Regular expression length via arithmetic formula complexity

Ehud Cseresnyes, Hannes Seiwert*

Institute of Computer Science, Goethe University Frankfurt, Germany

Abstract

We prove lower bounds on the length of regular expressions for finite languages by methods from arithmetic circuit complexity. First, we show a reduction: the length of a regular expression for a language $L \subseteq \{0, 1\}^{n}$ is bounded from below by the minimum size of a monotone arithmetic formula computing a polynomial that has $L$ as its set of exponent vectors, viewing words as vectors. This result yields lower bounds for the binomial language of all words with exactly $k$ ones and $n-k$ zeros and for the language of all Dyck words of length $2n$. We also determine the blow-up of language operations (intersection and shuffle) of regular expressions for finite languages. Second, we adapt a lower bound method for multilinear arithmetic formulas by so-called log-product polynomials to regular expressions. With this method we show almost tight lower bounds for the language of all binary numbers with $n$ bits that are divisible by a given odd integer $p$, for the language of all words of length $n$ over a $k$ letter alphabet with an even number of occurrences of each letter and for the language of all permutations of $\{1, \ldots, n\}$.

Keywords: regular expression, lower bound, descriptional complexity, arithmetic circuit complexity, formal language, monotone arithmetic formula

1. Introduction

Deriving lower bounds on the length of regular expressions is a fundamental problem in formal language theory [2, 3, 4, 5, 6, 7, 8]. Particularly interesting are language families that have small finite automata but require long regular expressions, since they reveal the gap between the descriptional complexity of these models. However, despite regular expressions being around for some decades, only few lower bounds are known so far. This is in sharp contrast to state complexity of finite automata which is understood quite well.

One has to distinguish between finite and infinite languages, as well as between alphabets of constant and growing size. Ehrenfeucht and Zeiger [2] gave an exponential lower bound for the infinite language of all walks in a complete graph over an alphabet of size $n^2$, thereby showing that the blow-up of converting a deterministic finite automaton (DFA) into a regular expression may be exponential. This result was generalized by Gelade and Neven [6] for four-letter alphabets, and by Gruber and Holzer [5] for binary alphabets using concepts of cycle rank and star height; in particular, Gruber and Holzer showed that the length of any regular expression is exponential in the star height of its described language.

In this paper, we focus on finite languages. Since star height of finite languages trivially is zero, this method is not applicable here. Instead, techniques from circuit complexity have proven useful. In the next subsection we give an overview of the most important lower bound methods for finite languages and briefly discuss their strengths and weaknesses. Subsequently we present our own results.

*A preliminary version of this paper appeared in Descriptional Complexity of Formal Systems [1]. The second author was partially supported by the DFG grant JU 3105/1-2 (German Research Foundation).

*Corresponding author.

Email addresses: ehu@posteo.de (Ehud Cseresnyes), seiwert@em.uni-frankfurt.de (Hannes Seiwert)
1.1. Related work

As customary, a regular expression (or just expression) consists of the binary operations union (+) and concatenation (·), the unary star operation (∗), the empty language ∅, the empty word ε, and letters a of an alphabet Σ. Every expression R describes a regular language L(R) in the usual way. Note that for finite languages the star operation is useless and can be avoided. For an expression R we measure its length by the reverse polish length \( \text{rpn}(R) \), namely the number of nodes in its syntax tree. For a language L its expression length \( \text{rpn}(L) \) is the length of a shortest expression describing L. A language is homogeneous if all its words have the same length. For a word w and a letter a we denote by \(|w|_a\) the number of occurrences of a in w.

Circuits and formulas over semirings. A regular expression without star is nothing else than a formula (that is, a circuit whose graph is a tree) over the non-commutative free semiring \((+, ·)\). So, lower bounds on the size of formulas (or circuits) over semirings can also be applied to expression length. In particular, if we interpret union (+) as addition and concatenation (·) as multiplication, bounds for monotone arithmetic formulas (or circuits) also carry over.

For example, Jerrum and Snir [9] showed several such lower bounds on circuit size. By additionally taking into account the non-commutativity of the concatenation operation, they showed a lower bound of \(2^n - 2\) for the language \(P_n\) of all permutations over the alphabet \(Σ = [n]\), see [9, Sect. 5.4]. (Recently, Molina Lovett and Shallit [8] improved this bound to \(4^n n^{(\log n)/4+\Theta(1)}\) by a custom argument.) Also Hrubes, Wigderson and Yehudayoff [10, 11] developed a lower bound method for non-commutative circuits, and Filmus [12] used a similar method for lower bounds on the size of context-free grammars for finite languages.

By using a lower bound on monotone arithmetic formula size, Hrubes and Yehudayoff [13] (implicitly) showed that the language \(L_{n,k} = \{i_1i_2⋯i_k : 1 ≤ i_1 < i_2 < ⋯ < i_k ≤ n\}\) of all length-k subwords of the word \(12⋯n\) over the alphabet \(Σ = [n]\) requires expressions of length \(nk\Omega(\log k)\), provided \(k ≤ n/2\).

Fooling sets. It is well known that every expression of length \(s\) can be transformed into a nondeterministic finite automaton (NFA) with \(s + 1\) states. Therefore, lower bounds for NFAs carry over to expressions. Such bounds can be shown by the fooling set method of Birget [14], rediscovered by Glaister and Shallit [15].

Given a language L, a fooling set for L is a set of pairs of words \((u_i, v_i)\) such that the word \(u_i v_i\) lies in L for all \(i\), but for all \(i ≠ j\) one of the words \(u_i v_j\) or \(u_j v_i\) lies not in L. The number of states of every NFA for L is bounded from below by the size of a largest fooling set for L.

Example 1. Consider the language \(L_{2n}^{\text{pal}} = \{ww^{\text{reverse}} : w ∈ \{0, 1\}^n\}\) of all palindromes of length \(2n\). Then \(F = \{(w, w^{\text{reverse}}) : w ∈ \{0, 1\}^n\}\) is a fooling set for \(L_{2n}^{\text{pal}}\). Hence, any NFA for \(L_{2n}^{\text{pal}}\) has at least \(|F| = 2^n\) states and any expression has length at least \(2^n - 1\).

Reduction to boolean formula complexity. Ellul, Krawetz, Shallit and Wang [3] described a construction that transforms a given expression R for a homogeneous language \(L ⊆ \{0, 1\}^n\) into a boolean formula of size \(\text{rpn}(R)\) for its characteristic function \(f_L : \{0, 1\}^n → \{0, 1\}\). This transformation is done as follows: Every letter 1 (resp. 0) is replaced by the literal \(x_i\) (resp. \(¬x_i\)), where the index \(i\) corresponds to the position of the letter in the words of L. Every union is replaced by OR and every concatenation is replaced by AND.

Hence, the expression length of L is bounded from below by the boolean formula complexity of \(f_L\):

\[\text{rpn}(L) ≥ \text{Boolean}(f_L)\]

Together with a result by Khrapchenko [16] on the formula complexity of the XOR function, they derived a tight lower bound of \(Ω(n^2)\) for the XOR language \(L_n^{\text{xor}} = \{w ∈ \{0, 1\}^n : |w|_1 \text{ is even}\}\).

Reduction to monotone boolean formula complexity. A language \(L ⊆ \{0, 1\}^*\) is monotone if it is closed under replacing any number of zeros by ones. By methods from communication complexity, Gruber and Johannsen [4] developed a construction that transforms a given expression for a monotone homogeneous language \(L ⊆ \{0, 1\}^n\) into a monotone boolean formula for \(f_L\). Thus, the expression length of L is bounded from below by the monotone boolean formula complexity of \(f_L\):

\[\text{rpn}(L) ≥ \text{Monotone-Boolean}(f_L)\]
With this reduction they showed that the blow-up for converting a DFA accepting a finite language into an expression may be as large as $n^{o(\log n)}$. This result is based upon a lower bound on the monotone boolean complexity for the FORK problem shown by Grigori and Sipser [17]. Since any DFA (or NFA) for a finite language can be simulated by an expression of length $n^{O(\log n)}$ [3, 4], this result is tight. In particular, expression length of finite languages with polynomial size NFAs is at most quasi-polynomial.

**Strengths and limitations.** We give a brief discussion of the above methods, focusing on two desirable properties: (i) the ability to show blow-ups between finite automata and expressions, and (ii) the possibility to work over constant size alphabets.

General circuit lower bound methods may work well for languages over alphabets of growing size. Often, however, and in particular when working over constant size alphabets, non-commutativity must be used explicitly. For example, if we ignore non-commutativity, the permutation language $P_n$ collapses to the trivial language $\{12\cdots n\}$ containing only a single word. Moreover, circuit methods cannot show nontrivial blow-ups since NFAs can be simulated by circuits (see [18]). However, formula size methods can (e.g., the aforementioned language $L_{n,k} = \{i_1\cdots i_k : 1 \leq i_1 < \cdots < i_k \leq n\}$ has small DFAs with $O(nk)$ states, but no short expressions).

The fooling set method can be applied to languages over arbitrary alphabets. It is usually simpler in its application than circuit methods and gives the same (sometimes even better) bounds, so it is often a promising first attempt. However, since lower bounds shown with fooling sets hold also for NFAs, clearly this method cannot show blow-ups either.

The two “boolean methods” in contrast can show blow-ups and also work for constant size (even binary) alphabets. On the downside, they rely on lower bounds for boolean (or monotone boolean) complexity which are rare and whose proofs are usually quite involved. Also, bounds obtained by the boolean methods can be rather loose for many languages (see Section 5.3 and Remarks 3 and 5) and are restricted to languages over the alphabet $\{0, 1\}$.

**Our contribution.** Our goal in this paper is to improve the existing methods with regard to strength as well as to simplicity of application. We present two new methods that naturally refine the boolean methods resp. the formula size method. Both methods have our two desired properties (i) and (ii), that is, they are able to show blow-ups and work over constant size alphabets. As demonstration of these methods, we also show several explicit lower bounds.

2. Results

**Reduction to monotone arithmetic formula complexity.** A monotone arithmetic formula is a rooted tree with leaves holding variables $x_1, \ldots, x_n$ and inner nodes (gates) performing multiplication ($\times$) or addition ($+$) operations. Any such formula computes a polynomial $f(x_1, \ldots, x_n) = \sum_{a \in A} \lambda_a \prod_{i=1}^n x_i^{a_i}$ over the nonnegative reals in a natural manner, where $A \subseteq \mathbb{N}^n$ is its set of exponent vectors and $\lambda_a > 0$ are positive coefficients. For a set $A \subseteq \mathbb{N}^n$ let $\text{arith}_F(A)$ be the size of a smallest monotone arithmetic formula that computes a polynomial whose set of exponent vectors is $A$. We identify words $w = w_1 \cdots w_n$ with vectors $(w_1, \ldots, w_n)$.

Our first method reduces expression length to monotone arithmetic formula complexity.

**Result 1** (Arithmetic bound, Theorem 4.2). Let $L \subseteq \{0, 1\}^n$ be a homogeneous language. Then

$$\text{rpm}(L) \geq \text{arith}_F(L).$$

Informally stated, the following hierarchy for the different formula complexities holds:

$$\text{Monotone-Arithmetic} \geq \text{Monotone-Boolean} \geq \text{Boolean}$$

The gaps between these complexities can be exponentially large (see Remark 3). So, our arithmetic bound covers both aforementioned boolean methods, and can be exponentially stronger. Another advantage is that bounds on arithmetic complexity can be proven much more easily than on boolean complexity. In particular, there are already many strong bounds known. In contrast to the “monotone boolean method” of Gruber and Johanssen [4], the arithmetic bound is not restricted to monotone languages. For a survey on boolean complexity resp. arithmetic complexity, including the non-monotone case, see [19] resp. [20, 21].
The log-product bound. A general flaw of Result 1 is that the non-commutativity of the concatenation operation cannot be fully utilized since arithmetic operations (+ and ×) are commutative. Further, it is restricted to languages over the alphabet Σ = {0, 1}. To cope with these issues, we adapt a lower bound method from Shilpka and Yehudayoff [20] resp. Hrubéš and Yehudayoff [13] for multilinear arithmetic formula size directly to expression length. (Essentially, we include non-commutativity in their lower bound method for formula size, as similarly done in [10, 11] for a circuit size method.)

An expression R is homogeneous if it describes a homogeneous language (all words have the same length) and its degree deg R is the length of its described words. A homogeneous expression B is log-product if it is either a factor or a concatenation of two homogeneous expressions B1, B2 such that deg B1 ≥ deg B2 and B1 is log-product itself. We show that every homogeneous expression R can be written as union B1 + · · · + Bℓ of ℓ = rpn(R) log-product expressions Bi. This yields our second method.

Result 2 (Log-product bound, Theorem 6.2). Let L be a homogeneous language and h ∈ R≥0. If |L(B)| ≤ h holds for every log-product expression B with L(B) ⊆ L, then any expression for L has length at least

\[ rpn(L) ≥ |L|/h. \]

So, in order to show a lower bound on rpn(L), we only have to upper bound the number of words in L(B) for any log-product expression B with L(B) ⊆ L. Every log-product expression can be written as factorization B = F1F2 · · · Fn of m ≥ log(deg B) nontrivial factors Fi (see Section 6.2 for details). Since every word described by B lies in L, we can derive properties of languages described by the factors Fi. For example, if L ⊆ {0, 1}n is a uniform language, that is, every word has the same number of ones, then every language L(Fi) described by a factor Fi must be uniform as well. These properties then allow us to upper bound the number of words described by each factor, and hence, by B.

Lower bounds for explicit languages. We apply our two methods (Results 1 and 2) to several explicit language families. Namely, we prove lower bounds for:

- the binomial language Bn,k = \{w ∈ {0, 1}n : |w|1 = k\},
- the threshold language Tn,k = \{w ∈ {0, 1}n : |w|1 ≥ k\},
- the language D2n of all length 2n Dyck words,
- the divisibility language Ln,p,p of all binary numbers with n bits that are divisible by p,
- the parity language Ln,peven = \{w ∈ Σn : |w|i is even for all i ∈ Σ\} over Σ = \{1, . . . , k\}, and
- the language Pn of all permutations of Σ = \{1, . . . , n\}.

The results are summarized in Table 1, for comparison we also list upper bounds on expression length and DFA size. These DFAs are more or less trivial, and the upper bounds follow from a simple conversion of the DFAs (see Proposition 3.2). All these languages reveal a large gap between DFA size and expression length. The first three bounds follow from the arithmetic bound (Result 1), the other three are derived with the log-product bound (Result 2).

In particular, we answer a question from Ellul et al. [3] asking for the length of optimal expressions for the binomial language Bn,k = \{w ∈ {0, 1}n : |w|1 = k\}, namely for k = nΩ(1), length nΩ(log n) is optimal. In Corollary 5.2 we show a superpolynomial lower bound for all uniform languages that contain sufficiently many words. This yields our lower bound for the language D2n of all length 2n Dyck words. The bound for the threshold language Tn,k is an easy consequence of our bound for the binomial language.

Ellul et al. [3] also asked for lower bounds for the language of all (arbitrary length) binary numbers that are divisible by a number p. So, our lower bound for the divisibility language Ln,p,p answers the “finite variant” of this question. Our lower bound for the parity language Ln,peven naturally generalizes the lower bound from Ellul et al. [3, Thm. 23] for the XOR language Ln,pxor = \{w ∈ {0, 1}n : |w|1 is even\} to non-binary alphabets. The bound for the permutation language Pn was already shown by Molina Lovett and Shallit [8], however, we give an alternative, shorter proof.
We then apply the arithmetic bound to uniform languages (Section 5.1), investigate the blow-up of language operations (intersection and shuffle) for finite languages (Section 5.2) and address limitations (Section 5.3). In Section 6 we prove our log-product bound (Result 2) and show some useful factorization properties of log-product expressions (Section 6.2).

Table 1: Upper and lower bounds on expression length and DFA sizes for several languages.

| language | DFA | rpn, upper bound | rpn, lower bound |
|----------|-----|------------------|------------------|
| $B_{n,k}$ | $O(nk)$ | $O(nk \cdot n^{\log(k+1)})$ | $nk^{\Omega(\log k)}$ [Corollary 5.1] |
| $T_{n,k}$ | $O(nk)$ | $O(nk \cdot n^{\log(k+1)})$ | $nk^{\Omega(\log k)}$ [Corollary 8.2] |
| $D_{2n}$ | $O(n^2)$ | $O(n^{\log n+3})$ | $n^{\Omega(\log n)}$ [Corollary 5.3] |
| $L_{n,p}^{\text{div}}$ | $O(np)$ | $O(np \cdot p^{\log(n/\log p)})$ | $\Omega(n^{-1}p^{\log(n/\log p)-2})$ [Theorem 7.1] |
| $L_{\text{even}}^{n,k}$ | $O(2^n n)$ | $O(k2^n n^k)$ | $n^{(k-2)\log(k)+\Theta(k)}$ [Theorem 7.3] |
| $P_n$ | $O(2^n)$ | $4^n n^{-(4+O(1))}$ | $\Omega(4^n n^{-(3+\log n)/4})$ [Theorem 7.5] |

Table 2: Blow-ups of operations and conversion for finite languages (assuming an alphabet of size at most $n^{O(1)}$).

| conversion | upper bound | lower bound |
|------------|-------------|-------------|
| $\text{RE} \cap \text{RE}, \text{RE} \sqcup \text{RE} \rightarrow \text{RE}$ | $n^{O(\log n)}$ | $n^{\Omega(\log n)}$ [Thm. 5.4] |
| DFA, NFA $\rightarrow$ RE | $n^{\log(n)+O(1)}$ [3, 4] | $n^{\log(n)/4-\Theta(\log\log n)}$ [Cor. 7.2] |

Blow-up of language operations and DFA conversion. A classical question is: by how much can expression length increase when performing operations like intersection, shuffle (also known as interleaving) or complementation? For infinite languages this problem was solved by Gelade and Neven [6] and Gruber and Holzer [5, 22]: The blow-up is exponential for intersection and shuffle, and even double-exponential for complementation. In contrast, for finite languages the blow-up for intersection and shuffle is at most $n^{O(\log n)}$. We show that for some languages this is inevitable: There are finite languages $L_1, L_2$ with expressions of length $O(n)$ such that any expression describing their intersection $L_1 \cap L_2$ or their shuffle $L_1 \sqcup L_2$ requires length $n^{\Omega(\log n)}$. We prove this as a consequence of our lower bound for the binomial language.

Gruber and Johanssen [4] showed that the blow-up of converting a DFA for a finite language into an expression can be as large as $n^{\log(n)/192}$. Ellul et al. [3] showed an upper bound of $n^{\log(n)+O(1)}$ for such conversions (see Proposition 3.1), so the lower bound is tight apart from the constant factor in the exponent. We will improve this factor to $1/4 - o(1)$. An overview of the blow-ups is given in Table 2.

Non-homogeneous and infinite languages. All results mentioned so far are obtained for homogeneous languages (i.e., all words have the same length). For non-homogeneous and even infinite languages we show that their expression length is bounded from below by the expression length of their lower and higher envelopes, that is, their sublanguages of all shortest resp. all longest words, see Lemma 8.1.

By this lemma our lower bounds for the binomial language $B_{n,k}$ and the permutation language $P_n$ carry over to their infinite variants

- $B_{n,k}^+ = \{w \in \{0,1\}^* : |w|_1 \geq k, |w|_0 \geq n-k\}$ and
- $P_n^+ = \{w \in [n]^* : |w|_a \geq 1$ for all $a \in [n]\}$.

This is particularly interesting since the previous known lower bound methods for infinite languages (relying on star height) do not work for these languages.

2.1. Organization

In the next section (Section 3) we recall basic concepts of languages and expressions, show how to transform automata into expressions and introduce monotone arithmetic formulas. In Section 4 we prove our arithmetic bound (Result 1) and discuss some properties. We then apply the arithmetic bound to uniform languages (Section 5.1), investigate the blow-up of language operations (intersection and shuffle) for finite languages (Section 5.2) and address limitations (Section 5.3). In Section 6 we prove our log-product bound (Result 2) and show some useful factorization properties of log-product expressions (Section 6.2).
Applications of the log-product bound are demonstrated in Section 7, namely we show lower bounds for the divisibility language $L_{n,p}^{\text{div}}$ (Section 7.1), the parity language $L_{n,k}^{\text{even}}$ (Section 7.2) and the permutation language $P_{n}$ (Section 7.3). In Section 8 we treat non-homogeneous and infinite languages with the help of envelopes. Finally, we summarize our results and sketch open problems in Section 9.

In order to provide some background regarding the connection between expressions for finite languages and circuit complexity in general, we give an introduction to semirings and circuits in Appendix A. Since these concepts are not actually needed in our proofs, we moved them into the appendix.

3. Preliminaries

Throughout, let $\mathbb{R}_{\geq 0}$ be the set of all nonnegative reals, $\mathbb{N} = \{0, 1, 2, \ldots \}$ the set of all nonnegative integers and $[n] = \{1, 2, \ldots, n\}$ the set of the first $n$ positive integers. The logarithm of base $2$ is denoted by $\log(\cdot)$ and the natural logarithm by $\ln(\cdot)$. We assume the reader to be familiar with basic concepts of regular languages and only recall some central aspects. An introduction can be found, for example, in [23].

3.1. Languages and regular expressions

An alphabet $\Sigma$ is a finite nonempty set of letters, a word $w = w_1w_2\cdots w_n$ over $\Sigma$ is an element of $\Sigma^n$. As customary, $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ is the set of all words over $\Sigma$, including the empty word $\varepsilon$. A language over $\Sigma$ is a set $L \subseteq \Sigma^*$. For a word $w$ its length $|w|$ is the total number of its letters, and for a given letter $a \in \Sigma$ we denote by $|w|_a$ its number of occurrences in $w$. To avoid pathological situations, we assume throughout that our languages satisfy $L \neq \{\varepsilon\}, \emptyset$.

Regular expressions (or just expressions) over an alphabet $\Sigma$ are defined recursively as follows.

- The symbols $\emptyset$ and $\varepsilon$ are expressions, as well as all letters $a \in \Sigma$.
- If $R$ and $R'$ are expressions, then so are $R^*$, $(R \cdot R')$ and $(R + R')$.

Every expression $R$ describes a regular language $L(R)$ as follows.

- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$ for all $a \in \Sigma$.
- $L(R^*) = (L(R))^*$, $L(R \cdot R') = L(R) \cdot L(R')$ and $L(R + R') = L(R) \cup L(R')$.

As customary, we abbreviate concatenations $R \cdot R'$ by $RR'$ and omit parenthesis where possible. Two expressions $R$ and $R'$ are equivalent, denoted by $R \equiv R'$, if they describe the same language. With a slight abuse of notation, we will sometimes identify expressions with their languages, e.g., $(a+b)$ stands for $\{a,b\}$. We assume throughout and w.l.o.g that no expression contains the symbol $\emptyset$ and that no expression for a finite language contains a star. Furthermore, we identify every expression with its syntax tree. For a node $u$ in the syntax tree of an expression $R$, the subtree of $u$ is a subexpression of $R$, denoted by $R_u$. There are several measures for the length of an expression. In this paper, we use the reverse polish length $\text{rpl}(R)$ which is the number of nodes in the syntax tree of $R$; for an overview of other length measures see [3]. For a regular language $L$ its expression length $\text{rpl}(L)$ is the length of a shortest expression that describes $L$.

We will mainly deal with homogeneous languages. A language $L \subseteq \Sigma^*$ is homogeneous if all words have the same length, i.e., $L \subseteq \Sigma^n$ holds for some $n \in \mathbb{N}$. An expression $R$ is homogeneous if $R$ describes a homogeneous language. Since we do not allow the symbol $\emptyset$, every subexpression of a homogeneous expression is homogeneous itself. The degree of $R$, denoted by $\deg R$, is the length of its described words, and the degree of a node $u$ in the syntax tree of $R$ is the degree of its subexpression $R_u$. For a language $L \subseteq \Sigma^*$ we call the homogeneous language $L \cap \Sigma^n$ its $n$-slice.

3.2. Conversions of automata

To bring our lower bounds in context, we compare them with upper bounds as well as with automata for the respective languages. For infinite languages, the conversion of an NFA into an expression can cause an exponential blow-up [2, 6, 5]. For finite languages, however, this blow-up is at most quasi-polynomial, as the following proposition tells. We refer to [24] for a comprehensive survey on conversions. All results in this subsection given for NFAs hold analogously for DFAs.
Proposition 3.1 (Conversion for finite languages, [3, Cor. 22] or [4, Cor. 13]). Let $L$ be a finite language over a $k$-letter alphabet accepted by an NFA with $s$ states. Then $L$ can be described by regular expressions of length

$$\text{rpn}(L) \leq O(k^n s_{\log s}).$$

This conversion can be refined for the subclass of “layered” automata, and in particular for homogeneous languages. Call an NFA for a finite language layered if there is a partition $Q_0 \cup Q_1 \cup \cdots \cup Q_n$ of its states such that every word of length $j$ leads to a state $q \in Q_j$ when given into the automaton. We call $Q_j$ the $j$-th layer. Note that in a layered automaton only transitions between neighboring layers $Q_j$ and $Q_{j+1}$ are allowed. For a layered automaton define its width as $\omega = \max_{0 \leq j \leq n} |Q_j|$ and its length as the length $n$ of its longest path.

Every NFA\textsuperscript{1} for a homogeneous language is layered since every word starting from its initial state to a final state has length exactly $n$; and in this case one final state always suffices. In particular, given an NFA $A$ with $s$ states for a language $L$, we obtain a layered NFA for its $n$-slice $L \cap \Sigma^n$ by constructing the product automaton of $A$ and the trivial DFA for $\Sigma^n$; we call it the $n$-slice NFA of $A$\textsuperscript{2}. This automaton has at most $sn$ states, width $s$ and length $n$.

The following proposition tells us how to transform a layered automaton into an expression.\textsuperscript{3}

Proposition 3.2 (Conversion of layered automata). Let $L$ be a finite language over a $k$-letter alphabet accepted by a layered NFA of width $\omega$ and length $n$ with $f$ final states. Then $L$ can be described by regular expressions of length

$$\text{rpn}(L) \leq O(f n \omega \cdot \omega_{\log n}).$$

Note the difference in the bounds given by the two propositions above: The number of states of a layered automaton of width $\omega$ and length $n$ is at least $\omega + n$ and can be as large as roughly $\omega n$. So, for example, if the width $\omega$ is constant, then Proposition 3.2 yields a polynomial bound $kn^{O(1)}$ while Proposition 3.1 yields only a superpolynomial bound $kn^{O(\log n)}$.

**Proof.** We use a standard idea which can be found, for example, in [2] or [3]. Let $A$ be an NFA for $L$, let $Q = Q_0 \cup Q_1 \cup \cdots \cup Q_n$ be its set of states, $i \in Q_0$ be its initial state and $F \subseteq Q$ be its set of final states. We recursively construct an expression $R_d^{x \rightarrow y}$ that describes all words of length $d$ that lead from state $x$ to state $y$. For $\ell \in \mathbb{N}$ and $x \in Q_\ell$ define

$$R_d^{x \rightarrow y} = \sum_{q \in Q_{\ell + \lfloor d/2 \rfloor}} R_{\lfloor d/2 \rfloor}^{x \rightarrow q} \cdot R_{\lfloor d/2 \rfloor}^{q \rightarrow y} \quad (1)$$

for $d \geq 2$, while $R_1^{x \rightarrow y}$ is given directly by the transitions of the automaton. Finally, the expression

$$R := \sum_{j=0}^{n} \sum_{q \in F \cap Q_j} R_j^{x \rightarrow q}$$

describes the language $L$.

For an expression $R'$ let $\text{leaf}(R')$ be the number of leaves in the syntax tree of $R'$, and let $T(d) = \max_{x,y} (\text{leaf}(R_d^{x \rightarrow y}))$. Then Eq. (1) gives the recursion

$$T(d) \leq \omega \cdot \big( T(\lfloor d/2 \rfloor) + T(\lceil d/2 \rceil) \big)$$

for $d \geq 2$, and $T(1) \leq k$. Solving this recursion leads to $T(d) \leq O(kd \omega \cdot \omega_{\log d})$ (see [3, Lem. 21]). Clearly, $\text{rpn}(R) = O(\text{leaf}(R))$ holds and we get $\text{rpn}(R) \leq O(f \cdot T(n)) = O(f n \omega \cdot \omega_{\log n})$. \hfill $\square$

**Example 2** (Divisibility language). Let $p$ be an odd integer and consider the language $L_{d,p} = \{0, 1\}^*$ of all binary numbers that are divisible by $p$. This language can be accepted by a DFA with states $Q = \{0, 1, \ldots, p-1\}$, initial and final state 0, and transitions $\delta(q, a) = (2q + a) \mod p$ for all $q \in Q$ and $a \in \{0, 1\}$. The $n$-slice of this language is $L_{n,p}^d = L \cap \{0, 1\}^n$ and the $n$-slice DFA has width $p$ and length $n$. Thus, Proposition 3.2 implies that $L_{d,p}^n$ has expressions of length $O(np \cdot p_{\log n})$. In Section 7.1 we will improve this upper bound to $O(np \cdot p_{\log(n/p)})$ and give an almost matching lower bound.

\textsuperscript{1}We assume that there are no “useless” states from which no path to a final state exists.

\textsuperscript{2}We implicitly collapse equivalent states; in particular, any slice automaton has only one initial and one final state.

\textsuperscript{3}In fact, a layered automaton is nothing else than a layered branching program over the free semiring $(+, \cdot)$ (see Appendix A) and our conversion follows the standard simulation of branching programs by formulas.
3.3. Monotone arithmetic formulas

In this subsection we briefly treat monotone arithmetic formulas. A more detailed exposition of monotone arithmetic circuits is given in Appendix A.

Given a number \( n \in \mathbb{N} \), a monotone arithmetic formula of the variables \( x_1, \ldots, x_n \) is a rooted tree with leaves holding either one of the variables \( x_i \) or a constant \( c \in \mathbb{R}_{\geq 0} \). Every inner node (a gate) performs one of the operations addition (\(+\)) or multiplication (\(\times\)). The size of a formula is the number of its nodes. Every formula computes a polynomial

\[
f(x_1, \ldots, x_n) = \sum_{a \in A} \lambda_a \prod_{i=1}^{n} x_i^{a_i}
\]

over \( \mathbb{R}_{\geq 0} \) in a natural manner, where \( A \subseteq \mathbb{N}^n \) is its (finite) set of exponent vectors and \( \lambda_a \in \mathbb{R}_{\geq 0} \) are positive coefficients. We say that the formula produces the set \( A \).

For a given set \( A \subseteq \mathbb{N}^n \) denote by \( \text{arith}_R(A) \) the size of a smallest formula that produces \( A \), i.e., that computes a polynomial whose set of exponent vectors is \( A \). Note that we do not require the formula to compute a polynomial with specific coefficients, we are only interested in its set of monomials. This is in agreement with almost all lower bounds shown for monotone arithmetic circuit (or formula) complexity (see, e.g., [25, Rem. 1] or [26]). We identify every vector \((w_1, w_2, \ldots, w_n) \in \mathbb{N}^n \) with the word \( w_1 w_2 \cdots w_n \). Thus, the produced set \( A \subseteq \mathbb{N}^n \) can be interpreted as a homogeneous language over the alphabet \( \Sigma = \{0, 1, \ldots, k\} \), with \( k \) being the largest entry of a vector in \( A \). In particular, if \( A \subseteq \{0, 1\}^n \) (i.e., \( A \) is produced by a multilinear formula), then \( A \) is a language over \( \Sigma = \{0, 1\} \).

4. Reducing expression length to monotone arithmetic formula size

Let \( n \geq 1 \), \( L \subseteq \{0, 1\}^n \) be a homogeneous language and \( R \) be a homogeneous expression describing \( L \). In this section, we assume w.l.o.g. that \( R \) does not contain the symbol \( \varepsilon \) (in addition to not containing the symbol \( \emptyset \)); hence, every leaf in \( R \) is a letter. Ellul et al. [3, Lem. 24] transformed \( R \) into a boolean formula for the function \( f_L: \{0, 1\}^n \to \{0, 1\} \) such that \( f_L(x) = 1 \) iff \( x \in L \). Namely, they assigned a unique position \( i \in [n] \) to each leaf of \( R \), such that its letter occurs as the \( i \)-th letter in all words in \( L \). For example, in the expression \((a+b)(cd+ce)\) the positions of the leaves holding the letters \( a, b, c, d, e \) are 1, 1, 2, 3, 3, respectively. The transformation is as follows: Replace each union by OR, each concatenation by AND, and each leaf at position \( i \) holding the letter 1 (resp. 0) by the literal \( x_i \) (resp. \( \neg x_i \)). We present a similar transformation of \( R \) into a monotone arithmetic formula that produces the set \( L \).

The arithmetic version of \( R \) is the monotone arithmetic formula \( \Phi_R \) of the variables \( x_1, \ldots, x_n \) that is constructed as follows: Replace each union node (\(+\)) of \( R \) by an arithmetic addition gate (\(+\)) and each concatenation node (\(\cdot\)) by an arithmetic multiplication gate (\(\times\)). Replace each leaf holding the letter 0 by the constant 1 and replace each leaf holding the letter 1 at position \( i \) by the variable \( x_i \). Note that in any case a leaf holding a letter \( \sigma \in \{0, 1\} \) at position \( i \) is replaced by \( x_i^\sigma \). For example, the arithmetic version of the expression \( 000+011+100 \) is the formula \( 1+x_2x_3+x_1 \), which produces the set \( \{(0, 0, 0), (0, 1, 1), (1, 0, 0)\} \).

**Lemma 4.1.** Let \( R \) be a homogeneous expression with \( L(R) \subseteq \{0, 1\}^n \) and \( \Phi_R \) be its arithmetic version. Then \( \Phi_R \) has size at most \( \text{rpn}(R) \) and produces the set \( L(R) \).

That is, the arithmetic formula \( \Phi_R \) computes a polynomial \( \sum_{a \in A} \lambda_a \prod_{i=1}^{n} x_i^{a_i} \) with \( A = L(R) \) and some coefficients \( \lambda_a > 0 \).

**Proof.** The claim for size is trivial. To show the claim for the produced set, we first bring both \( R \) and \( \Phi_R \) in “sum-product normal form”, that is, we move all union or addition nodes to the top by iteratively applying the distributive law: replace \( u \cdot (v + w) \) by \( uw + uw \) and \( (u + v) \cdot w \) by \( uw + vw \). By this procedure, the language \( L(R) \) described by \( R \) and both the computed polynomial and the produced set of \( \Phi_R \) do not change. Hence, it suffices to show the claim for these modified versions of \( R \) and \( \Phi_R \).

For the rest of the proof assume that all union nodes in \( R \) and all addition gates in \( \Phi_R \) are at the top. Call a subexpression of \( R \) describing a single word of length \( n \) a singleton. Hence, the expression \( R \) is a
union over all words \( w \in L(R) \) (possibly with repetitions) of singletons, each describing one of the words \( w \).

By construction of \( \Phi_R \), every singleton describing \( w = w_1 \cdots w_n \) is replaced by a subformula that computes the monomial \( x^w := x_1^{w_1} \cdots x_n^{w_n} \). The entire formula \( \Phi_R \) is a sum over all monomials \( x^w \) for \( w \in L(R) \), that is, it computes the polynomial \( f(x) = \sum_{w \in L(R)} \lambda_w x^w \), where the constant \( \lambda_w > 0 \) denotes the number of singletons in \( R \) that describe the same word \( w \). Hence, \( \Phi_R \) produces \( L(R) \), as desired.

Recall that \( \text{arith}_F(L) \) is the size of a smallest monotone arithmetic formula that produces \( L \). Lemma 4.1 directly yields the following theorem.

**Theorem 4.2 (Arithmetic bound).** Let \( L \subseteq \{0,1\}^n \) be a homogeneous language. Then any regular expression describing \( L \) has length at least 

\[
\text{rpn}(L) \geq \text{arith}_F(L).
\]

We make some remarks before turning to the applications.

**Remark 1.** Note that lower bounds on the size of a monotone arithmetic formula yield bounds for two different related languages. As a simple example, take the monotone arithmetic formula \( f = 1 + x_2 x_3 + x_1 \). Then lower bounds on the formula size of \( f \) carry over to the expression length of two different languages \( L \) and \( L' \), namely

- the language \( L = \{ \varepsilon, x_2 x_3, x_1 \} \) of all monomials of \( f \) over the alphabet \( \Sigma = \{x_1, x_2, x_3\} \), and
- the language \( L' = \{000, 011, 100\} \) of all exponent vectors of \( f \) over the alphabet \( \Sigma = \{0,1\} \).

The first claim is long known, see for example [9], and we already mentioned it in Section 1.1. The second claim is our arithmetic bound Theorem 4.2.

**Remark 2 (Invariance under permutations).** Since arithmetic operations \(+\) and \(\times\) are commutative, the order of the variables does not matter, i.e., reordering the variables \( x_1, \ldots, x_n \) in a polynomial does not change its arithmetic complexity. For a language \( L \subseteq \{0,1\}^n \) and a permutation \( \sigma : [n] \rightarrow [n] \) define \( \sigma(L) := \{ w_{\sigma(1)} \cdots w_{\sigma(n)} : w_1 \cdots w_n \in L \} \). Then \( \text{arith}_F(L) = \text{arith}_F(\sigma(L)) \) holds, and all bounds shown with Theorem 4.2 for \( L \) also hold for the permuted language \( \sigma(L) \).

On the other hand, this is a nice feature; on the other, it limits the possibilities for application. In Section 5.3 we address this issue in detail.

**Remark 3 (Arithmetic vs. boolean complexity).** Every language \( L \subseteq \{0,1\}^n \) also defines a monotone boolean function \( f_L^{\text{mon}} : \{0,1\}^n \rightarrow \{0,1\} \) with \( f_L^{\text{mon}}(x) = 1 \) iff there is a word \( y \in L \) such that \( x \geq y \) holds componentwise. Let \( \text{bool}_F(L) \) be the size of a smallest monotone boolean formula that computes the function \( f_L^{\text{mon}} \). Gruber and Johannsen [4] showed that the expression length of a (monotone) language \( L \subseteq \{0,1\}^n \) is bounded from below by the monotone boolean formula size of \( f_L^{\text{mon}} \), that is, they showed \( \text{rpn}(L) \geq \text{bool}_F(L) \) (see Section 1.1).

Theorem 4.2 strictly improves on this bound. It is well known that

\[
\text{bool}_F(L) \leq \text{arith}_F(L)
\]

holds for every language \( L \) (see, e.g., [27]). We give an example for an exponential gap between these complexities which is due to Jukna [28].

Consider the complete bipartite \( n \times n \) graph \( K_{n,n} \) and let \( U, V \) be its two sets of vertices. A quasi matching is a subgraph of \( K_{n,n} \) that is constructed by picking an incident edge for each vertex \( u \in U \) and each vertex \( v \in V \) and then take the union of these two sets. Let \( \text{QM}_n \) be the set of all quasi matchings, each viewed as characteristic vector of its set of edges. Then the lower bound \( \text{arith}_F(\text{QM}_n) = 2^\Omega(n) \) is known [28, Sect. 3.1] (this holds even for circuits). On the other hand, we have \( \text{bool}_F(\text{QM}_n) \leq O(n^2) \): For an edge \( (u,v) \in U \times V \) let \( x_{u,v} \) be the corresponding variable. Then the boolean function corresponding to \( \text{QM}_n \) can be computed by the following monotone boolean formula of size \( O(n^2) \).

\[
f_{\text{QM}_n}^{\text{mon}}(x) = \left( \bigwedge_{u \in U} \bigvee_{v' \in V} x_{u,v'} \right) \land \left( \bigwedge_{v \in V} \bigvee_{u' \in U} x_{u',v} \right)
\]
Proposition 3.2

Theorem 4.2

Remark 4 (Formula vs. circuit complexity). Let us emphasize the use of formula size over circuit size in Theorem 4.2. For a language $L \subseteq \{0,1\}^n$ let $\text{arith}(L)$ denote the size of a smallest monotone arithmetic circuit producing $L$; for a proper introduction of circuits see Appendix A. Since every formula is a circuit, the inequality stated in Theorem 4.2 still holds if we replace formula size $\text{arith}(L)$ by circuit size $\text{arith}_c(L)$. However, with circuit size it is not possible to show superpolynomial blow-ups between finite automata and expressions since automata can be simulated by circuits.

Let $L \subseteq \{0,1\}^n$ be a language that can be accepted by a layered NFA (as defined in Section 3.2). From this NFA we obtain a monotone algebraic branching program (MABP) of same size via the following transformation: Replace all edges between two neighboring layers $Q_{j-1}$ and $Q_j$ labeled with the letter 0 (resp. the letter 1) by the constant 1 (resp. by the variable $x_i$). The resulting MABP then describes a polynomial $f(x) = \sum_{a \in L} a_0 \prod_{i=1}^k x_i^{a_i}$, with $L$ as set of exponent vectors. By a standard construction, this MABP can be transformed into a monotone arithmetic circuit of polynomial size (see, e.g., [29]). Thus, if a language $L$ can be described by small NFAs, then the monotone arithmetic circuit complexity of $L$ is also small.

5. Applications and limits of the arithmetic bound

5.1. Uniform languages

Recall that a language $L \subseteq \{0,1\}^n$ is uniform if all words in $L$ have the same number of ones. The most basic uniform language is the binomial language

$$B_{n,k} = \{ w \in \{0,1\}^n : |w|_1 = k \}$$

which was investigated by Ellul et al. [3]. They constructed an expression of length $n^{O(\log k)}$ (resp. length $O(n \log^k(n))$ if $k$ is a constant) and asked whether its length is optimal. Recently, Mousavi [7] showed the optimality for $k \leq 3$ by analyzing a linear program derived from the language. We show that the length is asymptotically optimal also for $k = n^{\Theta(1)}$, using a lower bound shown by Hrubeš and Yehudayoff [13] for the corresponding arithmetic formula complexity. Note that the binomial language can be accepted by a DFA of width $k+1$ and length $n$ (see Fig. 1), and thus, also Proposition 3.2 implies an upper bound of $\text{rpn}(B_{n,k}) \leq O(k n^{1+\log(k+1)})$.

Corollary 5.1 (Binomial language). Let $n, k \in \mathbb{N}$ with $k \leq n/2$. Then the binomial language $B_{n,k}$ requires regular expressions of length $\text{rpn}(B_{n,k}) \geq n k^{\Omega(\log k)}$.

Proof. The elementary symmetric polynomial $f_{n,k} = f_{n,k}(x_1,\ldots,x_n) = \sum_{1 \leq i_1 < \ldots < i_k} x_{i_1} x_{i_2} \cdots x_{i_k}$ has exactly the words of the binomial language $B_{n,k}$ as its exponent vectors. In [13, Thm. 1 (a)] Hrubeš and Yehudayoff showed that any monotone arithmetic formula computing a polynomial with the same set of exponent vectors as $f_{n,k}$ has size at least $n k^{\Omega(\log k)}$. By Theorem 4.2 the same bound holds for the expression length of $B_{n,k}$. □

Remark 5. The corresponding monotone boolean function to $B_{n,k}$ is the threshold function $\tau_{n,k} : \{0,1\}^n \to \{0,1\}$ that outputs 1 iff there are at least $k$ ones among $x_1,\ldots,x_n$. Valiant [30] has shown that this function can be computed by monotone boolean formulas of size $O(n^{5.3})$. So, by the boolean methods (see Section 1.1) only lower bounds of polynomial size can be obtained for $B_{n,k}$. 10
Figure 2: A (partial) DFA accepting the language of all Dyck words of height at most $h$. Since in the $2n$-slice $D_{2n}$, the height of every word is at most $n$, if we set $h = n$, then the associated $2n$-slice DFA accepts the language $D_{2n}$ and has width $n + 1$ and length $2n$. So, the upper bound $\text{rpn}(D_{2n}) \leq O(n^{3+\log n})$ follows from Proposition 3.2.

The following corollary tells us, that any uniform language requires long expressions, provided that it contains sufficiently many words.

**Corollary 5.2 (Uniform languages).** Let $k \leq \lfloor n/2 \rfloor$ and $L \subseteq B_{n,k}$ be a uniform language. Then $L$ requires regular expressions of length $\text{rpn}(L) \geq nk^{\Omega(\log k)} \cdot |L|/\binom{n}{k}$.

**Proof.** Hrubý and Yehudayoff [13, Prop. 7] showed that any polynomial $g$ with a set $A_g \subseteq B_{n,k}$ of exponent vectors requires monotone arithmetic formulas of size at least $nk^{\Omega(\log k)} \cdot |A_g|/\binom{n}{k}$, where $|A_g|$ is the number of monomials in $g$. Since each monomial of $g$ corresponds to an exponent vector in $A_g$ and, hence, to a word in $L$, the claim follows by Theorem 4.2.

A Dyck word is a word $w \in \{0,1\}^*$ such that $w$ has the same number of zeros and ones, and every prefix of $w$ contains not more ones than zeros. The language $D$ of all Dyck words can be defined recursively as follows:

- $\varepsilon \in D$, and
- if $v$ and $w$ are in $D$, then so are $vw$ and $0v1$.

Usually, $D$ is interpreted as the language of all correctly nested sequences of brackets, with 0 representing opening and 1 representing closing brackets. It is well known that $D$ is context-free, but not regular. However, if we restrict the length or the height of the Dyck words, the language turns regular. (The height of a Dyck word $w$ is the minimal number $n$ such that $|w'|_0 - |w'|_1 \leq n$ holds for every prefix $w'$ of $w$.)

For an integer $n$ consider the $2n$-slice $D_{2n} := D \cap \{0,1\}^{2n}$ consisting of all Dyck words of length $2n$. The language $D_{2n}$ can be accepted by a DFA with $O(n^2)$ states, see Fig. 2. In contrast it has no short expressions.

**Corollary 5.3 (Dyck language).** The language $D_{2n}$ requires regular expressions of length $\text{rpn}(D_{2n}) \geq n^{\Omega(\log n)}$.

**Proof.** Clearly $D_{2n} \subseteq B_{2n,n}$ holds, since every word in $D_{2n}$ contains the same number of zeros and ones. It is well known that $D_{2n}$ contains exactly $\binom{2n}{n}$ words – this number is known as the Catalan number – see, for example, [31] for an elegant proof. Thus, by Corollary 5.2, we have $\text{rpn}(D_{2n}) \geq \frac{2n}{n+1} \cdot n^{\Omega(\log n)}$.

### 5.2. Blow-up of language operations

A classical question is to determine by how much expression length can increase when performing operations like complementation, intersection or shuffle. The situation for infinite languages has been resolved by Gelade and Neven [6] resp. Gruber and Holzer [5, 22]; the blow-up is exponential for shuffle and intersection and double-exponential for complementation. For finite languages, however, the blow-up of intersection and shuffle is at most $n^{O(\log n)}$, this can be shown as follows. Given two expressions of lengths $m_1$ resp. $m_2$ describing finite languages, transform them into NFAs. Then construct the corresponding (intersection or shuffle) product automaton. For either of the two operations, this gives an NFA with $n = O(m_1m_2)$ states. Finally, translate this NFA back into an expression. Since its accepted language is finite, length $n^{O(\log n)}$ suffices according to Proposition 3.1.

We now give matching lower bounds, thereby showing that this construction cannot be substantially improved. Recall the definition of the shuffle operation $\uplus$ (also called interleaving): For two words $v, w$, their shuffle $v \uplus w$ is the set of all words of the form $v_1w_1v_2w_2\cdots v_kw_k$ where $k \in \mathbb{N}$, $v_i, w_i \in \Sigma^*$ for all $i$ and $v_1v_2\cdots v_k = v$ and $w_1w_2\cdots w_k = w$. The shuffle of two languages is $L_1 \uplus L_2 = \bigcup_{v \in L_1, w \in L_2} v \uplus w$. 

11
Corollary 5.1. We already mentioned in Lemma 4.1 that arithmetic complexity ignores the order of the variables. This prevents us from proving bounds for languages L that have a permutation σ such that σ(L) has short expressions. Take, for example, the language L = L_{2n}^{pal} of all palindromes over \{0, 1\} of length 2n. The fooling set method gives an exponential lower bound rpn(L) ≥ \Omega(2^n) (see Example 1). However, reordering the letters yields the language L' := \{w_1 w_2 w_2 w_2 2n-1 \ldots w_n w_2 \} with rpn(L') = O(n). According to Remark 2, lower bounds obtained by Theorem 4.2 are the same for rpn(L) and rpn(L'), and, thus, are at most linear. In other words, arith(L) ≤ O(n), but rpn(L) ≥ \Omega(2^n). The same problem arises for the boolean methods [3, 4] (see Section 1.1), Gruber and Johannsen [4] actually presented the same example.

More subtle problems of “ignored non-commutativity” can occur even for languages that are invariant under permutations (like the binomial language): If we transform an expression R into its arithmetic version \Phi_R (as described before Lemma 4.1), this formula \Phi_R has a special structure: For every multiplication gate u \cdot v, the variables of the gates u and v must be “consecutive” in that u contains only variables from the set \{x_k, \ldots, x_j\} and v contains only variables from the set \{x_{j+1}, \ldots, x_\ell\} for some k ≤ j ≤ \ell. In contrast, an arbitrary monotone arithmetic formula (not derived from an expression) can have multiplications of arbitrary sets of variables, e.g., (x_1 + x_3) \cdot (x_2 + x_4) is possible.

Finally, the arithmetic bound only works for languages over the alphabet \{0, 1\}. In the next section, we circumvent these issues by translating a lower bound method from arithmetic formula complexity directly to regular expression length.

6. Direct lower bounds

Until now we only used already existing bounds from arithmetic formula complexity. But how can such bounds be obtained? One possibility is to lower bound the circuit depth. By standard balancing arguments a lower bound \(2 \Omega^{(\text{depth})}\) on formula size follows. Shamir and Snir [32] and Tiwari and Tompa [33] developed techniques for such depth bounds, Jukna [27] put these into a general framework. Another option is the lower bound method for the size of monotone (or even multilinear) arithmetic formulas by Shpilka and
Yehudayoff [20] resp. Hrubč and Yehudayoff [13] by so-called log-product polynomials. With this method Hrubč and Yehudayoff showed their lower bound for the elementary symmetric polynomial that we used in Corollary 5.1 for the binomial language.

So, in order to obtain a lower bound on the expression length of a given language \( L \), one can lower bound the monotone arithmetic formula size of \( L \) with one of the above methods and apply the arithmetic bound (Theorem 4.2). However, we here take a different approach: we translate the “log-product method” from [20, 13] directly to expression length. Thereby we solve the problem of “ignored non-commutativity” discussed in Section 5.3 and can use arbitrary alphabets. A similar translation of lower bound methods to the non-commutative world was done in [10, 11] (see also [12] and [34]), the difference is that we here do it for formulas instead of circuits.

### 6.1. The log-product bound

In the following, \( \Sigma \) is an arbitrary alphabet, \( L \subseteq \Sigma^n \) a homogeneous language and \( R \) a homogeneous expression describing \( L \). The high-level idea for lower bounding \( \text{rpn}(L) \) is roughly as follows.

- Write \( R \) as union \( B_1 + \cdots + B_\ell \) of “log-product” expressions \( B_i \), where \( \ell \leq \text{rpn}(R) \) and every log-product expression \( B_i \) can be factorized into \( m \geq \log n \) nontrivial factors \( B_i \equiv F_1 F_2 \cdots F_m \).
- From \( L \) derive structural properties that any language described by a factor \( F_j \) must have.
- Upper bound the number of words in any language with these properties to obtain an upper bound on \( |L(B_i)| \).

Recall that the degree \( \deg R \) of a homogeneous expression \( R \) is the length of its described words.

**Definition 6.1 (Log-product).** A homogeneous expression \( B \) is log-product\(^4\), if

- \( B \) is a letter, or
- if there are homogeneous expressions \( B_1, B_2 \) such that \( B_1 \) is log-product itself, \( \deg B_1 \geq \deg B_2 \) and \( B = B_1 B_2 \) or \( B = B_2 B_1 \).

In other words, any single letter is log-product, and if an expression \( B_1 \) is log-product, then so are \( B_1 B_2 \) and \( B_2 B_1 \) for any homogeneous expression \( B_2 \) with \( \deg B_2 \leq \deg B_1 \). For example, the expression \( (00 + 11)(00 + 11) \) is log-product, but the expression \( (00 + 11)(00 + 11) \) is not. In Section 6.2 we will investigate some useful properties of log-product expressions.

The following lemma is a straightforward adaption of [13, Lem. 4] or [20, Lem. 3.5].

**Lemma 6.1.** Let \( R \neq \varepsilon \) be a homogeneous expression. Then there exist \( \ell \leq \text{rpn}(R) \) log-product expressions \( B_1, \ldots, B_\ell \) such that \( R \equiv B_1 + \cdots + B_\ell \).

**Proof.** We proceed by induction on \( R \). If \( R \) is a single letter, the claim is trivial. If \( R = R_1 + R_2 \) is a union, we can apply the induction hypothesis to both \( R_1 \) and \( R_2 \), and are finished. Finally, let \( R = R_1 R_2 \) be a concatenation. Assume \( \deg R_1 \geq \deg R_2 \), the other case is analogous. By induction hypothesis there are log-product expressions \( B_1, \ldots, B_\ell \) such that \( R_1 \equiv B_1 + \cdots + B_\ell \) for an \( \ell \leq \text{rpn}(R_1) \). Since \( \deg B_i = \deg R_1 \geq \deg R_2 \) holds for all \( i \), every expression \( B_i R_2 \) is also log-product. So, \( R \equiv B_1 R_2 + \cdots + B_\ell R_2 \) is a union of \( \ell \leq \text{rpn}(R_1) \leq \text{rpn}(R) \) log-product expressions, as desired.

**Remark 6.** Lemma 6.1 still holds if we extend regular expressions by a squaring operation \( (\cdot)^2 \) defined by \( L(R^2) := (L(R))^2 \) as introduced in [35]; see also [36] for a more recent overview. To show this, proceed analogously to the case when \( R \) is a concatenation of two identical subexpressions \( R = R' \cdot R' \).

From Lemma 6.1 our second lower bound method follows.

\(^4\)In the preliminary version of this paper [1] we used the term “balanced” instead of “log-product”, and used a slightly different definition.
Lemma 6.1, then for the expression $\text{Fig. 3}$ Theorem 6.2 Theorem 6.2 in the following way. Let $\Sigma$ be a homogeneous language and $h \in \mathbb{R}_{\geq 0}$. If $|L(B)| \leq h$ holds for every log-product expression $B$ with $L(B) \subseteq L$, then any expression for $L$ has length $\text{rpn}(L) \geq |L|/h$.

Proof. Let $R$ be an expression of length $\ell$ for $L$ and assume $|L(B)| \leq h$ holds for all log-product expressions $B$ with $L(B) \subseteq L$. By Lemma 6.1 the expression $R$ can be written as union of at most $\ell$ log-product expressions. At least $\ell \geq |L|/h$ such expressions are necessary to describe all words in $L$. \hfill $\square$

Remark 7 (Languages with weightings). It is possible to generalize Theorem 6.2 in the following way. Let $\lambda : L \rightarrow \mathbb{R}_{\geq 0}$ be a weighting of the words in $L$ and let $\lambda(L') := \sum_{w \in L'} \lambda(w)$ for every language $L' \subseteq L$. In Theorem 6.2, we can replace $|\cdot|$ by $\lambda(\cdot)$: If $\lambda(L(B)) \leq h$ holds for every log-product expression $B$ with $L(B) \subseteq L$, then $\text{rpn}(L) \geq \lambda(L)/h$ follows. However, in this paper we will not use this generalization.

Remark 8 (Sublanguages). If a lower bound $\text{rpn}(L) \geq \ell$ for a language $L$ is shown by Theorem 6.2, then for every sublanguage $L' \subseteq L$ a lower bound of $\text{rpn}(L') \geq \ell \cdot |L'|/|L|$ follows.

6.2. Factorizations of log-product expressions

We take a closer look on log-product expressions. Given a log-product expression $B$ of degree $\deg B = n$, we can construct a path from the root to a leaf by always continuing with the child whose subexpression is log-product and that has larger degree (i.e., $B_1$ in Definition 6.1).\footnote{It may happen that this node is not unique. In this case, choose any of the nodes.} We call this path the canonical path of $B$; note that all inner nodes on this path are concatenation nodes and that all their subexpressions are log-product. From the canonical path we obtain a factorization $B = F_1F_2\cdots F_m$ of $B$ with $m \geq \log n$ factors $F_i$ being some homogeneous expressions, namely the subexpressions of the siblings of the nodes in the canonical path in $B$, plus the last node in this path; see Fig. 3 for an example. Note that for any factor $F_i$ of $B$ there are words $x$ and $y$ such that $L(xF_1y) \subseteq L(B)$.

In many cases, factorizations with additional restrictions on the degree of the factors can be useful. Hrubıš and Yehudayoff [13] used “balanced” factorizations of log-product polynomials where each factor $F_i$ has a degree lying between $(1/3)^{i} n$ and $(2/3)^i n$. We use factorizations where most factors have “sufficiently large” degree.

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node (root) at (0,2) {$\Sigma$};
\node (a) at (-1,1) {$a$};
\node (b) at (1,1) {$b$};
\node (c) at (0,0) {$c$};
\node (d) at (-1.5,-1) {$a \cdot (b + c)$};
\node (e) at (1.5,-1) {$a + b$};
\node (f) at (0,-2) {$a + c$};
\draw (root) -- (a);
\draw (root) -- (b);
\draw (root) -- (c);
\draw (a) -- (d);
\draw (b) -- (e);
\draw (c) -- (f);
\end{tikzpicture}
\caption{A log-product expression $B = (a \cdot (b + c)) \cdot ((b \cdot (a + c)) \cdot (a + b))$ and its canonical path (shaded in orange). $B$ has the factorization $B = F_1F_2F_3F_4$ where the factors are $F_1 = (a \cdot (b + c))$, $F_2 = b$, $F_3 = (a + c)$ and $F_4 = (a + b)$.}
\end{figure}
Proposition 6.3 ($\gamma$-factorization). Let $B$ be a log-product expression of degree $n$ and let $\gamma \geq 1$. Then there exist $m \geq \log(1+n/\gamma)$ and $2m$ homogeneous expressions $P_1, \ldots, P_m, S_1, \ldots, S_m$ such that $B \equiv P_1 P_2 \cdots P_m S_m S_{m-1} \cdots S_1$ and $\deg P_i + \deg S_i \geq \gamma$ holds for all $i \in [m-1]$, and $\deg P_m + \deg S_m \leq \gamma$.

We call $P_1 \cdots P_m S_m \cdots S_1$ a $\gamma$-factorization. Note that trivial factors $P_i = \varepsilon$ or $S_i = \varepsilon$ are allowed.

Proof. We proceed by induction. For $n \leq \gamma$ the claim is trivial, so assume $n > \gamma$. To obtain a $\gamma$-factorization of $B$ initialize $S = \varepsilon$ and $P = \varepsilon$ and follow the canonical path downwards, starting at the root of $B$. For each node $v = x \cdot y$ passed by, update either $P$ or $S$: if we went to the right child $y$, set $P := P \cdot B_x$, if we went to the left child $x$, set $S := B_y \cdot S$. By this procedure the invariant $B \equiv PB_n S$ holds in each step, where $v$ is the currently reached node. Eventually, we arrive at some node $u$ whose degree $n':=\deg B_u$ satisfies $(n-\gamma)/2 < n' \leq n - \gamma$. This is the case just because the degree cannot drop by more than a factor of two at each step. Since $B_u$ is log-product, by induction hypothesis there is a $\gamma$-factorization $B_u \equiv P_1 \cdots P_{m'} S_{m'} \cdots S_1$ with $m' \geq \log(1+n'/\gamma)$. We claim that $PP_1 \cdots P_{m'} S_{m'} \cdots S_1 S$ is a $\gamma$-factorization of $B$ with $2(m'+1)$ factors: since $n' \leq n - \gamma$, we have $\deg P + \deg S = n - n' \geq \gamma$, and since $n' > (n-\gamma)/2$, we have $m' = m' + 1 \geq \log(1+n'/\gamma) + 1 > \log(1+(n-\gamma)/2\gamma) + 1 = \log(1+n/\gamma)$.

A log-product expression can also be written as “balanced” concatenation of two expressions.

Proposition 6.4. Let $B$ be a log-product expression of degree $n \geq 2$. Then there are two homogeneous expressions $X$ and $Y$ such that $B \equiv X \cdot Y$ and $n/3 \leq \deg X, \deg Y \leq 2n/3$.

Proof. Analogously to the proof of Proposition 6.3 initialize $P = \varepsilon$ and $S = \varepsilon$ and follow the canonical path of $B$ downwards, updating $P$ and $S$ in every step such that the invariant $B \equiv PB_n S$ holds. Eventually, the degree of $P$ or $S$ will jump from $< n/3$ to $\geq n/3$. At each update step, the degree of $P$ (resp. $S$) can increase by at most $(n - \deg P - \deg S)/2$ since we always follow the child of larger degree. Let $d$ be the degree of $P$ before updating, then, after updating, $\deg P \leq d + (n - d - \deg S)/2 = (d + n - \deg S)/2 < (n/3 + n - 0)/2 = 2n/3$ must hold (resp. $\deg S < 2n/3$ must hold). This gives us the factorization $B \equiv X \cdot Y$ with $X = P$ and $Y = B_n S$ (resp. $X = PB_n$ and $Y = S$).

6.3. Utilizing non-commutativity

In Section 5.3 we discussed that the arithmetic bound (Theorem 4.2) is incapable of giving a nontrivial lower bound for the palindromic language $L = L^\text{pal}_{2n} = \{ww^\text{reverse} : w \in \{0,1\}^n\}$. Now, with Theorem 6.2 we can give a (suboptimal but nevertheless exponential) lower bound. This bound itself is not of interest, we present it only as proof of concept.

Let $n = 3k$ for a positive integer $k$. Take an arbitrary log-product expression $B$ with $L(B) \subseteq L$. By Proposition 6.4 there are homogeneous expressions $X, Y$ such that $B \equiv X \cdot Y$ and $2n/3 \leq \deg X, \deg Y \leq 4n/3$. Consider a word $w = w_1 \cdots w_{2n}$ described by $X \cdot Y$. Since $\deg X \geq 2n/3 = 2k$, all letters $w_1, \ldots, w_{2k}$ belong to $X$, and since $\deg Y \geq 2n/3 = 2k$, all letters $w_{4k+1}, \ldots, w_{6k}$ belong to $Y$. By definition of $L$, we must have $w_i = w_{6k+1-i}$ for all $i \in [3k]$. Thus, all words in $L(X)$ have $w_{6k} w_{6k-1} \cdots w_{2k}$ as prefix and all words in $L(Y)$ have $w_{2k} w_{2k-1} \cdots w_1$ as suffix. Hence, $|L(B)| = |L(X)| \cdot |L(Y)| \leq 2^k \cdot 2^k = 2^{2k} = 2^n/3$.

7. Applications of the log-product bound

In the next three subsections we demonstrate applications of Theorem 6.2 on the divisibility language $L_{n,k}^\text{div}$, the parity language $L_{n,k}^\text{even}$ and the permutation language $P_n$. We will abbreviate the number $|L(R)|$ of described words of an expression $R$ to $|R|$.
7.1. The divisibility language

Let $p$ be an odd integer. Ellul et al. [3] considered the language of all binary numbers that are divisible by $p$. This language has small DFAs with just $p$ states (see Example 2), but it seems that expressions must be large. However, no lower bound is known so far. Here we consider the $n$-slice of this language. For a word $w \in \{0, 1\}^*$ denote its interpretation as binary number by $\langle w \rangle_2$; we assume that the most significant bit is the leftmost letter, for example $\langle 0101 \rangle_2 = 5$, and for convenience let $\langle \epsilon \rangle_2 = 0$. The divisibility language

\[ L_{n,p}^{\text{div}} = \{ w \in \{0, 1\}^n : \langle w \rangle_2 \equiv_p 0 \} \]

consists of all binary numbers with $n$ bits that are divisible by $p$. This language also has small DFAs with $O(np)$ states, and expressions of length $\rpn(L_{n,p}^{\text{div}}) \leq O(np \cdot p^{\log(n/\log p)})$ (see below). So, the following lower bound is tight, apart from small polynomial factors.

**Theorem 7.1** (Divisibility language). Let $p > 2$ be odd. Then any regular expression describing the divisibility language $L_{n,p}^{\text{div}}$ has length at least

\[ \rpn(L_{n,p}^{\text{div}}) \geq \Omega \left( n^{-1}p^{\log(n/\log p)-2} \right). \]

In particular, if $p$ is constant, then $\Omega(n^{-1}p^{\log n}) \leq \rpn(L_{n,p}^{\text{div}}) \leq O(np^{\log n})$ holds.

**Proof.** Let $L = L_{n,p}^{\text{div}}$ and $B$ be any log-product expression with $L(B) \subseteq L$. We will show an upper bound $h = 2^n n \cdot p^{-\log(n/\log p)+1}$ on the number $|B|$ of words in $L(B)$. Every $p$-th natural number (beginning with $0$) is divisible by $p$, so there are $|L| \geq 2^n/p$ words in $L$. Hence, the claimed bound $\rpn(L) \geq |L|/h \geq 2^n/p \cdot 2^{-n}n^{-1}p^{\log(n/\log p)-1} = \Omega(n^{-1}p^{\log(n/\log p)-2})$ will follow by Theorem 6.2.

For $\gamma := \log p$ let $P_1 \cdots P_m S_m \cdots S_1$ be a $\gamma$-factorization of $B$ ensured by Proposition 6.3. To prove the bound on $|B|$ we upper bound the number of words described by each factor. For $r \in \{0, 1, \ldots, p-1\}$ and $d \in \mathbb{N}$ let $L_{d,p}^r := \{ w \in \{0, 1\}^d : \langle w \rangle_2 \equiv_p r \}$ be the language of all $d$-bit numbers that have remainder $r$ when divided by $p$, for example $L_{n,p}^0 = L_{n,p}^{\text{div}}$. For a word $w$ with $\langle w \rangle_2 \equiv_p r$ and a word $x$ of length $d$ with $\langle x \rangle_2 \equiv_p r'$, their concatenation satisfies $\langle wx \rangle_2 \equiv_p r \cdot 2^d + r'$. Since $p$ is odd, the mapping $r \mapsto r \cdot 2^d \mod p$ is a bijection over $\{0, 1, \ldots, p-1\}$ for every $d$. Thus, if $\langle w \rangle_2 \not\equiv_p \langle v \rangle_2$ holds for two words $w$ and $v$ of same length, then also $\langle wx \rangle_2 \not\equiv_p \langle vx \rangle_2$ and $\langle wx \rangle_2 \not\equiv_p \langle vx \rangle_2$ hold for any word $x$.

Call a homogeneous expression $T$ pure, if all words in $L(T)$ have the same remainder when divided by $p$, that is, if $L(T) \subseteq L_{r,p}^r$ holds for some $r$ and $d$.

**Claim 1.** Every factor $F$ of $B$ is pure.

**Proof of Claim 1.** Assume to the contrary that some factor $F$ of $B$ is not pure, i.e., there are words $w, v \in L(F)$ with $\langle w \rangle_2 \not\equiv_p \langle v \rangle_2$. Since $F$ is a factor of $B$, there are words $x, y$ such that $xwy, xvy \in L(B) \subseteq L$. The observation above yields $\langle xwy \rangle_2 \not\equiv_p \langle xvy \rangle_2$. But all words in $L$ must have the same remainder $r = 0$, a contradiction. \hfill $\square$

Let $F_i := P_i \cdot S_i$ and $d_i := \deg F_i = \deg P_i + \deg S_i$. The argument above also implies that concatenations of pure expressions are pure themselves, so every expression $F_i$ is pure. For all $d$ and all $r$ we have $|L_{d,p}^r| \leq [2^d/p] \leq 2^d/p + 1$, and since every expression $F_i$ describes a subset of some language $L_{d_i,p}^r$, the inequality $|F_i| \leq 2^{d_i}/p + 1$ holds. Proposition 6.3 ensures that $d_i \geq \log p$ holds for all $i \in [m-1]$ and therefore $|F_i| \leq 2^{d_i}/p + 1 \leq 2 \cdot 2^{d_i}/p$. Further, since our alphabet is binary, trivially $|F_m| \leq 2^{d_m}$ must hold. Finally, recall that $\sum_{i=1}^m d_i = n$ and $m \geq \log(1 + n/\gamma) \geq \log(n/\log p)$. Hence,

\[ |B| = \prod_{i=1}^m |F_i| \leq 2^{d_m} \cdot \prod_{i=1}^{m-1} 2 \cdot 2^{d_i}/p = 2^{\sum_{i=1}^m d_i} \cdot (2/p)^{m-1} \leq 2^n \cdot (2/p)^{\log(n/\log p)-1} \leq 2^n n \cdot p^{-\log(n/\log p)+1} =: h. \]
Upper bound. In Example 2 we gave an upper bound $\text{rpn}(L_{n,p}^{\text{div}}) \leq O(np \cdot p^{\log n})$. Now we improve this bound to $O(np \cdot \log(n/\log p))$. To simplify notation assume that $n$ is a power of 2. For each $r \in \{0, \ldots, p-1\}$ and $d \in \{n, n/2, n/4, \ldots\}$ define the expression $R_{r,d}^p$ for the language $L_{r,d}^p$ recursively by $R_{r,d}^p := \sum_{r_1, r_2} R_{r_1/2,p}^r \cdot R_{d/2,2}^r$, where the sum ranges over all $p$ combinations $(r_1, r_2)$ with $r_1 2^{d/2} + r_2 \equiv_r r$. If $d < \log p$, then $L_{r,d}^p$ contains at most one single word $w$, in this case let $R_{r,d}^p := w$. Finally, $R_{0,n}^0$ describes the language $L_{n,p}^{\text{div}}$. This recursion has $2p$ branches in each step, depth at most $\lceil \log(n/\log p) \rceil$ and every base case expression has length at most $\log p$. Thus, length $\text{rpn}(L_{n,p}^{\text{div}}) \leq O(\log p \cdot (2p)^{\lceil \log(n/\log p) \rceil}) \leq O(np \cdot p^{\log(n/\log p)})$ suffices.

Blow-up of DFA Conversion. Gruber and Johannsen [4] showed that converting a DFA for a finite language into an expression can cause a blow-up of $n^{(\log n)/192}$, this is optimal apart from the factor in the exponent. (They actually stated the factor 1/75 instead of 1/192 in [4, Thm. 10], but there seems to be a minor mistake in their proof regarding the number of states of the DFA.) We now can improve this factor to $1/4 - o(1)$.

Corollary 7.2. There are finite languages $L_m$ that can be accepted by DFAs with $m$ states, but require regular expressions of length at least $m^{(\log m)/4 - o(\log \log m)}$.

Proof. Let $n = p = \sqrt{m} \geq 3$ and $L_m = L_{n,p}^{\text{div}}$. This language can be accepted by a DFA with $m = np$ states. The slice DFA described for $L_{r,d}^p$ in Example 2 has at most $p$ states in each layer $Q_j$ for $j \in \{n-1\}$, and additionally one initial, one final and one trap state. This leaves us with at most $3 + (n-1)p \leq np = m$ states in total, since $p \geq 3$. Theorem 7.1 yields the lower bound $\text{rpn}(L_{n,p}^{\text{div}}) \geq n^{-1} \cdot p^{\log(n/\log p) - 2}$. For $n = p = m^{1/2}$ we get $\text{rpn}(L_m) \geq n^{\log(n) - \Theta(\log \log n)} = m^{(\log m)/4 - \Theta(\log \log m)}$.$\square$

7.2. The parity language

Let $\Sigma = \{1, \ldots, k\}$ be an alphabet for $k \geq 2$ and $n$ be even. Consider the parity language

$L_{n,k}^{\text{even}} = \{w \in \Sigma^n : |w|_j \equiv_2 0 \text{ for all } j \in \Sigma\}$

doing all length $n$ words that have an even number of occurrences of every letter. This language naturally generalizes the XOR language $L_n^{\text{xor}} = \{w \in \{0,1\}^n : |w|_1 \text{ is even}\}$ mentioned in the introduction. It can be accepted by DFAs of size $O(2^k n)$ and has expressions of length $O(k 2^k n^k)$, see below. We give an almost matching lower bound.

Theorem 7.3 (Parity language). Let $n$ be even and $k \geq 2$. Then the parity language $L_{n,k}^{\text{even}}$ requires regular expressions of length $\text{rpn}(L_{n,k}^{\text{even}}) \geq \Omega(n^{k-2} (4k \ln k)^{2-k}) = n^{k-2} k^{-k}$.

In particular, if $k$ is constant, then $\Omega(n^{k-2}) \leq \text{rpn}(L_{n,k}^{\text{even}}) \leq O(n^k)$ holds.

To prove Theorem 7.3 it will be convenient to have a look at the minimal DFA $A_{k}^{\text{even}}$ for the infinite variant $L_k^{\text{even}} := \{w \in \Sigma^n : |w|_j \equiv_2 0 \text{ for all } j \in \Sigma\}$ of the parity language. This DFA has states $Q = \{0, 1\}^k$ where the $j$-th bit $q_j$ of a state $q \in Q$ indicates the parity of the number of occurrences of letter $j$. The initial state and the only final state is the all-0 vector $\vec{0}$. The transitions are defined as $\delta(q, j) = q + \vec{e}_j \mod 2$ for all $j \in [k]$ where $\vec{e}_j$ is the $j$-th unit vector. That is, when reading the letter $j$, the automaton flips the $j$-th bit of the current state. The underlying graph of $A_k^{\text{even}}$ is the $k$-dimensional hypercube. An example is given in Fig. 4.

To obtain the claimed upper bound on $\text{rpn}(L_{n,k}^{\text{even}})$, consider the $n$-slice DFA of $A_{k}^{\text{even}}$. Since for every word $w$ we have $|w| = \sum_{j \in \Sigma} |w|_j$, half of the states in each layer of the slice DFA are unreachable and can be removed. So we obtain a DFA of width $2^{k-1}$ and length $n$ accepting $L_{n,k}^{\text{even}}$. By Proposition 3.2 the upper bound $\text{rpn}(L_{n,k}^{\text{even}}) \leq O(n 2^{(k-1)(\log n + 1)}) = O(k 2^k n^k)$ follows.

We now turn to the proof of Theorem 7.3. For every $q \in Q$ and $d \in \mathbb{N}$ let $L_{d,q}^{\text{even}}$ be the language of all words $w \in \Sigma^d$ that end in state $q$ when given as input to the DFA $A_k^{\text{even}}$. These are exactly those words $w$ for which $|w|_j \equiv q_j$ holds for all $j \in \Sigma$. For example $L_{0,k}^{\text{even}} = L_{n,k}^{\text{even}}$. We need an estimate on the number of words in the languages $L_{n,k}^{\text{even}}$. 

17
Lemma 7.4. Let \( n, k \in \mathbb{N}, k \geq 2 \). Then the following hold:

(a) \( |L_{n,k}^{\text{even}}| \geq k^n \cdot 2^{1-k} \) for even \( n \),

(b) \( |L_{n,k}^{\text{odd}}| \leq k^n \cdot 2^{2-k} \) for \( n \geq k \ln k \) and all \( q \in \{0,1\}^k \).

The proof of this lemma is somewhat technical and is therefore postponed to the end of this subsection.

Proof of Theorem 7.3. Let \( L = L_{n,k}^{\text{even}} \) and \( B \) be any log-product expression with \( L(B) \subseteq L \). Our goal is to show an upper bound on the number \( |B| \) of words in \( L(B) \). For \( \gamma := k \ln k \) let \( P_1 \cdots P_m S_m \cdots S_1 \) be a \( \gamma \)-factorization of \( B \) ensured by Proposition 6.3. To prove the bound on \( |B| \) we upper bound the number of words described by each factor of \( B \). Call a homogeneous expression \( T \) pure if its described language is a subset of \( L_{n,k}^{\text{even}} \) for some \( d \in \mathbb{N} \) and \( q \in \{0,1\}^k \).

Claim 2. Every factor \( F \) of \( B \) is pure.

Proof of Claim 2. Assume to the contrary that some factor \( F \) is not pure, i.e., there are words \( w, v \in L(F) \) and a letter \( a \in \Sigma \) such that \( |w|_a \) is even and \( |v|_a \) is odd. Since \( F \) is a factor of \( B \), there are words \( x \) and \( y \) such that \( xwy, xvy \in L(B) \subseteq L \). But then \( |w'|_a = |x|_a + |w|_a + |y|_a \) is odd or \( |v'|_a = |x|_a + |v|_a + |y|_a \) is even, a contradiction to all words in \( L \) having an even number of occurrences of every letter.

Let \( F_i := P_i \cdot S_i \) and \( d_i := \deg F_i = \deg P_i + \deg S_i \). The argument above also implies that concatenations of pure expressions are pure themselves, so every \( F_i \) is pure. For all \( i \in [m-1] \), Proposition 6.3 ensures \( d_i \geq \gamma = k \ln k \) and so Lemma 7.4 (b) yields \( |F_i| \leq k^{d_i}2^{2-k} \). Moreover, \( |F_m| \leq k^{d_m} \) trivially holds and recall that \( \sum_{i=1}^{m} d_i = n \). Hence,

\[
|B| = \prod_{i=1}^{m} |F_i| \leq k^{d_m} \cdot \prod_{i=1}^{m-1} (k^{d_i}2^{2-k}) = k^{\sum_{i=1}^{m} d_i} \cdot 2^{(2-k)(m-1)} \leq k^n \cdot 2^{(2-k)(m-1)} =: h. \tag{2}
\]

According to Lemma 7.4 (a) there are at least \( |L| \geq k^n \cdot 2^{1-k} \) words in \( L \). Hence, the bound

\[
\text{rpn}(L) \geq |L|/h \geq 2^{1-k+(k-2)(m-1)} = \Omega(2^{(k-2)(m-2)}) \tag{3}
\]

follows by Theorem 6.2. We have \( m \geq \log(1 + n/\gamma) \geq \log(n/(k \ln k)) = \log n - \log(k \ln k) \). Thus,

\[
\text{rpn}(L) \geq \Omega(2^{(k-2)((\log n - \log(k \ln k))-2)}) = \Omega(n^{k-2} \cdot (4k \ln k)^{2-k}). \tag{4}
\]
Proof of Lemma 7.4. Our goal is to show these two inequalities:

\[ |L_{n,k}^{\text{even}}| \geq k^n \cdot 2^{1-k} \text{ for even } n, \quad (4) \]

\[ |L_{n,k}^{\text{odd}}| \leq k^n \cdot 2^{2-k} \text{ for } k \geq 2, n \geq k \ln k \text{ and } q \in \{0,1\}^k. \quad (5) \]

We analyze the probability of a random word \(w \in \Sigma^n\) being accepted by the DFA \(A_n^{\text{even}}\), recall that \(\Sigma = [k]\) is our alphabet and that \(A_n^{\text{even}}\) has \(Q = \{0,1\}^k\) as its set of states. Let the word \(w = w_1 \cdots w_n\) be drawn uniformly at random from \(\Sigma^n\); that is, each letter \(w_i\) is chosen independently uniform from \(\Sigma\). If we travel for each letter \(w_i\) along the corresponding edge of \(A_n^{\text{even}}\), this process corresponds to a random walk on the \(k\)-dimensional hypercube \(\{0,1\}^k\), starting in the origin \(\vec{0}\). For a state \(\vec{q}\) let \(p_{\vec{q}}\) be the probability of ending in \(\vec{q}\) after \(n\) steps and let \(|\vec{q}|\) be the number of ones in \(\vec{q}\).

According to Diaconis, Graham and Morrison\(^6\) [37, Lem. 1] the probability \(p_{\vec{q}}\) is given by

\[ p_{\vec{q}} = 2^{-k} \sum_{j=0}^k \left( 1 - \frac{2j}{k} \right)^n \sum_{i=0}^{|\vec{q}|} (-1)^i \binom{|\vec{q}|}{i} \binom{k - |\vec{q}|}{j - i}. \quad (6) \]

The probability of a word being accepted by \(A_n^{\text{even}}\) is exactly the probability \(p_{\vec{0}}\) of ending in the origin \(\vec{0}\). So, to show Eq. (4) we need to lower bound \(p_{\vec{0}}\) for even \(n\). For \(q = \vec{0}\) the probability given in Eq. (6) simplifies to \(p_{\vec{0}} = 2^{-k} \sum_{i=0}^k \binom{k}{i} \binom{k-i}{i} \). For even \(n\), every summand is nonnegative. Hence, we can drop all summands with \(0 < j < n\) to obtain the lower bound

\[ p_{\vec{0}} \geq 2^{-k} \sum_{j=0}^k \left( 1 - \frac{2j}{k} \right)^n \binom{k}{j} = 2^{-k} \left[ 2^n \binom{k}{0} + (-1)^n \binom{k}{k} \right] = 2^{1-k}. \quad (7) \]

To show Eq. (5) we need to upper bound the probability \(p_{\vec{q}}\) given by Eq. (6) for an arbitrary state \(q\). First, we bound the inner sum using the Chu–Vandermonde identity \(\sum_{i=0}^{\lfloor |\vec{q}|/2 \rfloor} \binom{|\vec{q}|}{i} \binom{k-|\vec{q}|}{j-i} = 1\) (which is clear from interpreting the summands as probabilities of a hypergeometric distribution), and the fact that the binomial coefficient \(\binom{k-|\vec{q}|}{j-i}\) in the inner sum of Eq. (6) is zero for all values \(i > j\):

\[ Z_j := \sum_{i=0}^{\lfloor |\vec{q}|/2 \rfloor} (-1)^i \binom{|\vec{q}|}{i} \binom{k - |\vec{q}|}{j - i} \leq \sum_{i=0}^{\lfloor |\vec{q}|/2 \rfloor} \binom{|\vec{q}|}{i} \binom{k - |\vec{q}|}{j - i} = \binom{k}{j}. \quad (8) \]

Analogously, we obtain \(Z_j \geq -\binom{k}{j}\). Then

\[ p_{\vec{q}} = 2^{-k} \sum_{j=0}^k \left( 1 - \frac{2j}{k} \right)^n \left( 1 \cdot Z_j + \sum_{j<k/2} \left( 2j/k - 1 \right)^n Z_{k-j} \right). \quad (9) \]

If \(n\) is odd, then all summands are positive in the first sum and negative in the second, and we use the bound \(Z_j \leq \binom{k}{j}\) in the first sum and the bound \(-Z_{k-j} \leq \binom{k}{k-j} = \binom{k}{j}\) in the second sum. If \(n\) is even then the summands in both sums are positive and we use the bounds \(Z_j \leq \binom{k}{j}\) and \(Z_{k-j} \leq \binom{k}{k-j} = \binom{k}{j}\). In both cases we get

\[ p_{\vec{q}} \leq 2^{-k} \sum_{j<k/2} \binom{k}{j} \left( 1 - \frac{2j}{k} \right)^n \leq 2^{1-k} \sum_{j<k/2} k^j e^{-2jn/k} = 2^{1-k} \sum_{j=0}^{\infty} \left( k e^{-2n/k} \right)^j \leq 2^{1-k} \cdot \frac{1}{1 - k e^{-2n/k}} \leq \frac{k \geq 2}{1-k} \leq 2^{2-k}. \quad (10) \]

Since there are \(|\Sigma^n| = k^n\) possible choices for a word \(w \in \Sigma^n\), we have \(|L_{n,k}^{\text{even}}| = p_{\vec{q}} \cdot k^n\), and the claimed bounds \(|L_{n,k}^{\text{even}}| = k^n p_{\vec{q}} \geq k^n \cdot 2^{1-k}\) and \(|L_{n,k}^{\text{odd}}| = k^n p_{\vec{q}} \leq k^n \cdot 2^{2-k}\) follow from Eqs. (7) and (11).

\(^6\)To be precise, they consider a slightly different random walk with self-loops. This difference only affects the term \(2j/k\).
7.3. The permutation language

Let \( n \geq 1 \) and consider the language \( P_n \) of all permutations over the alphabet \( \Sigma = [n] \). A (partial) DFA for \( P_n \) with \( 2^n \) states can be constructed as follows. Let \( Q = \{ q \subseteq [n] \} \) be the set of states, \( i = \emptyset \) the initial state, \( [n] \) the only final state and the (partial) transition function defined as \( \delta(q, a) = q \cup \{ a \} \) for all \( a \in \Sigma \setminus q \). An example for \( n = 3 \) is given in Fig. 5.

Jerrum and Snir [9] showed a lower bound of \( \text{rpn}(P_n) \geq 2^n - 2 \) by a method for (non-commutative) circuit complexity. Independently, Ellul et al. showed a lower bound of \( 2^n - 1 \) [3, Thm. 29] by the fooling set method. These bounds were recently improved by Molina Lovett and Shallit [8] to \( 4^n n^{-(\log n)/4 + \Theta(1)} \) by a custom argument; they also showed that this bound is tight. We give an alternative, simpler proof of this latter bound using the log-product bound (Theorem 6.2).

**Theorem 7.5** (Permutation language, [8]). The permutation language \( P_n \) requires regular expressions of length at least \( \text{rpn}(P_n) \geq \Omega(4^n n^{-(\log n)/4 - 3/4}) \).

We claim that any log-product expression describing a subset of the permutation language \( P_n \) describes not more than roughly \( (n/2)! (n/4)! (n/8)! \cdots 2! \) words. By the Stirling approximation \( n! \sim \sqrt{2\pi n} (n/e)^n \), this term is approximately \( n! \cdot 4^{-n} n^{3/2 + \log n} \). In the following lemma we make this estimate precise.

**Lemma 7.6.** Let \( n \geq 1 \) and \( B \) be a log-product expression with \( L(B) \subseteq P_n \). Then \( B \) describes at most \( |B| \leq h(n) := n! \cdot 4^{-n} n^{3/2 + \log n} \) words.

We show first that Theorem 7.5 follows from this lemma and then give its proof.

**Proof of Theorem 7.5.** Let \( B \) be any log-product expression with \( L(B) \subseteq P_n \). By Lemma 7.6 we have \( |B| \leq h(n) \) and there are \( n \) words in \( P_n \). Thus, Theorem 6.2 yields

\[
\text{rpn}(P_n) \geq |P_n|/h(n) = 4^{n-1} n^{-(\log n)/4 - 3/4}.
\]

**Proof ofLemma 7.6.** In contrast to the proofs of Theorems 7.1 and 7.3 we do not use \( \gamma \)-factorizations here, but work directly with Definition 6.1. We proceed by induction. If \( n = 1 \), then only one single word (consisting of a single letter) can be described, and \( |B| \leq 1 = 1! \cdot 4^0 1^{3/4 + 0/4} = h(1) \), as desired.

Now let \( n \geq 2 \) and assume that the claim holds for all \( k < n \). By Definition 6.1, there are homogeneous expressions \( A, B' \) such that \( B = AB' \) or \( B = B'A \) holds, \( B' \) is log-product and \( k := \deg B' \geq \deg A \). We can assume w.l.o.g. that \( \deg A = n - k \geq 1 \). Since every word described by \( B \) is a permutation, there must be a partition \( \Sigma_A \cup \Sigma_{B'} = \Sigma \) of the alphabet \( \Sigma = [n] \) with \( |\Sigma_A| = \deg A, |\Sigma_{B'}| = \deg B' \) such that all words described by \( A \) and by \( B' \) are permutations over \( \Sigma_A \) and \( \Sigma_{B'} \), respectively. Thus, we have \( |A| \leq (\deg A)! = (n - k)! \) and, since \( B' \) is log-product, by induction \( |B'| \leq h(k) \) holds. Hence,

\[
|B| = |A| \cdot |B'| \leq (n - k)! h(k) = (n - k)! k! \cdot 4^{1-k} k^{3/2 + \log k} = 4n! \left( \frac{n}{k} \right)^{-1} 4^{-k} k^{3/2 + \log k}.
\]
We claim that \( g \) is convex for \( n \geq 16 + 8 \). Therefore, \( g(k) \) achieves its maximum on the boundary, namely for \( k = n/2 \) or \( k = n - 1 \). We show that in both cases \( g(k) \leq h(n) \) holds, which completes the proof.

**Case 1:** If \( k = n - 1 \), then \( x = (n - 1)/n \) and we get for entropy

\[
H\left(n - 1 \over n\right) = -n \left(\frac{n - 1}{n} \log \frac{n - 1}{n} + \frac{1}{n} \log \frac{1}{n}\right) = (n - 1) \log \frac{n - 1}{n} - (n - 1) \log n \geq (1 - \varepsilon) \log n \geq \log(2n),
\]

where inequality \((*)\) holds since \( \log(1 + \varepsilon) \geq \varepsilon \) holds for all \( \varepsilon \in [0, 1] \). Hence,

\[
g(n - 1) = 4n! \sqrt{2n} \cdot 2^{-n H((n - 1)/n)} \leq 4n! \sqrt{2n} \cdot 2^{(n - 1) \log 2} \leq 4n! \sqrt{2n} \cdot 2^{(n - 1) \log n} \leq 4n! \sqrt{2n} \cdot 2^{(n - 1) \log n} \leq h(n).
\]

**Case 2:** If \( k = n/2 \), then \( x = 1/2 \) and \( H(1/2) = 1 \). Hence,

\[
g(n/2) = 4n! \sqrt{2n} \cdot 2^{-n H(1/2)} \leq 4n! \sqrt{2n} \cdot 2^{-n (n/2) \log 2} \leq 4n! \sqrt{2n} \cdot 2^{-n \log n} \leq h(n),
\]

where equality \((**)*\) holds since

\[
\sqrt{2n} (n/2) \log n \leq \sqrt{2n} \log (n/2) \log n = \sqrt{2n} \log n \leq h(n).
\]

For the cases \( n = 2, \ldots, 7 \) we check by hand that \((n - k)! h(k) \leq h(n)\) holds, see Table 3. For the rest of the proof assume \( n \geq 8 \). We use the inequality \( \binom{n}{k} \geq 2^{n H(1/2)/\sqrt{2n}} \), where \( H(x) = -x \log x - (1-x) \log(1-x) \) is the binary entropy function and \( x = k/n \) (see, e.g., [38, Ch. 10, Lem. 7]). By applying this inequality to Eq. (12), we obtain

\[
|B| \leq g(k) := 4n! \sqrt{2n} \cdot 2^{f(k/n)} \quad \text{with} \quad f(x) := -nH(x) - 2nx + \frac{1}{4} \left[3 \log(nx) + \log^2(nx)\right]. \tag{13}
\]

We claim that \( f(x) \) is convex on the interval \([1/2, 1]\). Intuitively, this is the case because the entropy function \( H(x) \) is concave (and, thus, \(-H(x)\) is convex) and dominates the other terms. To prove this rigorously, we show that the second derivative \( f''(x) \) of \( f(x) \) is positive for all \( x \in [1/2, 1] \):

\[
\ln(2) \cdot f''(x) = \frac{n}{1-x} + \frac{n}{x} + \frac{3-2 \log(nx/e)}{4x^2} \quad \text{since} \quad 1/2 \leq x < 1
\]

\[
> 2n + n - 3 - 2 \log(n/e) \quad \text{since} \quad n \geq 8
\]

\[
\geq 16 + 8 - 3 - 6 + 2 \log e > 0
\]

Since \( f(x) \) is convex for \( x \in [1/2, 1] \), also the function \( g(k) = 4n! \sqrt{2n} \cdot 2^{f(k/n)} \) is convex for \( k \in [n/2, n-1] \). Therefore, \( g(k) \) achieves its maximum on the boundary, namely for \( k = n/2 \) or \( k = n - 1 \). We show that in both cases \( g(k) \leq h(n) \) holds, which completes the proof.

Table 3: For the cases \( n = 2, \ldots, 7 \) we see that for all possible values of \( k \) indeed \(|B| \leq (n - k)! h(k) \leq h(n)\) holds.

| \( n \) | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| \( \max_{n-1 \leq k \leq n/2} \binom{n}{k} h(k) \) | 1.00 | 1.00 | 2.00 | 2.64... | 7.92... | 12.72... |
| \( h(n) \) | 1.00 | 1.32... | 2.12... | 3.89... | 8.58... | 20.74... |

\[21\]
8. Non-homogeneous and infinite languages

Until yet, we only dealt with homogeneous languages (in which all words have the same length). In Theorem 4.2 this restriction is crucial, since for a non-homogeneous expression it is in general not possible to assign a unique position to each leaf. For example, consider the expression $(0 + 00)1$ describing the non-homogeneous language $L = \{01, 001\}$. Then the letter 1 appears in one word at position 2 and in the other at position 3. Also Theorem 6.2 does not work for non-homogeneous languages since the definition of log-product expressions is based on the degree which is only defined for homogeneous expressions.

Nevertheless, in some cases one can circumvent these problems. Following the notation in [9], for a (possibly infinite) language $L$ define its lower envelope $L_{le}$ as the set of all words in $L$ of minimal length, and for a finite language $L$ define its higher envelope $L_{he}$ as the set of all words in $L$ of maximal length. Note that both envelopes are homogeneous languages.

Further, let $\Sigma$ be an alphabet and $\mu : \Sigma \to \mathbb{R}_{\geq 0}$ be a nonnegative weighting of the letters in $\Sigma$. For a word $w \in \Sigma^n$ its weight is $\mu(w) = \sum_{i=1}^{n} \mu(w_i)$. Given such a weighting $\mu$, for a language $L$ its lower $\mu$-envelope $L_{\mu, le}$ is the set of all words in $L$ of minimal weight, and for a finite language $L$ its higher $\mu$-envelope $L_{\mu, he}$ is the set of all words in $L$ of maximal weight.

Lemma 8.1 (Envelopes). Let $\Sigma$ be an alphabet, $L$ a finite language and $L'$ a regular language. Then the following hold:

(a) $rpn(L) \geq rpn(L_{he})$ and $rpn(L') \geq rpn(L'_{he}).$

(b) $rpn(L) \geq rpn(L_{\mu, le})$ and $rpn(L') \geq rpn(L'_{\mu, le})$ for every weighting $\mu : \Sigma \to \mathbb{R}_{\geq 0}$.

Proof. (a) We use the same construction as in [9, Thm. 2.4]; see also [27, Cor. 10]. To prove the first part, we transform an expression $R$ for the language $L$ into an expression for $L_{he}$ without increasing its size. Traverse the syntax tree of $R$ in postorder to homogenize the described language of every node as follows. For a leaf there is nothing to do, so consider an inner node $u$ with children $x$ and $y$. By induction, the children $x, y$ already describe homogeneous languages. If $u = x + y$ is a union node and $x$ and $y$ are of different degree, delete the child (and its whole subtree) of smaller degree. If $u = x \cdot y$ is a concatenation node, there is nothing to do since concatenation preserves homogeneity.

After performing this procedure for all nodes, the resulting expression describes a homogeneous language. Since no word of $L_{he}$ is deleted (we always keep the subtree with the words of larger length), the resulting language must be $L_{he}$.

To show the second part of (a), first replace all star expressions by the empty word $\varepsilon$, i.e., expressions of the form $T^*$ are replaced by $\varepsilon$. Then proceed as above, but this time delete the children of larger degree.

(b) The proof is analogous to (a), this time deleting the children whose described words have smaller resp. larger weight (rather than their length). Further, we do not replace star expressions $T^*$ that describe only words of zero weight.

The next two corollaries give example applications of Lemma 8.1. The threshold language $T_{n,k} = \{w \in \{0, 1\}^n : |w|_1 \geq k\}$ is the monotone closure of the binomial language $B_{n,k}$. Mousavi [7] showed an optimal lower bound of $\Omega(n \log^k(n))$ for $k \leq 3$. We give a bound which is asymptotically optimal for $k = n^{\Theta(1)}$.

Corollary 8.2 (Threshold language). Let $k \leq n/2$. Then the threshold language $T_{n,k}$ requires regular expressions of length $rpn(T_{n,k}) \geq nk^\Theta(\log k)$.

Proof. Let $L = T_{n,k}$ and define the weighting $\mu : \{0, 1\} \to \mathbb{R}_{\geq 0}$ as $\mu(0) = 0, \mu(1) = 1$. Then the weight of a word equals the number of its ones. The words of smallest weight are those with the fewest ones, thus $L_{\mu, le} = B_{n,k}$. From Lemma 8.1(b) the inequality $rpn(L) \geq rpn(B_{n,k})$ follows, and Corollary 5.1 yields the lower bound $rpn(B_{n,k}) \geq nk^\Theta(\log k)$.

Corollary 8.3 (Factors, prefixes, suffixes, subwords). Let $L$ be a homogeneous language and let $L'$ be the set of all factors, all prefixes, all suffixes or all subwords of $L$. Then $L'$ requires regular expressions of length $rpn(L') \geq rpn(L)$.
Proof. The upper envelope of $L'$ is $L$, so the claim follows from Lemma 8.1 (a).

In Corollary 8.3 the condition of homogeneity is crucial. To see this, let $\Sigma$ be an alphabet and take any language $L \subseteq \Sigma^n$ with $\text{rpn}(L) \geq \omega(n)$. Consider the non-homogeneous language $K := L \cup \Sigma^{n+1}$. Since the lower envelope of $K$ is $L$, Lemma 8.1 (a) implies $\text{rpn}(K) \geq \text{rpn}(L) \geq \omega(n)$. Let $K'$ be the set of all prefixes of $K$. Then $K' = \Sigma^{\leq n+1} = \Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^{n+1}$ holds and $\text{rpn}(K') = O(n)$ follows.

8.1. Infinite languages

So far, all results in this paper were obtained only for finite languages. However, also bounds for infinite languages can be derived from them. This is particularly useful for languages with small star height or languages whose star height is hard to determine. The star height $h(R)$ of an expression $R$ is defined as $h(R) = 0$, if $R$ is a letter or the empty word $\varepsilon$, and $h(R_1 \cdot R_2) = h(R_1 + R_2) = \max(h(R_1), h(R_2))$, $h(R^*) = h(R) + 1$. That is, the star height is the maximal nesting depth of star operations. The star height $h(L)$ of a regular language $L$ is the minimal star height of an expression describing $L$. Gruber and Holzer showed the following:

Theorem 8.4 (Gruber and Holzer [5, Thm. 6]). Let $L$ be a regular language. Then $L$ requires regular expressions of length at least $\text{rpn}(L) \geq \Omega(2^{h(L)/3})$.

Consider an infinite variant of the binomial language, namely $B_{n,k}^+ := \{w \in \{0, 1\}^* : |w|_1 \geq k, |w|_0 \geq n-k\}$, and an infinite variant of the permutation language, namely $P_n^+ := \{w \in [n]^* : |w|_a \geq 1 \text{ for all } a \in [n]\}$. For either of these languages we can construct an expression of star height $h = 1$ if we take any expression describing their finite variant ($B_{n,k}$ resp. $P_n$) and replace every letter $a$ by $a \cdot \Sigma^*$. Thus, both languages have star height $h = 1$, and only trivial bounds follow from Theorem 8.4. In contrast, with the help of envelopes and our previous results we obtain the following lower bounds.

Corollary 8.5. Let $k \leq n/2$. Then the language $B_{n,k}^+$ requires regular expressions of length at least $\text{rpn}(B_{n,k}^+) \geq nk^{\Omega(\log k)}$.

Proof. Let $L = B_{n,k}^+$. We have $L_{le} = B_{n,k}$, Lemma 8.1 (a) implies $\text{rpn}(L) \geq \text{rpn}(B_{n,k})$, and Corollary 5.1 yields the claimed lower bound.

Corollary 8.6. The language $P_n^+$ requires regular expressions of length at least $\text{rpn}(P_n^+) \geq \Omega(4^n n^{-(3+\log n)/4})$.

Proof. Let $L = P_n^+$. We have $L_{le} = P_n$, Lemma 8.1 (a) implies $\text{rpn}(L) \geq \text{rpn}(P_n)$, and Theorem 7.5 yields the claimed lower bound.

9. Conclusion

We developed two lower bound methods for the length of regular expressions for finite languages. With the arithmetic bound (Theorem 4.2) we reduced expression length of homogeneous languages $L \subseteq \{0, 1\}^n$ to monotone arithmetic formula complexity. This method naturally refines the methods from Ellul et al. [3] and Gruber and Johannsen [4] who gave reductions to boolean resp. monotone boolean formula complexity. Already with this method we solved two open problems: show a lower bound for the binomial language $B_{n,k} = \{w \in \{0, 1\}^n : |w|_1 = k\}$ and determine the blow-up of intersection and shuffle on finite languages.

With the log-product bound (Theorem 6.2) we provide a more general method which works for homogeneous languages over arbitrary alphabets, explicitly utilizes non-commutativity (see Section 6.3) and even holds for expressions extended by a squaring operation (see Remark 6). As applications we proved lower bounds for the divisibility language $L_{n,k}^{\text{div}}$ of all binary numbers with $n$ bits that are divisible by $p$ and the parity language $L_{n,k}^{\text{par}} = \{w \in [k]^n : |w|_a \text{ is even for all } a \in [k]\}$. These bounds are tight apart from small polynomial factors. For the language $P_n$ of all permutations over $[n]$ we presented an alternative proof of the lower bound recently shown by Molina Lovett and Shallit [8].

Finally, with the help of envelopes we showed how to transfer some results to non-homogeneous and infinite languages (see Section 8).
9.1. Open problems

We obtained bounds for several languages of “combinatorial type” (like the binomial or permutation language) or “modulo arithmetic type” (like the divisibility or parity language), but no result for a “string matching type” language, for example, the language \( L_n, w \) of all length \( n \) words that do not contain a given word \( w \) as a factor. In fact, this problem is the finite variant of a question already posed in [3, Open Problem 3]. For the choice \( w = 1^k \) we conjecture that \( \text{rpn}(L_n, w) \geq n^{\Omega(\log k)} \) can be proven.

**Problem 1.** Prove a lower bound for the language \( L_n, w := \Sigma^n \setminus (\Sigma^* w \Sigma^*) \) for some word \( w \in \Sigma^* \).

The lower bound of \( nk^{\Omega(\log k)} \) shown for the binomial language in Corollary 5.1 is (asymptotically) tight only for large values of \( k \). It would be nice to tighten this bound also for small values, like \( k = O(\log n) \). Applying Theorem 6.2 together with an appropriate weighting on the language (see Remark 7) might be a promising attempt.

**Problem 2.** Prove or disprove a lower bound \( \text{rpn}(B_{n,k}) \geq n^{\Omega(\log k)} \) on the binomial language.

Both our methods rely crucially on the homogeneity of the languages. Envelopes can provide a loophole (see Lemma 8.1), but only in some special situations. For example, Jukna [25] developed lower bound methods for non-homogeneous problems for tropical (max, +) circuits. It might be possible to transfer some ideas to regular expressions.

**Problem 3.** Show lower bounds for finite, but non-homogeneous languages beyond the use of envelopes.

Finally, possibly the most interesting question is whether methods from circuit complexity also work for infinite languages, that is, for regular expressions with star operations.

**Problem 4.** Does circuit complexity help proving lower bounds for regular expressions with star operations?

Acknowledgments

We wish to thank Mario Holldack, Stasys Jukna and Georg Schnitger for inspiring discussions and the anonymous referees for their kind and useful comments.

References

[1] E. Cseresnyes, H. Seiwert, Regular Expression Length via Arithmetic Formula Complexity, in: Descriptional Complexity of Formal Systems, 2020, pp. 26–38. doi:10.1007/978-3-030-62536-8_3.
[2] A. Ehrenfeucht, P. Zeiger, Complexity measures for regular expressions, J. Comput. Syst. Sci. 12 (1976) 134–146. doi:10.1016/S0022-0000(76)80034-7.
[3] K. Ellul, B. Krawetz, J. Shallit, M.-w. Wang, Regular expressions: New results and open problems, J. Autom. Lang. Comb. 9 (2004) 233–256. doi:10.25596/jalc-2004-233.
[4] H. Gruber, J. Johannsen, Optimal lower bounds on regular expression size using communication complexity, in: Foundations of Software Science and Computational Structures (FoSSaCS), 2008, pp. 273–286. doi:10.1007/978-3-540-78499-9_20.
[5] H. Gruber, M. Holzer, Finite automata, digraph connectivity, and regular expression size, in: 35th International Colloquium on Automata, Languages, and Programming (ICALP), 2008, pp. 39–50. doi:10.1007/978-3-540-70583-3_4.
[6] W. Gelade, F. Neven, Succinctness of the complement and intersection of regular expressions, ACM Transactions on Computational Logic (TOCL) 13 (2012) 4:1–19. doi:10.1145/2071368.2071372.
[7] H. Mousavi, Lower bounds on regular expression size, CoRR abs/1712.00811 (2017). arXiv:1712.00811.
[8] A. Molina Lovett, J. Shallit, Optimal regular expressions for permutations, in: 46th International Colloquium on Automata, Languages, and Programming (ICALP), 2019, pp. 121:1–12. doi:10.4230/LIPIcs.ICALP.2019.121.
[9] M. Jerrum, M. Snir, Some exact complexity results for straight-line computations over semirings, J. ACM 29 (1982) 874–897. doi:10.1145/322326.322341.
[10] P. Hrubeš, A. Wigderson, A. Yehudayoff, Non-commutative circuits and the sum-of-squares problem, in: STOC, 2010, pp. 667–676. doi:10.1145/1806689.1806781.
[11] P. Hrubeš, A. Yehudayoff, Formulas are exponentially stronger than monotone circuits in non-commutative setting, in: Conference on Computational Complexity, 2013, pp. 10–14. doi:10.1109/CCC.2013.11.
[12] Y. Filmus, Lower bounds for context-free grammars, Inf. Process. Lett. 111 (2011) 895–898. doi:10.1016/j.ipl.2011.06.006.
Appendix A. Semirings and circuits

In this section we give some background for the connection between regular expressions without star and monotone arithmetic circuit complexity. Although the concepts given here are not actually needed in our proofs, it might help the reader’s intuition.

A semiring \((S, \oplus, \otimes)\) consists of a set \(S\) closed under two binary associative operations “addition” \((\oplus)\) and “multiplication” \((\otimes)\), where addition is commutative and multiplication distributes over addition: \(x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)\) and \((y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x)\). Furthermore, the set \(S\) contains an additive identity element \(0\) with \(x \oplus 0 = x\) and a multiplicative identity element \(1\) with \(1 \otimes x = x \otimes 1 = x\). A semiring is

\footnote{For convenience, in this section we understand “=” semantically rather than syntactically. For example, we say that the union operation \((+)\) is commutative since the expressions \((R_1 + R_2)\) and \((R_2 + R_1)\) describe the same languages although they do not coincide syntactically.}

In this section we give some background for the connection between regular expressions without star and monotone arithmetic circuit complexity. Although the concepts given here are not actually needed in our proofs, it might help the reader’s intuition.

A semiring \((S, \oplus, \otimes)\) consists of a set \(S\) closed under two binary associative operations “addition” \((\oplus)\) and “multiplication” \((\otimes)\), where addition is commutative and multiplication distributes over addition: \(x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)\) and \((y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x)\). Furthermore, the set \(S\) contains an additive identity element \(0\) with \(x \oplus 0 = x\) and a multiplicative identity element \(1\) with \(1 \otimes x = x \otimes 1 = x\). A semiring is

\footnote{For convenience, in this section we understand “=” semantically rather than syntactically. For example, we say that the union operation \((+)\) is commutative since the expressions \((R_1 + R_2)\) and \((R_2 + R_1)\) describe the same languages although they do not coincide syntactically.}

In this section we give some background for the connection between regular expressions without star and monotone arithmetic circuit complexity. Although the concepts given here are not actually needed in our proofs, it might help the reader’s intuition.

A semiring \((S, \oplus, \otimes)\) consists of a set \(S\) closed under two binary associative operations “addition” \((\oplus)\) and “multiplication” \((\otimes)\), where addition is commutative and multiplication distributes over addition: \(x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)\) and \((y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x)\). Furthermore, the set \(S\) contains an additive identity element \(0\) with \(x \oplus 0 = x\) and a multiplicative identity element \(1\) with \(1 \otimes x = x \otimes 1 = x\). A semiring is

\footnote{For convenience, in this section we understand “=” semantically rather than syntactically. For example, we say that the union operation \((+)\) is commutative since the expressions \((R_1 + R_2)\) and \((R_2 + R_1)\) describe the same languages although they do not coincide syntactically.}
commutative if its multiplication is commutative (that is, \( x \otimes y = y \otimes x \) holds), and is idempotent if its addition is idempotent (that is, \( x \oplus x = x \) holds). We consider only semirings with characteristic zero (that is, \( 1 \oplus 1 \cdots \oplus 1 \neq 0 \) holds for every sum of multiplicative identity elements) and with an absorbing additive identity element (that is, \( 0 \otimes x = x \otimes 0 = 0 \) holds for all elements \( x \)).

Given a number \( n \in \mathbb{N} \), a circuit over a semiring is a directed acyclic graph with one sink; parallel edges joining the same pair of nodes are allowed. Each indegree-zero node (a leaf) holds one of the symbols \( x_1, \ldots, x_n \) which are interpreted as formal variables over \( S \), or a semiring element interpreted as a constant; we assume here that only the identity elements 0 and 1 are allowed as constants. Every other node (a gate) has indegree two and performs one of the semiring operations. If the semiring is non-commutative, we assume that predecessors of gates are ordered, i.e., every gate has a left and a right predecessor. The sink is designated as the output gate. The size of a circuit is the total number of its nodes. A circuit whose each node has out-degree \( \leq 1 \) is a formula; that is, formulas are circuits whose underlying graphs are trees.

Since in any semiring \((S, \oplus, \otimes)\) multiplication distributes over addition, each circuit over \( S \) defines (at the output gate) some formal polynomial

\[
f(x_1, \ldots, x_n) = \bigoplus_{m \in M} \lambda_m X(m) \quad \text{with} \quad X(m) := \bigotimes_{i=1}^{[m]} m_i\tag{A.1}
\]

over \( S \) in a natural way, where \( M \subseteq \{x_1, \ldots, x_n\}^* \) is the set of all monomials \( X(m) \) represented as vectors (or, equivalently, as words) \( m = (m_1, \ldots, m_{[m]}) \), and \( \lambda_m \in S \setminus \{0\} \) are (non-zero) coefficients. If the semiring is commutative, we can write this polynomial also as

\[
f(x_1, \ldots, x_n) = \bigoplus_{a \in A} \lambda_a X[a] \quad \text{with} \quad X[a] := \bigotimes_{i=1}^n x_i^{a_i}\tag{A.2}
\]

where \( A \subseteq \mathbb{N}^n \) is its set of exponent vectors, and \( x_i^k \) stands for \( x_i \otimes \cdots \otimes x_i \) \( k \)-times. An example is shown in Fig. A.6. Note the difference in the notation: In Eq. (A.1) the entry \( m_i \in \{x_1, \ldots, x_n\} \) denotes which variable occurs in the monomial \( X(m) \) at position \( i \), in Eq. (A.2) the entry \( a_i \in \mathbb{N} \) denotes the power (possible 0) of the \( i \)-th variable \( x_i \) in the monomial \( X[a] \).

In this paper, we mainly deal with two semirings: the free and the arithmetic semiring.

The free semiring. Let \( n \in \mathbb{N}, \Sigma = \{x_1, \ldots, x_n\} \) be an alphabet and \( \text{Reg}(\Sigma) \) be the set of all regular expressions over \( \Sigma \) without stars. In the non-commutative, idempotent free semiring \((\text{Reg}(\Sigma), +, \cdot)\) we have \( S = \text{Reg}(\Sigma) \), \( x \oplus y = x + y \) (union) and \( x \otimes y = x \cdot y \) (concatenation). The identity elements are \( 0 = \emptyset \) and \( 1 = \varepsilon \). The formal variables \( x_i \) are interpreted as letters. The polynomial given by Eq. (A.1) is

\[
f = \sum_{m \in M} \lambda_m m_1 m_2 \cdots m_{[m]}.
\]

Since the free semiring is idempotent (\( x + x = x \) holds), all coefficients \( \lambda_m = 1 = \varepsilon \) are trivial. Thus, this polynomial \( f \) corresponds to a regular expression \( R_f \) (in “sum-product normal form” where all unions are at the top). In particular, any monomial \( X(m) = m_1 \otimes \cdots \otimes m_{[m]} \) is a word \( m \) and the set \( M \) of monomials is just the (finite) language \( L(R_f) \) described by the expression \( R_f \). A formula over the free semiring is nothing else than a regular expression without stars, viewed as its syntax tree.

Remark 9 (Circuits for infinite regular languages). If we additionally allow unary gates performing the star operation, we obtain circuits that describe arbitrary regular languages, also called straight line programs for regular languages. These circuits can simulate NFAs and can be exponentially more succinct. For more information we refer to [18]; see also [34] and references therein.
Remark 3 defines a monotone boolean function \( L \). We compare produced sets with \( A \). If we plug in the free semiring, this formula defines the regular expression \( cba + aba + a \) and describes the language \( L = \{ cba, aba, a \} \). If we plug in the arithmetic semiring, the formula defines the polynomial \( abc + a^2b + 2a \) and produces the set \( A = \{(1, 1, 1), (2, 1, 0), (1, 0, 0)\} \).

\[
\begin{array}{|c|c|c|}
\hline
\text{Free semiring} & \text{Arithmetic semiring} \\
\hline
\text{Constants 0 and 1} & L(\emptyset) = \emptyset, L(\varepsilon) = \{\varepsilon\} & A(0) = \emptyset, A(1) = \{\vec{0}\} \\
\text{Variables/letters } x_i & L(x_i) = \{x_i\} & A(x_i) = \{\vec{e}_i\} \\
\text{Addition } \oplus & L(R_1 + R_2) = L(R_1) \cup L(R_2) & A(u+v) = A(u) \cup A(v) \\
\text{Multiplication } \otimes & L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2) & A(u \times v) = A(u) \boxplus A(v) \\
\hline
\end{array}
\]

Table A.4: Comparison between the free and the arithmetic semiring. Circuits over the free semiring describe languages \( L \) while monotone arithmetic circuits produce sets \( A \) of exponent vectors. Here, \( \boxplus \) denotes the Minkowski sum.

The arithmetic semiring. In the commutative arithmetic semiring \( (\mathbb{R}_{\geq 0}, +, \times) \), we have \( S = \mathbb{R}_{\geq 0} \). \( x \oplus y = x + y \) (addition) and \( x \otimes y = x \times y \) (multiplication). The identity elements are \( 0 = 0 \) and \( 1 = 1 \). Circuits over the arithmetic semiring are called monotone arithmetic circuits. Here the polynomial given by Eq. (A.2) defines a function over the non-negative reals

\[
f(x_1, \ldots, x_n) = \sum_{a \in A} \lambda_a \prod_{i=1}^n x_i^{a_i},
\]

with the set \( A \subseteq \mathbb{N}^n \) of exponent vectors and positive reals \( \lambda_a \) as coefficients. We call \( A \) the produced set of the circuit and, up to coefficients, the polynomial \( f \) is determined by \( A \). The produced set \( A \subseteq \mathbb{N}^n \) of a circuit can also be defined recursively as follows:

A leaf holding the constant 0 produces the empty set, a leaf holding the constant 1 produces the set \( \{\vec{0}\} \) and a leaf holding a variable \( x_i \) produces the set \( \{\vec{e}_i\} \) where \( \vec{e}_i \) is the unit vector with exactly one 1 at position \( i \) and 0 elsewhere. An addition (\( + \)) gate produces the union of the sets produced by its predecessors. Finally, a multiplication (\( \times \)) gate produces the Minkowski sum \( \boxplus \) of the sets produced by its predecessors; the Minkowski sum of two sets \( A, B \subseteq \mathbb{N}^n \) of vectors is \( A \boxplus B = \{a + b : a \in A, b \in B\} \). The set produced by the entire circuit is the set produced by its output gate. In Table A.4 we compare produced sets with described languages.

The boolean semiring. For completeness also the boolean semiring \( \{0, 1\}, \lor, \land \) should be mentioned. Here we have \( S = \{0, 1\} \), \( x \oplus y = x \lor y \) and \( x \otimes y = x \land y \), both operations are commutative and idempotent. The identity elements are \( 0 = 0 \) and \( 1 = 1 \). A circuit over the boolean semiring is a monotone boolean circuit. The polynomial given by Eq. (A.2) defines a monotone boolean function

\[
f(x_1, \ldots, x_n) = \bigvee_{a \in A} \bigwedge_{i \in [n]: a_i \neq 0} x_i.
\]

Since also the “multiplication” of the boolean semiring is idempotent (\( x \land x = x \) holds), often simplifications in the computed polynomials are possible. This is the reason why monotone boolean complexity can be smaller than monotone arithmetic complexity (see Remark 3).