well as $\mathrm{s}(F) = \mathrm{s}(E) = 1$, hence the claim holds.

Now let $\mathrm{s}(E) > 1$, and thus $E = E_1 \oplus E_2$ where the symbol $\oplus$ stands for one of the operators $+$ or $\cdot$, and in the latter case, E_1 and E_2 are not sums. Thus we have $\mathrm{s}(E) = \mathrm{s}(E_1) + \mathrm{s}(E_2)$ and hence $\mathrm{s}(E_i) < \mathrm{s}(E)$ for $i = 1, 2$. By the induction hypothesis, we get expressions F_i with $L(F_i) = \uparrow(L(E_i))$ and $\mathrm{s}(F_i) = \mathrm{s}(E_i)$. We set $F := F_1 \oplus F_2$, and we obtain $\mathrm{s}(F) = \mathrm{s}(F_1) + \mathrm{s}(F_2) = \mathrm{s}(E_1) + \mathrm{s}(E_2) = \mathrm{s}(E)$. By Proposition 4, we obtain $L(F) = \uparrow(L(E_1)) \oplus \uparrow(L(E_2)) = \uparrow(L(E_1) \oplus L(E_2)) = \uparrow(L(E))$, so the claim holds. □

Now we are ready to derive a technique for bounding alphabetic width of homogeneous languages in terms of communication complexity:

Lemma 6. *For every homogeneous language L with $\emptyset \subset L \subset \Sigma^n$ and $n \geq 1$,*

$$\mathrm{alph}(L) \geq \mathrm{s}(L) \geq C^P(R_L)\,.$$

Moreover, if L is monotone, then

$$\mathrm{alph}(L) \geq \mathrm{s}(L) \geq C^P(R_L^m)\,.$$

Proof. Let E be a regular expression with $L(E) = L$. By Lemma 3, we can assume that E is homogeneous. If E is a homogeneous regular expression with $L(E)$ homogeneous, then for every subexpression F of E the language $L(F)$ is homogeneous as well, and we denote by $\lambda(F)$ the length of the words in $L(F)$.

We will now, given a homogeneous regular expression E for L, construct a protocol for R_L with $\mathrm{s}(E)$ many leaves.

Recall that Alice is given an input $x \in L$, Bob a $y \notin L$, and they have to find an index i with $x_i \neq y_i$. At each state of the protocol, Alice and Bob keep a subexpression F of E together with an interval $[i, j]$ of length $j - i + 1 = \lambda(F)$, satisfying the invariant that $x_i \ldots x_j \in L(F)$ and $y_i \ldots y_j \notin L(F)$. At the beginning $F = E$ and $[i, j] = [1, n]$, hence the invariant holds.

At a state of the protocol with a subexpression $F = F_0 + F_1$ with $\mathrm{s}(F) > 1$ and interval $[i, j]$, it must hold that either $x_i \ldots x_j \in L(F_0)$ or $x_i \ldots x_j \in L(F_1)$, but $y_i \ldots y_j \notin L(F_0)$ and $y_i \ldots y_j \notin L(F_1)$. Thus Alice can transmit $\delta \in \{0, 1\}$ such that $x_i \ldots x_j \in L(F_\delta)$, and the protocol continues with F updated to F_δ and $[i, j]$ unchanged.

At a state with subexpression $F = F_0 \cdot F_1$ and interval $[i, j]$, let $\ell := i + \lambda(F_0) - 1$. Then it must hold that $x_i \ldots x_\ell \in L(F_0)$ and $x_{\ell+1} \ldots x_j \in L(F_1)$, but either $y_i \ldots y_\ell \notin L(F_0)$ (case 0) or $y_{\ell+1} \ldots y_j \notin L(F_1)$ (case 1). Then Bob can transmit $\delta = 0, 1$ such that case δ holds, and the protocol continues with F updated to F_δ and $[i, j]$ set to $[i, \ell]$ in case 0 and $[\ell + 1, j]$ in case 1.

At a state with a subexpression F that is a maximal sum in E, it must be the case that $i = j$, and that $x_i \in L(F)$ and $y_i \notin L(F)$, hence in particular $x_i \neq y_i$ and the protocol can terminate with output i.

Obviously, the protocol solves R_L, and the tree of the protocol constructed is isomorphic to the parse tree of E with its maximal sums at the leaves, thus the number of leaves is $\mathrm{s}(E)$.

If L happens to be monotone, then by Lemma 5 we can assume that E is a monotone expression. Then also all subexpressions of E that appear in the above proof are monotone, and in the terminating case it must moreover be the case that $x_i > y_i$, therefore the protocol solves R_L^m. □

4 Lower Bounds for the Conversion Problem

For given integers ℓ, n, we define a family of graphs $\mathcal{F}_{\ell,n}$ with parameters ℓ, n as the set of directed acyclic graphs whose vertex set V is organized in $\ell+2$ layers, with n vertices in each each layer. Hence we assume $V = \{\, \langle i,j\rangle \mid 1 \le i \le n, 0 \le j \le \ell+1\}$. For all graphs in $\mathcal{F}_{\ell,n}$, we require in addition that each edge connects a vertex in some layer i to a vertex in the adjacent layer $i+1$.

The following definition serves to represent the set $\mathcal{F}_{\ell,n}$ as a finite set of strings over the alphabet $\{0,1\}$: Fix a graph $G \in \mathcal{F}_{\ell,n}$ for the moment. Let $e(i,j,k) = 1$ if G has an edge from vertex i in layer j to vertex k in layer $j+1$, and let $e(i,j,k) = 0$ otherwise. Next, for vertex i in layer j, the word $f(i,j) = e(i,j,1)e(i,j,2)\cdots e(i,j,n)$ encodes the set of outgoing edges for this vertex. Then for layer j, the word $g(j) = f(1,j)f(2,j)\cdots f(n,j)$ encodes the set of edges connecting vertices in layer j to vertices in layer $j+1$, for $0 \le j \le \ell$. Finally, the graph G is encoded by the word $w(G) = g(0)g(1)\cdots g(\ell)$. It is easy to see that each word in the set $\{0,1\}^{n^2(\ell+1)}$ can be uniquely decoded as a graph in the set $\mathcal{F}_{\ell,n}$.

A graph $G \in \mathcal{F}_{\ell,n}$ belongs to the subfamily $\mathsf{fork}_{\ell,n}$, if there exists a simple path starting in $\langle 1,0\rangle$ ending eventually in a fork, i.e., a vertex of outdegree at least two. The goal of this section will be to show that the language

$$L = L_{\ell,n} = \{\, w \in \{0,1\}^{n^2(\ell+1)} \mid w = w(G) \text{ with } G \in \mathsf{fork}_{\ell,n}\,\}$$

can be accepted by a DFA of size polynomial in both parameters, while this cannot be the case for any regular expression describing this language.

Proposition 7. *For every pair (ℓ,n) with $\ell \ge 2$ and $n \ge 5$, the language $L_{\ell,n}$ can be accepted by a DFA having at most $\ell \cdot n^4$ states.*

Proof. We describe a DFA A accepting L, which has special states q_i^j for $1 \le i \le n$ and $0 \le j \le \ell+1$. These states will have the following property: If G has a simple non-forking path starting in vertex $\langle 1,0\rangle$ and ending in vertex $\langle i,j\rangle$, then the DFA is in state $q_{i,j}$ after reading the first $j \cdot n^2$ letters of the word $w(G)$. A DFA having this property is obtained by setting q_1^0 to be the start state, and by applying the construction shown in Figure 1 one by one to all states q_i^j, for $1 \le i \le n$ and $0 \le j \le \ell$. Each transition in Figure 1 labeled with some regular expression has to be unrolled to a simple path.

To complete the construction, we have to ensure that from every state q for which $\delta(q,1)$ is not yet defined, the transition $\delta(q,1)$ leads to a state that leads every suffix of admissible length to an accepting state. This will be achieved by adding the transition structure of another deterministic finite automaton

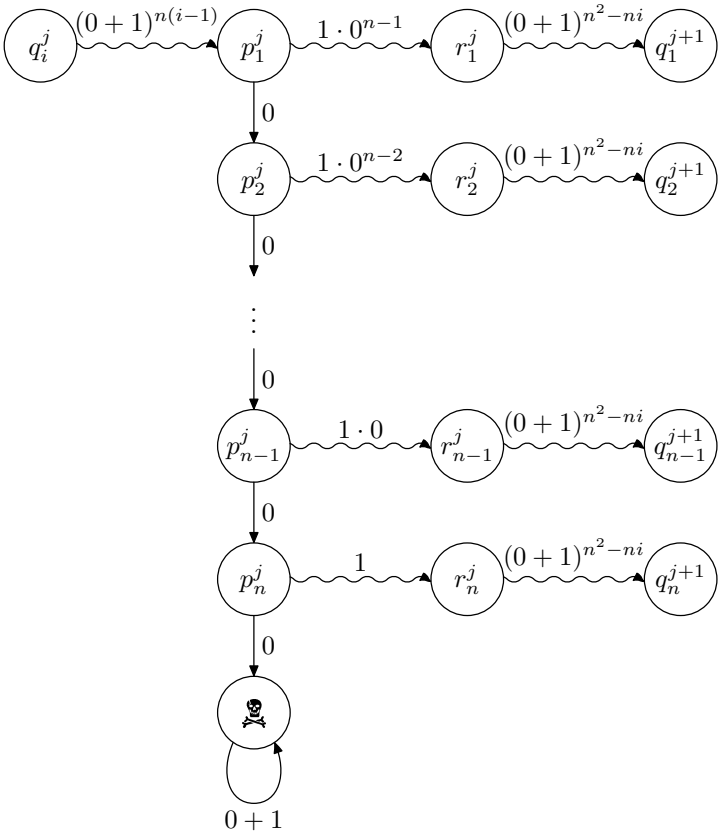

Fig. 1. Connecting q_i^j to the corresponding states in the next layer

that accepts $\{0,1\}^{n^2(\ell+1)}$, and routing the lacking transitions into states of this automaton appropriately, in a way that each time the suffixes of admissible length are accepted.

Finally, we count the number of states in A: Unrolling the construction depicted in Figure 1 introduces

$$\sum_{j=0}^{\ell} \sum_{i=1}^{n} \left(n \cdot i + \sum_{k=1}^{n} (n+1-k+n^2-n \cdot i) \right)$$

states, excluding the dead state. All dead states can be merged, and the minimal DFA accepting the language $\{0,1\}^{n^2(\ell+1)}$ has $n^2(\ell+1)+2$ states, one of which is already a dead state. By adding up and simplifying, we see that the number of states equals

$$(\ell+1)(\frac{1}{2}n^4 + \frac{1}{2}n^3 + 2n^2) + 2.$$

We have $\ell + 1 < \sqrt{2}\ell$ and $2/(\ell+1) < 1$ provided $\ell \geq 2$, and for $n \geq 5$ holds $\frac{1}{2}n^4 + \frac{1}{2}n^3 + 2n^2 + 1 < \frac{n^4}{\sqrt{2}}$, and thus we can conclude that the number of states is bounded above by $\ell \cdot n^4$, provided $\ell \geq 2$ and $n \geq 5$. □

Next, we give a lower bound on the alphabetic width of this language. To this end, we show that the communication complexity of the monotone search problem for $L_{\ell,n}$ is bounded below by the communication complexity of the relation $\mathsf{FORK}_{\ell,n}$, which is defined as follows (cf. [11, ch. 5.3]):

Let $W := \{1, \ldots, n\}^\ell$. The relation $\mathsf{FORK}_{\ell,n}$ is a subset of $W \times W \times \{0, 1, \ldots, \ell\}$ For two strings $x = x_1 x_2 \cdots x_\ell$ and $y = y_1 y_2 \cdots y_\ell$, and $i \in \{0, 1, \ldots, \ell\}$ we have $(x, y, i) \in \mathsf{FORK}_{\ell,n}$ iff $x_i = y_i$ and $x_{i+1} \neq y_{i+1}$, with the convention that $x_0 = y_0 = 1$, $x_{\ell+1} = n - 1$ and $y_{\ell+1} = n$. The following lower bound on this relation is found in the monograph [11] and is due to[1] Grigni and Sipser [4]:

Lemma 8. $D(\mathsf{FORK}_{\ell,n}) \geq \lfloor (\log n)/4 \rfloor \cdot \lfloor \log \ell \rfloor$ □

It remains to give a reduction from $\mathsf{FORK}_{\ell,n}$ to the monotone search problem.

Lemma 9. *Let $L = L_{\ell,n}$. Then $D(R_L^m) \geq \lfloor \frac{1}{4} \log n \rfloor \cdot \lfloor \log \ell \rfloor$.*

Proof. We show that for $L = L_{\ell,n}$, any protocol that solves R_L^m can be used to solve $\mathsf{FORK}_{\ell,n}$ without any additional communication, which implies the stated lower bound. The reduction is similar to one used by Grigni and Sipser [4].

From her input $x \in W$, Alice computes a graph $G_x \in \mathcal{F}_{\ell,n}$ having for every $0 \leq i \leq \ell$ an edge from $\langle x_i, i \rangle$ to $\langle x_{i+1}, i+1 \rangle$, and an additional edge from $\langle x_\ell, \ell \rangle$ to $\langle n, \ell + 1 \rangle$. By construction, $G_x \in \mathsf{fork}_{\ell,n}$ and thus $w(G_x) \in L_{\ell,n}$.

Similarly, from his input $y \in W$, Bob computes a graph G_y having for every $0 \leq i \leq \ell$ an edge from $\langle y_i, i \rangle$ to $\langle y_{i+1}, i+1 \rangle$. Additionally, G_y has all the edges from $\langle i, j \rangle$ to $\langle i', j+1 \rangle$ where $i \neq y_j$ and i' is arbitrary. Therefore $G_y \notin \mathsf{fork}_{\ell,n}$ and thus $w(G_y) \notin L_{\ell,n}$.

Now running the protocol for R_L^m on $w(G_x)$ and $w(G_y)$ yields a position k where $w(G_x)_k = 1$ and $w(G_y)_k = 0$, i.e., an edge that is present in G_x, but not in G_y. By construction, this edge goes from $\langle x_i, i \rangle$ to $\langle x_{i+1}, i+1 \rangle$ for some i, and it must be that $y_i = x_i$ and $y_{i+1} \neq x_{i+1}$, as otherwise the edge would be present in G_y. Thus i is a solution such that $(x, y, i) \in \mathsf{FORK}_{\ell,n}$. □

Theorem 10. *There exist infinitely many languages L_m such that L_m is acceptable by a DFA with at most m states, but*

$$\mathrm{alph}(L_m) \geq m^{\frac{1}{75} \log m}.$$

Proof. For an integer k, we choose $n = 2^{4k}$, $\ell = n^4$ and $m = n^5$. Then our witness language is $L = L_m = L_{n^4,n}$. This language is acceptable by a DFA with at most $m = n^5$ states. In contrast, we have $D(R_L^m) \geq (\log n)^2$ by Lemma 9, which together with Lemma 1 implies that $C^P(R_L^m) \geq 2^{\frac{1}{3}(\log n)^2} = m^{\frac{1}{75} \log m}$, and the latter is a lower bound for the alphabetic width of the language L_m. □

[1] In fact, Grigni and Sipser investigated a relation that is slightly different from the one used in [11] and in this work.

5 Strength and Limitations

In this section, we illustrate the power and limitations of the techniques we introduced. We show that our lower bound technique sometimes gives tight lower bounds, although the gap between the lower bound and the actual minimum regular expression size can be exponential, that is, we cannot hope that $C^P(R_L)$ has a performance guarantee for regular expressions similar to the case of Boolean formulas.

5.1 A Poor Lower Bound

For n even, consider the languages of palindromes of length n, $L_n = \{\, ww^R \mid w \in \{0,1\}^{n/2} \,\}$. To give an upper bound on $C^P(R_{L_n})$, recall from Section 2.2 that this number equals the minimum number of variable occurrences among all Boolean formulas describing the characteristic function of L_n. The following formula of size $2n$ describes the characteristic function:

$$\bigwedge_{i=1}^{n/2} (x_i \wedge x_{n/2+i}) \vee (\neg x_i \wedge \neg x_{n/2+i}).$$

For a lower bound on $\mathrm{alph}(L)$, we use the well known fact that $\mathrm{alph}(L)$ is bounded below by the minimum number of states required by a nondeterministic finite automaton accepting L. However, it is well known that every nondeterministic finite automaton accepting L_n has size exponential in n [15].

5.2 Optimal Expressions for Parity

Let par_n denote the parity language $\{w \in \{0,1\}^n \,;\, |w|_1 \text{ odd}\,\}$. In [3], it is shown that $\mathrm{alph}(\mathsf{par}_n) = \Omega(n^2)$ using Khrapchenko's bound [9] on the Boolean formula size of the parity function. From a recent improvement of this bound by Lee [12], we obtain the following better lower bound:

Theorem 11. *If E is a regular expression with $L(E) = \mathsf{par}_n$, and $n = 2^d + k$ with $k < 2^d$, then* $\mathrm{alph}(E) \geq 2^d(2^d + 3k)$.

We will now construct regular expressions for par_n that exactly match this lower bound. The construction is essentially the same as Lee's [12] upper bound for the size of Boolean formulas for parity, but our analysis is simpler, using only induction and elementary arithmetic.

We have that $\mathsf{par}_n = L(\mathsf{odd}_n)$, where the expressions even_n and odd_n are defined inductively by

$$\begin{aligned}
\mathsf{even}_1 &:= 0 \qquad \mathsf{odd}_1 := 1 \\
\mathsf{even}_{2m} &:= (\mathsf{even}_m \cdot \mathsf{even}_m) + (\mathsf{odd}_m \cdot \mathsf{odd}_m) \\
\mathsf{odd}_{2m} &:= (\mathsf{even}_m \cdot \mathsf{odd}_m) + (\mathsf{odd}_m \cdot \mathsf{even}_m) \\
\mathsf{even}_{2m+1} &:= (\mathsf{even}_{m+1} \cdot \mathsf{even}_m) + (\mathsf{odd}_{m+1} \cdot \mathsf{odd}_m) \\
\mathsf{odd}_{2m+1} &:= (\mathsf{even}_{m+1} \cdot \mathsf{odd}_m) + (\mathsf{odd}_{m+1} \cdot \mathsf{even}_m)
\end{aligned}$$

First we observe that $\mathrm{alph}(\mathsf{even}_n) = \mathrm{alph}(\mathsf{odd}_n)$ for every n, and we denote it by $r(n) := \mathrm{alph}(\mathsf{even}_n)$. Then the function $r(n)$ satisfies the following recursive equations:

$$r(1) = 1 \qquad r(2m) = 4r(m) \qquad r(2m+1) = 2r(m+1) + 2r(m)$$

We now show that if $n = 2^d + k$ with $k < 2^d$, then $r(n) = 2^d(2^d + 3k)$, by induction on n. Thus our expressions match Lee's lower bound.

The case $n = 1$ is obvious. For the induction step, we distinguish three cases. The first case is $n = 2m$ where $m = 2^d + k$, hence $n = 2^{d+1} + 2k$. In this case we have

$$r(n) = 4r(m) = 4 \cdot 2^d(2^d + 3k) = 2^{d+1}(2^{d+1} + 6k).$$

The second case is $n = 2m + 1$ where $m = 2^d + k$ and $m + 1 = 2^d + (k + 1)$ with $k + 1 < 2^d$, hence $n = 2^{d+1} + 2k + 1$ with $2k + 1 < 2^{d+1}$. In this case we obtain

$$\begin{aligned} r(n) &= 2r(m+1) + 2r(m) &&= 2^{d+1}(2^d + 3k + 3) + 2^{d+1}(2^d + 3k) \\ &= 2^{d+1}(2^{d+1} + 6k + 3)\ . \end{aligned}$$

The final case is $n = 2m+1$, where $m = 2^d+k$ and $m+1 = 2^{d+1}$, thus $k = 2^d-1$ and $n = 2^{d+1} + (2^{d+1} - 1)$. In this case we calculate

$$\begin{aligned} r(n) &= 2r(m+1) + 2r(m) &&= 2^{2d+3} + 2^{d+1}(2^d + 3(2^d - 1)) \\ &= 2^{d+1}(2^{d+2} + 2^d + 3(2^d - 1)) &&= 2^{d+1}(2^{d+1} + 2^{d+1} + 2^d + 3(2^d - 1)) \\ &= 2^{d+1}(2^{d+1} + 3 \cdot 2^d + 3(2^d - 1) &&= 2^{d+1}(2^{d+1} + 3(2^{d+1} - 1)) \end{aligned}$$

which shows the claim.

6 Upper Bounds for Converting NFAs into Regular Expressions

In this section, we identify a family of finite languages H_n which are the hardest finite languages for the NFA to RE conversion problem. The term *hardest* is made precise in the statement of the theorem below. The languages H_n were also studied in [2], where it was shown that $\mathrm{alph}(H_n) = n^{\Omega(\log \log n)}$.

Theorem 12. *For $n \geq 1$, let $G_n = (V_n, \Delta)$ be the complete directed acyclic graph on n vertices, that is $V_n = \{1, 2, \ldots, n\}$ and edge set $\Delta = \{\langle i, j\rangle \mid 1 \leq i < j \leq n\}$. Define the language $H_n \subset \Delta^{\leq n-1}$ as the set of all paths in G leading from vertex 1 to vertex n. Then the following holds:*

1. *H_n can be accepted by an n-state nondeterministic finite automaton.*
2. *Let Σ be an alphabet. For every finite language L over Σ acceptable by an n-state nondeterministic finite automaton holds* $\mathrm{alph}(L) \leq |\Sigma| \cdot \mathrm{alph}(H_n)$.

Proof. The first statement is easy to see. For the second statement, assume the theorem holds for all values up to $n-1$, and let A be an n-state nondeterministic finite automaton accepting $L \subseteq \Sigma^{\leq n-1}$. Without loss of generality, we assume A has state set $\{q_1, q_2, \ldots, q_n\}$ and the states are in topological order with respect to the transition structure of the directed acyclic graph underlying A, that is, the automaton cannot move from state q_j to state q_i if $i \leq j$. Furthermore, we can safely assume that the automaton has start state q_1 and single accepting state q_n. This can be achieved by the following construction: If q_1 is not the start state, and the state set is topologically ordered, then q_1 is not reachable from the start state. q_1 can be removed, and we can apply the theorem for the obtained $n-1$-state automaton. For similar reasons, we can assume that q_n is a final state. If A has another final state p, we add transitions such that for every transition entering p there is now a transition from the same source entering q_n. Then we remove p from the set of final states. Clearly, the accepted language is not altered by this construction, and the number of states remains n.

Let H be the minimal partial n-state deterministic finite automaton accepting H_n, i.e. the automaton has no dead state. We again assume that the state set of H is topologically ordered, as for A.

Let F and G be the regular expressions obtained by applying the standard state elimination algorithm [7,14] to the automata H and A, respectively. Since the algorithm is correct, we have $L(F) = H_n$, and $L(G) = L(A)$. For a pair of states (q_i, q_j) with $i < j$ in A, define the regular expression F_{ij} as the minimal expression describing the union of all transition labels under which the automaton can change its state from q_i to q_j. Then by the properties of the transformation algorithm holds $G = \mathrm{sub}(F)$, where sub is the substitution replacing every occurrence of the atomic expression $\langle i, j \rangle$ with an occurrence of the expression (F_{ij}).

Now let F' be an expression of minimal alphabetic width describing H_n, that is $L(F') = L(F)$. Then this equality is derivable using a sound and complete proof system for regular expression equations, e.g. see [19]. Then we can derive the equality $L(\mathrm{sub}(F')) = L(\mathrm{sub}(F))$ by a single application of the substitution rule [19], and recall $\mathrm{sub}(F) = G$. To estimate the size of $L(G)$, we simply observe that $\mathrm{alph}(F_{ij}) \leq |\Sigma|$, for all i, j. □

Thus an algorithm which does the job for the n-state automaton accepting H_n will not perform much worse given any other finite automaton of equal size. In doing this, we obtain a slightly improved upper bound for the conversion problem for all finite languages — the currently best known method [3, Cor. 22] gives a bound of $(n+1) \cdot kn(n-1)^{\log n + 1}$:

Corollary 13. *Let A be an n-state nondeterministic finite automaton accepting a finite language $L = L(A)$ over a k-symbol alphabet. Then*

$$\mathrm{alph}(L) < k \cdot n(n-1)^{\log(n-1)+1}$$

Proof. By the preceding theorem, it suffices to give an upper bound on $\mathrm{alph}(H_n)$. The language H_n coincides with set of all walks (of length at most $n-1$) in G_n that start in vertex 1 and end in vertex n. The analysis given in [3, Thm. 20]

implies that there exists a regular expression of size at most $n(n-1)^{\log(n-1)+1}$ describing this set, since for each pair (i, j), there is a regular expression of size at most 1 describing the set of walks of length at most 1 in G_n starting in i and end in j. Thus, $\text{alph}(H_n) \leq n(n-1)^{\log(n-1)+1}$. □

7 Conclusions and Further Work

We developed a new lower bound technique for regular expression size to show that converting deterministic finite automata accepting finite languages into regular expressions leads to an inevitable blow-up in size of $n^{\Theta(\log n)}$, solving an open problem stated in [2]. This bound still holds when restricting to alphabet size two. Note that finite automata accepting unary finite languages can be easily converted into regular expressions of linear size.

Compared to the finite automaton model, we feel that we have still a limited understanding of the power of regular expressions in terms of descriptional complexity. For instance, although we have determined an asymptotically tight bound of $n^{\Theta(\log n)}$, there is still a considerable gap between the two constants implied by the Θ-Notation. This is in stark contrast with the deterministic finite automaton model, where exact bounds are known for many questions regarding descriptional complexity, see [21] for an overview.

Another interesting line of research concerns lower bounds for the conversion problem on infinite languages, over alphabets of constant size. This problem has been solved in [5], where a corresponding lower bound of $2^{\Omega(n)}$ was established, given an n-state DFA over a binary alphabet accepting an infinite language. There a different proof technique based on digraph connectivity was used, which only gives trivial lower bounds for finite languages. That paper also contains lower bounds on alphabetic width of some basic regular language operations, such as intersection, shuffle, and complement. The effect of language operations on alphabetic width in the case of finite languages is also of interest: For finite automata accepting finite languages, the descriptional complexity often differs from the general case, see e.g. [21]. The lower bound techniques developed in this paper might be useful in that context.

Acknowledgment. We would like to thank Jeffrey Shallit for kindly providing us a copy of the corrected final version [3], which by now has already appeared.

References

1. Delgado, M., Morais, J.: Approximation to the smallest regular expression for a given regular language. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 312–314. Springer, Heidelberg (2005)
2. Ehrenfeucht, A., Zeiger, H.P.: Complexity measures for regular expressions. Journal of Computer and System Sciences 12(2), 134–146 (1976)
3. Ellul, K., Krawetz, B., Shallit, J., Wang, M.: Regular Expressions: New Results and Open Problems. Journal of Automata, Languages and Combinatorics 10(4), 407–437 (2005)

4. Grigni, M., Sipser, M.: Monotone separation of logarithmic space from logarithmic depth. Journal of Computer and System Sciences 50, 433–437 (1995)
5. Gruber, H., Holzer, M.: Finite automata, digraph connectivity and regular expression size. Technical report, Technische Universität München (December 2007)
6. Han, Y., Wood, D.: Obtaining shorter regular expressions from finite-state automata. Theoretical Computer Science 370(1–3), 110–120 (2007)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
8. Karchmer, M., Wigderson, A.: Monotone circuits for connectivity require superlogarithmic depth. SIAM Journal on Discrete Mathematics 3, 255–265 (1990)
9. Khrapchenko, V.M.: Methods for determining lower bounds for the complexity of π-schemes (English translation). Math. Notes Acad. Sciences USSR 10, 474–479 (1972)
10. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, Annals of Mathematics Studies, pp. 3–42. Princeton University Press, Princeton (1956)
11. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, New York (1997)
12. Lee, T.: A new rank technique for formula size lower bounds. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, Springer, Heidelberg (2007)
13. Martinez, A.: Efficient computation of regular expressions from unary NFAs. In: Dassow, J., Hoeberechts, M., Jürgensen, H., Wotschke, D. (eds.) Workshop on Descriptional Complexity of Formal Systems 2002, London, Canada, pp. 216–230 (2002)
14. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRA Transactions on Electronic Computers 9(1), 39–47 (1960)
15. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: IEEE Symposium on Switching and Automata Theory 1971, pp. 188–191 (1971)
16. Morais, J.J., Moreira, N., Reis, R.: Acyclic automata with easy-to-find short regular expressions. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 349–350. Springer, Heidelberg (2006)
17. Morris, P.H., Gray, R.A., Filman, R.E.: Goto removal based on regular expressions. Journal of Software Maintenance 9(1), 47–66 (1997)
18. Sakarovitch, J.: The language, the expression, and the (small) automaton. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 15–30. Springer, Heidelberg (2006)
19. Salomaa, A.: Two complete axiom systems for the algebra of regular events. Journal of the ACM 13(1), 158–169 (1966)
20. Schnitger, G.: Regular expressions and NFAs without ε-transitions. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 432–443. Springer, Heidelberg (2006)
21. Yu, S.: State complexity of finite and infinite regular languages. Bulletin of the EATCS 76, 142–152 (2002)