Electronic Colloquium on Computational Complexity, Report No. 79 (2020)

On Minimizing Regular Expressions Without Kleene Star

Hermann Gruber[1], Markus Holzer[2], and Simon Wolfsteiner[3]

[1] Knowledgepark GmbH, Leonrodstr. 68, 80636 München, Germany
hermann.gruber@kpark.de
[2] Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
holzer@informatik.uni-giessen.de
[3] Institut für Diskrete Mathematik und Geometrie, TU Wien,
Wiedner Hauptstr. 8–10, 1040 Wien, Austria
simon.wolfsteiner@tuwien.ac.at

Abstract. Finite languages lie at the heart of literally every regular expression. Therefore, we investigate the approximation complexity of minimizing regular expressions without Kleene star, or, equivalently, regular expressions describing finite languages. On the side of approximation hardness, given such an expression of size s, we prove that it is impossible to approximate the minimum size required by an equivalent regular expression within a factor of $O\left(\frac{s}{(\log s)^{2+\delta}}\right)$ if the running time is bounded by a quasipolynomial function depending on δ, for every $\delta > 0$, unless the exponential time hypothesis (ETH) fails. For approximation ratio $O(s^{1-\delta})$, we prove an exponential time lower bound depending on δ, assuming ETH. The lower bounds apply for alphabets of constant size. On the algorithmic side, we show that the problem can be approximated in polynomial time within $O(\frac{s \log\log s}{\log s})$, with s being the size of the given regular expression. For constant alphabet size, the bound improves to $O(\frac{s}{\log s})$. Finally, we devise a familiy of superpolynomial approximation algorithms that attain the performance ratios of the lower bounds, while their running times are only slightly above those excluded by the ETH.

1 Introduction

Regular expressions are used in many applications and it is well known that for each regular expression there is a finite automaton that defines the same language and *vice versa*. Automata are very well suited for programming tasks and immediately translate to efficient data structures. On the other hand, regular expressions are well suited for human users and therefore are often used as interfaces to specify certain patterns or languages.

Apart from more traditional applications in text processing tools such as `awk`, `grep`, and `sed`, regular expressions are a pervasive feature in a vast array of modern application programming interfaces (APIs). Recent practical examples, for instance, include:

ISSN 1433-8092

– being an integral part of the Perl programming language,
– querying semi-structured data in the MongoDB NoSql-database,
– defining rewrite rules in the nginx web server,
– specifying trigger rules for builds in continuous deployment with docker hub, and
– defining entities for matching custom data in the DialogFlow conversational suite.

Regarding performance optimization, putting effort into the internal representation inside the regex engine is of course a natural choice [23]. On the other hand, most of the time developers use existing APIs but are not willing, or able, to change the source code of these. Thus, sometimes practitioners may see a need for optimizing the input regular expressions, see, e.g., [19, 25, 26, 29]. In fact, the regular expressions from the above mentioned research papers are without Kleene star, that is, they describe only finite languages.

The problem of minimizing regular expressions accepting infinite languages is PSPACE-complete, and even attaining a sublinear approximation ratio is already equally hard [9]. When restricting to finite languages, it is not difficult to show that regular expression minimization is "only" in Σ_2^{P}, the second level of the polynomial hierarchy. Although there are hardness results for minimizing acyclic nondeterministic finite automata [3, 10], and also for minimizing acyclic context-free grammars [12], apparently no lower bounds about minimizing regular expression without Kleene star were known prior to this work [29]. Recent years have seen a renewed interest in the analysis of computational problems, among others, on formal languages, since more fine-grained hardness results can be achieved based on the exponential time hypothesis (ETH) than with more traditional proofs based on the assumption $\mathsf{P} \neq \mathsf{NP}$ [1, 2, 5, 7, 24, 27]. Roughly speaking, ETH posits that there is no algorithm that decides 3-SAT formulae with n variables in time $2^{o(n)}$, and is one among other strong hypotheses that are used during the last decade to perform fine grained complexity studies; for a short survey on some results obtained by some hypotheses we refer to [28].

We contribute a fine grained analysis of approximability and inapproximability for minimizing regular expressions without Kleene star. On the side of approximation hardness, given such an expression of size s, we prove that it is impossible to approximate the minimum size required by an equivalent regular expression within a factor of $O\left(\frac{s}{(\log s)^{2+\delta}}\right)$ if the running time is bounded by a quasipolynomial function depending on δ, for every $\delta > 0$, unless the ETH fails. For approximation ratio $O(s^{1-\delta})$, we prove an exponential-time lower bound depending on δ, assuming ETH. These lower bounds apply for alphabets of constant size. On the algorithmic side, we show that the problem can be approximated in polynomial time within $O(\frac{s \log\log s}{\log s})$, where s is the size of the given regular expression. For constant alphabet size, the bound improves to $O(\frac{s}{\log s})$. Finally, we devise a familiy of superpolynomial approximation algorithms that attain the performance ratios of the lower bounds, while their running times are only slightly above those excluded by the ETH. For instance, we attain an approximation ratio of $O\left(\frac{s}{(\log s)^{2+\delta}}\right)$ in time $s^{O\left((\log s)^{2+\delta}\right)}$, and a ratio of $s^{1-\delta}$

in time $2^{O(s^\delta)}$. For these ratios, our inapproximability results rule out running times of $s^{o\left((\log s)^{2+\delta-\epsilon}\right)}$ and $2^{o(s^{\delta-\epsilon})}$, respectively, provided the ETH holds.

This paper is organised as follows: in the next section we define the basic notions relevant to this paper. Section 3 covers approximation hardness results for various runtime regimes based on the ETH. Then in Section 4, these negative results are complemented with approximation algorithms that nearly attain these lower bounds. To conclude this work, we indicate possible directions for further research in the last section.

2 Preliminaries

We assume that the reader is familiar with the basic notions of formal language theory as contained in [15]. In particular, let Σ be an *alphabet* and Σ^* the *set of all words over the alphabet* Σ, including the *empty word* ε. The *length of a word* w is denoted by $|w|$, where $|\varepsilon| = 0$, and the total number of occurrences of the alphabet symbol a in w is denoted by $|w|_a$. In this paper, we mainly deal with finite languages. The *order* of a finite language L is the length of a longest word belonging to L. A finite language $L \subseteq \Sigma^*$ is called *homogeneous* if all words in the language have the same length. We say that a homogeneous language $L \subseteq \Sigma^n$ is *full* if L is equal to Σ^n. For languages $L_1, L_2 \subseteq \Sigma^*$, the *left quotient of* L_1 *and* L_2 is defined as $L_1^{-1}L_2 = \{\, v \in \Sigma^* \mid \text{there is some } w \in L_1 \text{ such that } wv \in L_2 \,\}$. If L_1 is a singleton, i.e., $L_1 = \{w\}$, for some word $w \in \Sigma^*$, we omit braces, that is, we write $w^{-1}L_2$ instead of $\{w\}^{-1}L_2$. The set $w^{-1}L_2$ is also called the *derivative of* L_2 *w.r.t. the word* w. In order to fix the notation, we briefly recall the definition of regular expressions and the languages described by them.

The *regular expressions* over an alphabet Σ are defined inductively in the usual way:[4] $\emptyset$, ε, and every letter a with $a \in \Sigma$ is a regular expression; and when E and F are regular expressions, then $(E + F)$, $(E \cdot F)$, and $(E)^*$ are also regular expressions. The language defined by a regular expression E, denoted by $L(E)$, is defined as follows: $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$, $L(E+F) = L(E) \cup L(F)$, $L(E \cdot F) = L(E) \cdot L(F)$, and $L(E^*) = L(E)^*$. The *alphabetic width* or *size* of a regular expression E over the alphabet Σ, denoted by $\mathsf{awidth}(E)$, is defined as the total number of occurrences of letters of Σ in E. For a regular language L, we define its alphabetic width, $\mathsf{awidth}(L)$, as the minimum alphabetic width among all regular expressions describing L.

Let E be a regular expression. According to [13] an expression E is *uncollapsible* if all of the following conditions hold: if E contains the symbol $\emptyset$ then $E = \emptyset$; the expression E contains no subexpression of the form $F \cdot G$ or $G \cdot F$ with $L(F) = \{\varepsilon\}$; if E contains a subexpression of the form $F+G$ or $G+F$ with $L(F) = \{\varepsilon\}$, then $\varepsilon \notin L(G)$; if E contains a subexpression of the form F^*, then $L(F) \neq \{\varepsilon\}$. It was shown in [13] that if E is an uncollapsible regular expression

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

describing a homogeneous language, then E is a *homogeneous expression*, i.e., an expression where none of the symbols $\emptyset$, ε, and * occur in E, or $L(E) = \emptyset$ and $E = \emptyset$ or $L(E) = \{\varepsilon\}$ and $E = \varepsilon$.

We are interested in regular expression optimization w.r.t. its alphabetic width (or, equivalently, its size). An algorithm that returns near-optimal solutions is called an *approximation algorithm*. Assume that we are working on an optimization problem in which each potential solution has a positive cost and that we wish to find a near-optimal solution. Depending on the problem, an optimal solution may be defined as one with maximum possible cost or one with minimum possible cost; the problem may be a maximization or a minimization problem. We say that an approximation algorithm for the problem has a *ratio bound of* $\rho(n)$ if for any input of size n, the cost C of the solution produced by the approximation algorithm is *within a factor of* $\rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

Here, the regular expression optimization problem is obviously a minimization problem. If the approximation algorithm is running in polynomial time, we speak of a *polynomial-time approximation algorithm*.

Since the problem we investigate is in the polynomial hierarchy, proving superpolynomial runtime bounds on approximation would imply $\mathsf{P} \neq \mathsf{NP}$. We thus resort to proving conditional lower bounds based on hardness assumptions such as $\mathsf{P} \neq \mathsf{NP}$. A more fine-grained analysis is possible when using the exponential time hypothesis (ETH) as hardness assumption, as surveyed in [20]. In particular, using the Sparsification Lemma [18], the ETH implies that there is no algorithm running in time $2^{o(n)}$ that decides satisfiability of 3-SAT formulae with n variables. This is of course a much stronger assumption than $\mathsf{P} \neq \mathsf{NP}$.

3 Inapproximability

In this section, we will show that, for a given regular expression without Kleene star, the minimum size required by an equivalent regular expression cannot be approximated within a certain factor if the running time is within certain bounds, assuming the ETH. We start off with an estimate of the required regular expression size for a language which we shall use as gadget. The proofs of the next three lemmata can be found in the Appendix.

Lemma 1. *Let* $S_r = \{\, xy \in \{0,1\}^* \mid |x| = |y| = r \text{ and } x = y \,\}$ *denote the language of all binary square words of length* $2r$*. Then* $2^r \leq \mathsf{awidth}(S_r) \leq 2r \cdot 2^r$.

Proof. The upper bound follows from the fact that a finite language can be described by a regular expression that essentially lists all of its words. Each word in S_r is of length $2r$ and there are 2^r of them.

For the lower bound, we start by giving a fooling set of size $2^r + 1$, thus witnessing that every nondeterministic finite state automaton requires at least

that number of states. For an introduction into the fooling set method, see, e.g., the recent survey in [16]. A fooling set of size 2^r is given by

$$\{ (w, w) \mid w \in \{0,1\}^r \}$$

—it can be readily observed that, for any two distinct pairs (w, w) and (x, x), it holds that $wx \notin S_r$, whereas, for every pair (w, w), it holds that $ww \in S_r$. It is quite straightforward to add one more pair to this fooling set, for example $(\varepsilon, 0^{2r})$. By the position automaton construction, see, e.g., [6], each regular expression of alphabetic width n can be converted into a nondeterministic finite state automaton with $n+1$ states which then gives the desired lower bound of 2^r on alphabetic width.

It has been shown in [11] that taking the quotient of a regular language can cause at most a quadratic blow-up in required regular expression size. Vice versa, the alphabetic width of a language can be lower-bounded by the order of the square root of the alphabetic width of any of its quotients. For our reduction, we need a tighter relationship. This is possible if we resort to special cases. Let us consider homogeneous languages and expressions in more detail. First, we need a simple observation that turns out to be very useful in the forthcoming considerations.

Lemma 2. *Let $L \subseteq \Sigma^n$ be a homogeneous language. If E is an expression describing L, then any subexpression of E describes a homogeneous language as well.*

Proof. It suffices to consider homogeneous expressions. Recall that a regular expression is homogeneous if none of the symbols $\emptyset$, ε, and * occur in E, or $L(E) = \emptyset$ and $E = \emptyset$ or $L(E) = \{\varepsilon\}$ and $E = \varepsilon$.

In the cases that $E = \emptyset$, $E = \varepsilon$, or $E = a$, for $a \in \Sigma$, the statement obviously follows. Because E is a homogeneous expression, we consider two subcases: (i) Let $E = F + G$ such that $L(E) = L(F + G) = L(F) \cup L(G)$ is a homogeneous language, for two regular expressions F and G. Assume, w.l.o.g., that $L(F)$ is not homogeneous, i.e., there are two words $w_1, w_2 \in L(F)$ such that $|w_1| \neq |w_2|$. Then, we also have that $w_1, w_2 \in L(E)$ and thus $L(E)$ is not homogeneous. This contradicts the fact that E describes a homogeneous language. (ii) Let $E = F \cdot G$ such that $L(E) = L(F \cdot G) = L(F) \cdot L(G)$ is a homogeneous language, for two regular expressions F and G. Suppose, w.l.o.g., that $L(F)$ is not homogeneous, i.e., there are two words $w_1, w_2 \in L(F)$ such that $|w_1| \neq |w_2|$. Then, there is some word $w_3 \in L(G)$ such that $w_1 w_3, w_2 w_3 \in L(E)$, but $|w_1 w_3| \neq |w_2 w_3|$. This, however, means that $L(E)$ is not homogeneous, a contradiction. Hence, any subexpression of E describes a homogeneous language as stated in the claim.

Now we are ready to consider the descriptional complexity of quotients of homogeneous languages in detail.

Lemma 3. *Let $L \subseteq \Sigma^n$ be a homogeneous language. Then*

$$\mathsf{awidth}(w^{-1}L) \leq \mathsf{awidth}(L),$$

for any word $w \in \Sigma^*$.

Proof. By [13], it suffices to consider a homogeneous expression E describing the language L, that is, $L = L(E)$. We construct a regular expression E' for the language $w^{-1}L(E)$ by induction as follows: for the base cases, let

$$E' = \begin{cases} \emptyset & \text{if } E = \emptyset, \\ \varepsilon & \text{if } E = \varepsilon \text{ and } w = \varepsilon, \\ a & \text{if } E = a \text{ and } w = \varepsilon, \\ \varepsilon & \text{if } E = a \text{ and } w = a, \\ \emptyset & \text{otherwise.} \end{cases}$$

Finally, we have to consider two subcases: (i) Let $E = F + G$. Then, by induction hypothesis, there are regular expressions F' and G' describing the languages $w^{-1}L(F)$ and $w^{-1}L(G)$. Thus, $E' = F' + G'$ and it is easy to see that $L(E') = w^{-1}L(E)$. (ii) Let $E = F \cdot G$. By Lemma 2, we know that F and G describe homogeneous languages. Thus, let $L(F) \subseteq \Sigma^{n_1}$ with $n_1 < n$. Now assume that $w = u_1u_2$ with $|u_1| = n_1$. Then $E' = F' \cdot G'$, where F' is a regular expression describing $u_1^{-1}L(F)$, which is either the empty language or the language containing only the empty word ε, and G' an expression for the language $u_2^{-1}L(G)$. In case w cannot be decomposed as described above, that is, $|w| < n_1$, we set $E' = F' \cdot G$, where F' is a regular expression for the language $w^{-1}L(F)$. In both of these cases, it is straightforward to verify that $L(E')$ is equal to the set $w^{-1}L(E)$. This completes the construction of the expression E' and proves the stated claim on the alphabetic width of $w^{-1}L$ w.r.t. the alphabetic width of the original language L.

We build upon the classical coNP-completeness proof of the inequality problem for regular expressions without star given in [17, Thm. 2.3]. We recall the reduction to make this paper more self-contained.

Theorem 4. *Let φ be a formula in 3-DNF with n variables and m clauses. Then a regular expression β can be computed in time $O(m \cdot n)$ such that the language $B = L(\beta)$ is homogeneous and B is full if and only if φ is a tautology.*

Proof. Let $\varphi = \bigvee_{i=1}^{m} c_i$ be a formula in 3-DNF. For each clause c_i, let $\beta_i = \beta_{i1}\beta_{i2}\cdots\beta_{in}$, where

$$\beta_{ij} = \begin{cases} (0+1) & \text{if both } x_j \text{ and } \overline{x}_j \text{ are not literals in } c_i, \\ 0 & \text{if } \overline{x}_j \text{ is a literal in } c_i, \\ 1 & \text{if } x_j \text{ is a literal in } c_i. \end{cases}$$

Let $\beta = \beta_1 + \beta_2 + \cdots + \beta_m$. Clearly, $B = L(\beta) \subseteq \{0,1\}^n$. Let w in $\{0,1\}^n$. Then w is in B if and only if w satisfies some clause c_i. Thus $B = \{0,1\}^n$ if and only if φ is a tautology. This completes the reduction.

Now if we wanted to apply the reduction from Theorem 4 to the minimization problem of regular expressions, the trouble is that we cannot predict the minimum required regular expression size for $B = L(\beta)$ in case it is not full. To make this happen, we use a similar trick as recently used in [12] for the analogous case of context-free grammars. In the following lemma, we embed the language S_r of all binary square words of length r together with the language $B = L(\beta)$ (as defined in Theorem 4) into a more complex language C. Depending on whether or not B is full, the alphabetic width of C is at most linear or at least quadratic, respectively, in n. Recall that n refers to the number of variables in the given 3-DNF formula φ.

Lemma 5. *Let φ be a formula in 3-DNF with n variables and m clauses and let β be the regular expression constructed in Theorem 4. Furthermore, let*

$$C = B \cdot \{0,1\}^{2r} \cup \{0,1\}^n \cdot S_r,$$

where $B = L(\beta)$, $2\log n \leq r \leq n$, and S_r is defined as in Lemma 1. Then

$$\mathsf{awidth}(C) = \begin{cases} O(n) & \text{if } B \text{ is full,} \\ \Omega(2^r) & \text{if } B \text{ is not full.} \end{cases}$$

Proof. We distinguish two cases:

1. On the one hand, if $B = \{0,1\}^n$, then the language $B \cdot \{0,1\}^{2r}$ is a superset of $\{0,1\}^n \cdot S_r$. Hence, C can be described by the regular expression $(0+1)^{n+2r}$. Since we assumed above that $2\log n \leq r$, it follows that this expression has size $O(n)$.
2. On the other hand, if B is not full, then there is a word $w \in \{0,1\}^n$ that is *not* a member of B. Now the derivative of C w.r.t. the word w, that is, the language $w^{-1}C$, is equal to the set of squares S_r. By Lemma 3, we have $\mathsf{awidth}(C) \geq \mathsf{awidth}(S_r)$. From Lemma 1, we know that $\mathsf{awidth}(S_r) \geq 2^r$, and thus we can conclude that $\mathsf{awidth}(C) = \Omega(2^r) = \Omega(|S_r|)$ in case B is not full.

This finishes the proof of the lemma.

The following technical lemma will serve as a main ingredient in the proof of Theorem 7.

Lemma 6. *Let E be a regular expression without Kleene star of size s, and let σ and τ be constants such that $\sigma \leq \frac{1}{2}$ and $\sigma < \tau < 1$. Then no deterministic $2^{o\left(\left(\frac{s}{\log s}\right)^{\sigma}\right)}$-time algorithm can approximate $\mathsf{awidth}(L(E))$ within a factor of $O(s^{1-\tau})$, unless ETH fails.*

Proof. We give a reduction from the 3-DNF tautology problem as in Lemma 5. That is, given a formula φ in 3-DNF with n variables and m clauses, we construct a regular expression that generates the language

$$C = B \cdot \{0,1\}^{2r} \cup \{0,1\}^n \cdot S_r.$$

The sets S_r and B are defined as in Lemma 1 and Theorem 4, respectively. Here, the set S_r features some carefully chosen parameter r, which will be fixed later on. For now, we only assume $2 \log n \leq r \leq n$.

Next, we need to show that the reduction is correct in the sense that if B is full, then $\mathsf{awidth}(C)$ is asymptotically strictly smaller than in the case where it is not full. By Lemma 5, it follows that $\mathsf{awidth}(C) = O(n)$ if B is full and $\mathsf{awidth}(C) = \Omega(2^r)$, otherwise. Thus, the reduction is correct, since we have assumed that $r \geq 2 \log n$ and consequently $2^r = \omega(n)$.

It is easy to see that the running time of the reduction is linear in the size of the constructed regular expression describing C. Now we estimate the size of that regular expression. Recall from Theorem 4 that the regular expression β has size $O(m \cdot n)$. By the Sparsification Lemma [18], we may safely assume that $m = O(n)$, so the size is in $O(n^2)$. The set $\{0,1\}^{n+2r}$ admits a regular expression of size $O(n)$; and $\mathsf{awidth}(S_r) = O(2^r \cdot r)$ by Lemma 1. Since we have assumed that $r \geq 2 \log n$, the order of magnitude of the constructed regular expression is $s = \Theta(2^r \cdot r)$.

Now we need to fix the parameter r in our reduction; let us pick $r = \frac{1}{\sigma} \cdot \log n$. Recall that the statement of the lemma requires $\frac{1}{\sigma} \geq 2$, so this is a valid choice for the parameter r—in the sense that the reduction remains correct.

Towards a contradiction, assume that there is an algorithm A_σ approximating the alphabetic width within $O\left(s^{1-\tau}\right)$ running in time $2^{o\left(\left(\frac{s}{\log s}\right)^\sigma\right)}$. Then A_σ could be used to decide whether B is full as follows: the putative approximation algorithm A_σ returns a regular expression size opt^* of at most $O(s^{1-\tau}) \cdot \mathsf{awidth}(C)$.

On the one hand, if B is full, then $\mathsf{awidth}(C) = O(n)$ by Lemma 5. In terms of r, this reads as $\mathsf{awidth}(C) = O\left(2^{\sigma \cdot r}\right)$. In this case, the hypothetical approximation algorithm A_σ returns a solution size opt^* with

$$\begin{aligned}
\mathrm{opt}^* &= O(n \cdot s^{1-\tau}) \\
&= O\left(2^{\sigma \cdot r} \cdot s^{1-\tau}\right) \\
&= O\left(2^{\sigma \cdot r} \cdot (2^r \cdot r)^{1-\tau}\right) \\
&= O\left(2^{\sigma \cdot r} \cdot 2^{(1-\tau) \cdot r} \cdot r^{1-\tau}\right) \\
&= O\left(2^{r \cdot (\sigma + 1 - \tau)} \cdot r^{1-\tau}\right) \\
&= o\left(2^r\right).
\end{aligned}$$

The last line follows since $\sigma + 1 - \tau < 1$ is equivalent to $\sigma < \tau$, as postulated in the statement of the lemma.

On the other hand, in case B is not full, then Lemma 5 states that

$$\mathsf{awidth}(C) = \Omega\left(2^r\right).$$

Using the constants implied by the O-notation, the size returned by algorithm A_σ could thus be used to decide, for large enough n, whether B is full.

It remains to show that the running time of A_σ in terms of n is in $2^{o(n)}$, which contradicts the ETH. We first express s in terms of n:

$$s = \Theta(2^r \cdot r) = \Theta(2^{\log(n^{\frac{1}{\sigma}})} \cdot \log(n^{\frac{1}{\sigma}})) = \Theta(n^{\frac{1}{\sigma}} \cdot \log n).$$

Next, we observe that $\log s = \Theta(\log n)$, since $\log(n^{\frac{1}{\sigma}} \cdot \log n) = \frac{1}{\sigma} \cdot \log n + \log\log n$. We thus can express the running time of the algorithm A_σ in terms of n:

$$\begin{aligned} 2^{o\left(\left(\frac{s}{\log s}\right)^\sigma\right)} &= 2^{o\left(\left(\frac{s}{\log n}\right)^\sigma\right)} \\ &= 2^{o\left(\left(\frac{n^{\frac{1}{\sigma}} \cdot \log n}{\log n}\right)^\sigma\right)} \\ &= 2^{o(n)}, \end{aligned}$$

which yields the desired contradiction.

Now that we have done the heavy lifting, we are in position to state our first inapproximability result.

Theorem 7. *Let E be a regular expression without Kleene star of size s, and let δ be a constant such that $0 < \delta \leq \frac{1}{2}$. Then no deterministic $2^{o\left(s^{\delta-\epsilon}\right)}$-time algorithm can approximate $\mathsf{awidth}(L(E))$ within a factor of $O(s^{1-\delta})$, unless ETH fails. Here, ϵ can be any small positive constant smaller than δ.*

Proof. The result follows by choosing some small positive $\epsilon' < \epsilon$ and invoking Lemma 6 with $\sigma = \delta - \epsilon'$ and $\tau = \delta$. To obtain the simpler expression in the lower bound on running time, observe, that ruling out a running time of $2^{o\left(\left(\frac{s}{\log\log s}\right)^\sigma\right)}$, implies ruling out a running time of $2^{o\left(s^{\delta-\epsilon}\right)}$, because $\sigma = \delta - \epsilon' > \delta - \epsilon$.

We note that the reduction in Lemma 6 is from an coNP-complete problem and runs in polynomial time. Hence, we obtain the following:

Corollary 8. *Let E be a regular expression without Kleene star of size s, and let $\delta > 0$ be a constant. Then no deterministic polynomial-time algorithm can approximate $\mathsf{awidth}(L(E))$ within a factor of $O(s^{1-\delta})$, unless $\mathsf{P} = \mathsf{NP}$.*

Very careful readers may have noticed the difference in the parameter range for δ in the statements of Theorem 7 and Corollary 8. An explanation is in order. We note that for the parameter range $\delta > \frac{1}{2}$, the proof of Theorem 7 would yield the same inapproximability bound in terms of δ assuming ETH. But in that case, the lower bound on the runtime is capped at $2^{o\left(s^{\frac{1}{2}-\epsilon}\right)}$ and no longer grows with δ. This runtime cap is immaterial for Corollary 8, which justifies the parameter range $\delta > 0$ in the statement of the latter.

Again assuming ETH, we can change the parameter r in the reduction in Lemma 6 to trade a sharper inapproximability ratio against a weaker lower bound on the running time. To show this result, we need another technical lemma:

Lemma 9. *Let E be a regular expression without Kleene star of size s, let τ and σ be constants such that $0 < \tau < \sigma < 1$. Then no deterministic $2^{o(\log s)^{\frac{1}{\sigma}}}$-time algorithm can approximate* $\mathsf{awidth}(L(E))$ *within a factor of $o(s/(\log s)^{1+\frac{1}{\tau}})$, unless ETH fails.*

Proof. The ETH rules out $o(2^n)$ algorithms for 3-SAT in n variables and, equivalently, for deciding whether a 3-DNF formula in n variables is a tautology. Consider some 3-DNF formula with n variables and m clauses. We can assume by the Sparsification Lemma that $m = O(n)$. Essentially, we use the same reduction as in the proof of Lemma 6, but this time we construct a regular expression of size $s = \Theta(2^r \cdot r) = \Theta(2^{n^\sigma} \cdot n^\sigma)$ by putting $r = n^\sigma$. Observe that the regular expression can be computed in time $O(s)$, and with $\sigma < 1$, the running time of the reduction is in $2^{o(n)}$.

For the sake of contradiction, assume that there is an algorithm A_σ that approximates $\mathsf{awidth}(L(E))$ within a factor of $o(s/(\log s)^{1+\frac{1}{\tau}})$ running in time $2^{o(\log s)^{\frac{1}{\sigma}}}$. In case the language B from the reduction is full, the putative approximation algorithm A_σ returns a solution size opt^* with

$$\begin{aligned}
\mathrm{opt}^* &= O(n \cdot s/(\log s)^{1+\frac{1}{\tau}}) \\
&= O(n) \cdot \Theta\left(2^{n^\sigma} n^\sigma\right) / \Theta\left((n^\sigma)^{1+\frac{1}{\tau}}\right) \\
&= O(2^{n^\sigma}) \cdot \Theta\left(n^{1+\sigma-\sigma\cdot(1+\frac{1}{\tau})}\right) \\
&= O(2^{n^\sigma}) \cdot o(1) \\
&= o(2^r)
\end{aligned}$$

The term $o(1)$ follows because $1 + \sigma - \sigma \cdot (1 + \frac{1}{\tau}) < 0$, or equivalently, $\sigma > \tau$. Along the lines of the proof of Lemma 6, this shows that A_σ can be used to tell apart tautologies from non-tautologies.

It remains to show that the running time of A_σ in terms of n is in $2^{o(n)}$, which contradicts the ETH. To begin, note that $\log s = \Theta(n^\sigma + \log(n^\sigma))$, which implies $\log s = O(n^\sigma)$. Thus,

$$\begin{aligned}
2^{o(\log s)^{\frac{1}{\sigma}}} &= 2^{o(n^\sigma)^{\frac{1}{\sigma}}} \\
&= 2^{o(n)}
\end{aligned}$$

which yields the desired contradiction.

Theorem 10. *Let E be a regular expression without Kleene star of size s, and let $\delta > 0$ be a constant. Then no deterministic $s^{o(\log s)^{\delta-\epsilon}}$-time algorithm can approximate* $\mathsf{awidth}(L(E))$ *within a factor of $o(s/(\log s)^{2+\delta})$, unless ETH fails. Here, ϵ can be any small positive constant smaller than δ.*

Proof. By setting $\frac{1}{\sigma} = 1 + \delta - \epsilon$ and $\frac{1}{\tau} = 1 + \delta$ and observing that $2^{o(\log s)^{1+\delta-\epsilon}} = s^{o(\log s)^{\delta-\epsilon}}$, the result immediately follows from Lemma 9.

4 Approximability

From the previous section, we know that there are severe limits on what we can expect from efficient approximation algorithms. In this section, we present different approximation algorithms for minimizing regular expressions describing finite languages. Each of them introduces a new algorithmic hook, some of which might be useful in implementations. We start off with an algorithm that requires the input to be specified non-succinctly as a list of words.

Theorem 11. *Let L be a finite language given as a list of words, with s being the sum of the word lengths. Then* $\mathsf{awidth}(L)$ *can be approximated in deterministic polynomial time within a factor of* $O(\frac{s}{\sqrt{\log s}})$.

Proof. We start with a case distinction by alphabet size.

1. The alphabet Σ used in L has size exceeding $\sqrt{\log s}$. It is easy to specify a regular expression of size s which simply "lists" the words in L. Since each alphabet symbol needs to appear at least once in every regular expression describing the language L, we have $\mathsf{awidth}(L) > \sqrt{\log s}$, and the regular expression thus constructed has performance ratio at least $\frac{s}{\sqrt{\log s}}$.
2. The alphabet Σ used in L has size at most $\sqrt{\log s}$. Since L is specified in a non-succinct manner, we can construct the minimum deterministic finite automaton A accepting L in polynomial time. We further distinguish the cases in which A has at most $2^{\sqrt{\log s - \log\log s}-3}$ states or not.
   (a) The minimal deterministic finite automaton A accepting the language L has more than $2^{\sqrt{\log s - \log\log s}-3}$ states. Then every nondeterministic finite automaton accepting L needs at least $\sqrt{\log s - \log\log s} - 3$ states, and the alphabetic width is at least $\Omega(\sqrt{\log s})$. Thus the regular expression we constructed for the first case has performance ratio in $O(\frac{s}{\sqrt{\log s}})$.
   (b) The minimal deterministic finite automaton A accepting the language L has at most $2^{\sqrt{\log s - \log\log s}-3}$ states. We make use of the fact that we can convert an r-state finite automaton accepting a finite language into an equivalent regular expression of size at most $r^{\log r+3}$ if the alphabet size is at most r, see [13]. With $r = 2^{\sqrt{\log s - \log\log s}-3}$, a simple calculation yields

$$\begin{aligned} r^{\log r+3} &= \left(2^{\sqrt{\log s - \log\log s}-3}\right)^{\log 2^{\sqrt{\log s - \log\log s}-3}+3} \\ &\leq \left(2^{\sqrt{\log s - \log\log s}}\right)^{\sqrt{\log s - \log\log s}} \\ &= 2^{\left(\sqrt{\log s - \log\log s}\right)^2} \\ &= 2^{\log \frac{s}{\log s}} \\ &= \frac{s}{\log s}. \end{aligned}$$

The edge case where the alphabetic width of L would be zero can be detected and treated separately; so, without loss of generality, the performance ratio of this regular expression is bounded by $\frac{s}{\log s}$. This meets the required performance guarantee also in this case.

This completes the proof and shows the stated performance guarantee.

Recall that the minimal deterministic finite automaton can be exponentially larger than regular expressions in the worst case, also for finite languages [22]. Also, the conversion from deterministic finite automata to regular expressions is only quasipolynomial in the worst case. These facts of course affect the performance guarantee. Nevertheless, we believe that the scheme from the proof of Theorem 11 is worth a look, since the minimal deterministic finite automaton may eliminate a lot of redundancy in practice. Furthermore, the algorithm works equally if we are able to construct a nondeterministic finite automaton which is smaller than the minimal deterministic finite automaton. To this end, some recently proposed effective heuristics for size reduction of nondeterministic automata could be used [4].

Admittedly, regular expressions are exponentially more succinct than a list of words and our inapproximability results crucially rely on that. So, we now turn to the second approximation algorithm. It makes use of the fact that if a given regular expression E describes very short words only, then it is not too difficult to produce a regular expression that is noticeably more succinct than E. In that case, the algorithm builds a trie, which then can be converted into an equivalent regular expression of size linear in the trie.

For the purpose of this paper, a *trie* (also known as *prefix tree*) is simply a tree-shaped deterministic finite automaton with the following properties:

1. The edges are directed away from the root, i.e., towards the leaves.
2. The root is the start state.
3. All leaves are accepting states.
4. Each edge is labelled with a single alphabet symbol.

The last condition is needed if we want to bound the size of an equivalent regular expression in terms of the nodes in the trie. The following lemma seems to be folklore; the observation is used, e.g., in [14].

Lemma 12. *Let T be a trie with n nodes accepting L. Then an equivalent regular expression of alphabetic width at most $n-1$ can be constructed in deterministic polynomial time from T.*

Proof. The proof is by induction on the height h of T. If the trie consists of a single node, then $L(T)$ is either empty or consists of the empty word, and the claimed statement clearly holds in this case. For the induction step, let T be a trie of height h and assume the statement holds for all tries of height at most $h-1$. When removing the root node and incident edges labelled with $a_1, a_2, \ldots, a_t$, then T falls apart into $t \geq 1$ subtries $T_1, T_2, \ldots, T_t$. Let $n_1, n_2, \ldots n_t$ denote the number of nodes of $T_1, T_2, \ldots, T_t$, respectively. By

induction, the trie-languages $L(T_1), L(T_2), \ldots, L(T_t)$ can be described by regular expressions $E_1, E_2, \ldots, E_t$, respectively, whose sum of their alphabet widths is at most $\sum_{i=1}^{t}(n_i - 1) = n - 1 - t$. Now, as desired, $a_1 \cdot E_1 + a_2 \cdot E_2 + \cdots + a_t \cdot E_t$ is a regular expression of alphabetic width $n - 1$ describing the language $L(T)$.

Now we have collected all tools for an approximation algorithm that works with regular expressions as input, which even comes with an improved approximation ratio.

Theorem 13. *Let E be a regular expression without Kleene star of alphabetic width s. Then* $\mathsf{awidth}(L(E))$ *can be approximated in deterministic polynomial time within a factor of* $O\left(\frac{s \log\log s}{\log s}\right)$.

Proof. We again start with a case distinction by alphabet size.

1. The size of the alphabet used in L is at most $\log s$. We further distinguish the cases in which the order of $L(E)$, i.e., the length of the longest word, is less than $\frac{\log s}{\log\log s}$ or not. The order of $L(E)$ can be easily computed recursively, in polynomial time, by traversing the syntax tree of E. We consider two subcases:
   (a) The order of $L(E)$ is less than $\frac{\log s}{\log\log s}$. We enumerate the words in $L(E)$, e.g., by performing a membership test for each word of length less than $\frac{\log s}{\log\log s}$. Then we use a standard algorithm to construct a trie for $L(E)$. The worst case for the size of T is when L contains all words of length less than $\frac{\log s}{\log\log s}$. Then T is a full $(\log s)$-ary trie of height $\frac{\log s}{\log\log s}$. All nodes are accepting, giving a one-to-one correspondence between the number of nodes in T and the number of words in $L(T)$. That is, the number of nodes in T is equal to
   $$\sum_{i=0}^{\frac{\log s}{\log\log s}-1} (\log s)^i = O\left(\frac{(\log s)^{\frac{\log s}{\log\log s}}}{\log s}\right).$$
   Using the fact that $(\log s)^{\frac{\log s}{\log\log s}} = s$, this is in $O\left(\frac{s}{\log s}\right)$ and we can construct an equivalent regular expression of that size in deterministic polynomial time by virtue of Lemma 12.
   (b) The order of $L(E)$ is at least $\frac{\log s}{\log\log s}$. We make use of the observation that the order of $L(E)$, i.e., the length of the longest word, is a lower bound on the required regular expression size, as observed, e.g., in [6, Proposition 6]. That is, the optimal solution is at least of size $\frac{\log s}{\log\log s}$ and thus the regular expression E given as input is already a feasible solution that is at most $\frac{s \log\log s}{\log s}$ times larger than the optimum solution size.
2. The size of the alphabet used in L is greater than $\log s$. The size of the alphabet used in L is likewise a lower bound on the required regular expression size and, similarly to the previous case, the input is a feasible solution that is at most $\frac{s}{\log s}$ times greater than the optimum solution size.

This proves the stated claim.

For alphabets of constant size, the performance ratio can be slightly improved—by a factor of $\log \log s$.

Theorem 14. *Let E be a regular expression without Kleene star of alphabetic width s over an alphabet of constant size k. Then* $\mathsf{awidth}(L(E))$ *can be approximated in deterministic polynomial time within a factor of* $O\left(\frac{s}{\log s}\right)$.

Proof. We proceed as in the algorithm of Theorem 11, but this time the approximation algorithm branches into Case (1a) if the order of $L(E)$ is less than $\log_k s - \log_k \log s$. Then the number of nodes in the trie T is at most

$$O\left(k^{\log_k s - \log_k \log s}\right) = O\left(\frac{s}{\log s}\right).$$

Further, note that $\log_k s - \log_k \log s = \Omega(\log s)$, yielding the desired performance ratio for Case (1b) as well. And of course, we can omit Case (2) altogether, since we have constant alphabet size.

A better performance ratio can be achieved if we allow for superpolynomial running time. In order to keep things simple, we stick to binary alphabets.

Theorem 15. *Let E be a regular expression without Kleene star of alphabetic width s over a binary alphabet, and let $f(s)$ be a time constructible*[5] *function with $f(s) = \Omega(\log s)$. Then* $\mathsf{awidth}(L(E))$ *can be approximated in deterministic time $2^{O(f(s))}$ within a factor of* $O\left(\frac{s}{f(s)}\right)$.

Proof. Again, we make a case distinction with respect to the order of the language.

1. The order of $L(E)$ is less than $f(s)$. We make use of the fact that there is a context-free grammar generating all regular expressions over binary alphabets [14]. Such a grammar can be used to enumerate all regular expressions of size less than $f(s)$ with polynomial delay [8], and there are $2^{O(f(s))}$ of these in total. For each enumerated candidate expression C and each word w of length less than $f(s)$, we test whether $w \in L(E)$, and if so, we verify that $w \in L(C)$. If C passes all these tests, we can conclude that $L(E) \subseteq L(C)$. To verify whether $L(C) \subseteq L(E)$, we enumerate the words in $L(E)$ and build a trie T that accepts the language. Notice, that the trie has at most $2^{O(f(s))}$ nodes. Since T is a deterministic finite automaton, it can be easily complemented, and we can apply the usual product construction—with the position automaton of C—to check whether $L(C) \cap \Sigma^* \setminus L(T) = \emptyset$. So, in this case, if $L(E)$ admits an equivalent regular expression of size at most $f(s)$, the

[5] We say that a function $f(n)$ is *time constructible* if there exists an $f(n)$ time-bounded multitape Turing machine M such that for each n there exists some input on which M actually makes $f(n)$ moves [15].

optimum solution can be found by exhaustive search with a running time bounded by a polynomial in $2^{O(f(s))}$. For the stated bound on the running time, notice that $2^{(O(f(s)))^{O(1)}} = 2^{O(f(s))}$.

2. The order of $L(E)$ is at least $f(s)$. Again, the order of $L(E)$ is a lower bound on required regular expression size. Thus the regular expression E given as input is already a feasible solution with performance ratio $\frac{s}{f(s)}$.

To compare this with our inapproximability results, we can pick $f(s) = (\log s)^{2+\delta}$ to obtain an approximation ratio of $O\left(\frac{s}{(\log s)^{2+\delta}}\right)$ in time $s^{O\left((\log s)^{2+\delta}\right)}$. In contrast, for this ratio, Theorem 10 rules out a running time of $s^{o\left((\log s)^{2+\delta-\epsilon}\right)}$, for every $\epsilon > 0$. Another pick is $f(s) = s^{\delta}$, yielding an approximation ratio of $s^{1-\delta}$ in time $2^{O(s^{\delta})}$. For this ratio, Theorem 7 rules out a running time of $2^{o\left(s^{\delta-\epsilon}\right)}$, for every $\epsilon > 0$. Thus the upper bound matches the obtained lower bounds (up to ϵ), and there is no room for substantial improvements, unless the exponential time hypothesis fails.

5 Conclusion

We conclude by indicating some possible directions for further research. First, we would like to continue with investigating inapproximability bounds within polynomial time, based on the strong exponential time hypothesis (SETH). Further topics are exact exponential-time algorithms and parameterized complexity. In addition to the natural parameter of desired solution size, the order of the finite language and the alphabet size seem to be natural choices. We remark that the proof of Theorem 15 essentially uses a kernelization technique.

Given the practical relevance of the problem we investigated, we think that implementing some of the ideas from the above approximation algorithms is worth a try. Also, POSIX regular expressions restricted to finite languages are a more complex model than the one we investigated, but a more practical one as well. Although we would rather not expect better approximability bounds in that model, we suspect that character classes and other mechanisms can offer practical hooks for reducing the size of regular expressions.

Also, we would like to stress that the classical computational complexity of minimizing regular expressions without star is not well understood. Here the main difficulty is the size estimation for the regular expressions used as gadgets in reductions. Proving asymptotically tight lower bounds on regular expression size can be quite challenging, even for such simple cases as the set of permutations [21]. While it is easy to prove that regular expression minimization is in Σ_2^{P} for finite languages, apparently no lower bounds were known for this problem prior to this work [29].

Acknowledgments. We would like to thank Michael Wehar for some discussion.

References

1. A. Abboud, A. Backurs, and V. V. Williams. If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2015.
2. K. Bringmann, A. Grnlund, and K. G. Larsen. A Dichotomy for Regular Expression Membership Testing. In *Proceedings of the* 58*th Annual IEEE Symposium on Foundations of Computer Science*, pages 307–318, Berkeley, California, USA, October 2017. IEEE.
3. P. Chalermsook, S. Heydrich, E. Holm, and A. Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In A. S. Schulz and D. Wagner, editors, *Proceedings of the* 22*th Annual European Symposium on Algorithms*, number 8737 in LNCS, pages 235–246, Wroclaw, Poland, September 2014. Springer.
4. L. Clemente and R. Mayr. Efficient reduction of nondeterministic automata with application to language inclusion testing. *Logical Methods in Computer Science*, 15(1), 2019.
5. M. de Oliveira Oliveira and M. Wehar. On the fine grained complexity of finite automata non-emptiness of intersection. In *Proceedings of the* 24*nd International Conference on Developments in Language Theory*, LNCS, Tampa, Florida, USA, March 2020. Springer. To appear.
6. Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-wei Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.
7. H. Fernau and A. Krebs. Problems on Finite Automata and the Exponential Time Hypothesis. *Algorithms*, 10(1), 2017.
8. Ch. C. Florêncio, J. Daenen, J. Ramon, J. Van den Bussche, and D. Van Dyck. Naive infinite enumeration of context-free languages in incremental polynomial time. *Journal of Universal Computer Science*, 21(7):891–911, 2015.
9. G. Gramlich and G. Schnitger. Minimizing nfa's and regular expressions. *Journal of Computer and System Sciences*, 73(6):908–923, September 2007.
10. H. Gruber and M. Holzer. Computational complexity of NFA minimization for finite and unary languages. In *Preproceedings of the* 1*st International Conference on Language and Automata Theory and Applications*, Technical Report 35/07, pages 261–272, Tarragona, Spain, March 2007. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili.
11. H. Gruber and M. Holzer. Language operations with regular expressions of polynomial size. *Theoretical Computer Science*, 410(35):3281–3289, August 2009.
12. H. Gruber, M. Holzer, and S. Wolfsteiner. On minimal grammar problems for finite languages. In M. Hoshi and S. Seki, editors, *Proceedings of the* 22*nd International Conference on Developments in Language Theory*, number 11088 in LNCS, pages 342–353, Kyoto, Japan, September 2018. Springer.
13. H. Gruber and J. Johannsen. Tight bounds on the descriptional complexity of regular expressions. In R. Amadio, editor, *Proceedings of the* 11*th Conference Foundations of Software Science and Computational Structures*, number 4962 in LNCS, pages 273–286, Budapest, Hungary, March–April 2008. Springer.
14. H. Gruber, J. Lee, and J. Shallit. Enumerating regular expressions and their languages. arXiv:1204.4982 [cs.FL], April 2012.
15. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

16. M. Hospodár, G. Jirásková, and P. Mlynárcik. A survey on fooling sets as effective tools for lower bounds on nondeterministic complexity. In H.-J. Böckenhauer, D. Komm, and W. Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, number 11011 in LNCS, pages 17–32. Springer, 2018.
17. H. B. Hunt, III. On the time and tape complexity of languages I. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 10–19, Austin, Texas, USA, April–May 1973. ACM.
18. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
19. P. Krauss. Minimal regular expression that matches a given set of words. Computer Science Stack Exchange, 2017. URL: https://cs.stackexchange.com/q/72344, Accessed: 2020-03-10.
20. D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the European Association for Theoretical Computer Science*, 105:41–72, 2011.
21. A. M. Lovett and J. O. Shallit. Optimal regular expressions for permutations. In Ch. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *LIPIcs*, pages 121:1–121:12, Patras, Greece, July 2019. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.
22. R. Mandl. Precise bounds associated with the subset construction on various classes of nondeterministic finite automata. In *Proceedings of the 7th Princeton Conference on Information and System Sciences*, pages 263–267, March 1973.
23. Justin Mason. A released perl with trie-based regexps! http://taint.org/2006/07/07/184022a.html, 2006. Accessed: 2020-03-10.
24. F. Mráz, D. Průša, and M. Wehar. Two-dimensional pattern matching against basic picture languages. In M. Hospodár and G. Jirásková, editors, *Proceedings of the 24th International Conference on Implementation and Application of Automata*, number 11601 in LNCS, pages 209–221, Košice, Slovakia, July 2019. Springer.
25. pdanese (StackOverflow username). Speed up millions of regex replacements in python 3. Stack Overflow, 2017. URL: https://stackoverflow.com/q/42742810, Accessed: 2020-03-10.
26. P. Scheibe. Regex performance: Alternation vs trie. Stack Overflow, 2019. URL: https://stackoverflow.com/q/56177330, Accessed: 2020-03-10.
27. M. Wehar. Hardness results for intersection non-emptiness. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, Part II*, number 8573 in LNCS, pages 354–362, Copenhagen, Denmark, July 2014. Springer.
28. V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In B. Sirakov, P. Ney de Souza, and M. Viana, editors, *Proceedings of the International Congress of Mathematicians*, pages 3447–3487, Rio de Janeiro, Brazil, April 2018. World Scientific.
29. Ch. Xu. Minimizing size of regular expression for finite sets. Theoretical Computer Science Stack Exchange, 2013. URL: https://cstheory.stackexchange.com/q/16860, Accessed: 2020-03-10.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il