Provably Shorter Regular Expressions from Deterministic Finite Automata (Extended Abstract)

Hermann Gruber[1] and Markus Holzer[2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstraße 67, D-80538 München, Germany
email: gruberh@tcs.ifi.lmu.de

[2] Institut für Informatik, Technische Universität München, Boltzmannstraße 3, D-85748 Garching bei München, Germany
email: holzer@in.tum.de

Abstract. We study the problem of finding good elimination orderings for the state elimination algorithm, which is one of the most popular algorithms for the conversion of finite automata into equivalent regular expressions. Based on graph separator techniques we are able to describe elimination strategies that remove states in large induced subgraphs that are "simple" like, e.g., independent sets or subgraphs of bounded treewidth, of the underlying automaton, that lead to regular expressions of moderate size. In particular, we show that there is an elimination ordering such that every language over a binary alphabet accepted by an n-state *deterministic* finite automaton has alphabetic width at most $O(1.742^n)$, which is, to our knowledge, the algorithm with currently the best known performance guarantee. Finally, we apply our technique to the question on the effect of language operations on regular expression size. In case of the intersection operation we prove an upper bound which matches, up to a small factor, a lower bound recently obtained in [9, 10], and thus settles an open problem stated in [7].

1 Introduction

One of the most basic theorems in formal language theory is that every regular expression can be effectively converted into an equivalent finite automaton, and *vice versa* [14], and algorithms accomplishing these tasks have been known since the beginning of automata theory, see, e.g., [17]. While regular expressions can be converted efficiently into nondeterministic finite automata, the other direction necessarily leads to an exponential blow-up in size [6]. Some very recent results on this problem imply an increase of $2^{\Omega(n)}$ in size, even given a deterministic finite automaton over a binary alphabet [9–11]. In spite of these strong negative results, already early authors noticed that, at least in many cases, it may be possible to improve the standard state elimination algorithm: The authors of the seminal work [17] noticed that the ordering in which the states of the given automaton are processed can greatly influence the size of the resulting regular expression, and an implementation study appearing in the 1960s notes [16]:

> "… a basic fault of the method is that it generates such cumbersome and so numerous expressions initially." …

But only the last few years have seen a renewed interest in heuristic algorithms that produce, at least in some cases, shorter regular expressions than the standard, non-optimized textbook procedure, see, e.g., [3, 7, 12, 18]. However, none of the mentioned algorithms is known to have a better performance guarantee than $O(4^n)$ in the worst case, which is (roughly) the guarantee of the standard textbook algorithms. It is worth mentioning that in [7] a recursive algorithm for converting planar n-state finite automata into regular expressions with a non-trivial performance guarantee of $2^{O(\sqrt{n})}$ was presented. As proved in [10], this bound is asymptotically optimal for the planar case. The mentioned algorithm exploits the separator theorem for planar graphs [15]. This was the starting point of our investigations.

The main idea underlying the graph separator technique is to identify large induced substructures that are "simple" that lead to regular expressions of moderate size or alphabetic width. Such a procedure is seemingly more difficult to implement than a mere state elimination strategy, but we will show how the idea of using separators can be generalized and implemented simply in a divide-and-conquer fashion. The difficulty when applying this idea is that on the one hand large or omnipresent substructures are needed, such that the algorithm can be applied successfully, and on the other hand, these substructures have to produce small regular expressions. These two conditions seem to clash at first thought. Nevertheless, we present two algorithms, one that uses independent sets, the other one induced subgraphs of bounded undirected treewidth, as basic building blocks for a strategy computing a good ordering on the states for the state elimination scheme. Both algorithms when applied to an n-state *deterministic* finite automata attain regular expressions with a performance guarantee of $O(c^n)$, for constants $c < 2.602$ and $c < 1.742$, respectively. As a side result, we identify a structural restriction, namely bounded treewidth, on the transition structure of the given finite automata that guarantees a polynomial upper bound on the resulting regular expression. These new insights on the conversion problem can be applied to some questions regarding the effect of language operations on regular expression size, too. Namely, we present a new algorithm computing a regular expression denoting the intersection of two regular languages. The performance guarantee is proved to be $2^{O(n \log \frac{m}{n})}$, where m and $n \leq m$ are sizes of the given regular expressions. This matches, up to a small factor,[1] a lower bound of $2^{\Omega(n)}$ recently established in [10]. We thus settle a question stated in [7] and complement previous lower bounds from [8–10]. We also prove a nontrivial upper bound for the alphabetic width of the language operation of half-removal, whose descriptional complexity in terms of finite automata was studied recently in [4].

[1] For example, assuming that storing a regular expression of alphabetic width k takes k bytes, and the larger expression is stored in an enormous plain text file taking 1 MByte $= 2^{10}$ KByte disk space, while the smaller one needs only 1 KByte, we still have $\log \frac{m}{n} = 10$.

2 Basic Definitions

We introduce some basic notions in formal language and automata theory—for a thorough treatment, the reader might want to consult a textbook such as [21]. In particular, let Σ be a finite 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 $|\epsilon| = 0$.

A *nondeterministic finite automaton* (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of input symbols, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. The *language accepted* by the finite automaton A is defined as $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \,\}$, where δ is naturally extended to a function $Q \times \Sigma^* \to 2^Q$. A NFA $A = (Q, \Sigma, \delta, Q_0, F)$ is *deterministic*, for short a DFA, if $|\delta(q, a)| \leq 1$, for every $q \in Q$ and $a \in \Sigma$. In this case we simply write $\delta(q, a) = p$ instead of $\delta(q, a) = \{p\}$. Two finite automata are *equivalent* if they accept the same language. Without loss of generality we assume throughout this paper, that every finite automaton accepting a nonempty language has useful states only, i.e., every state is accessible from the initial state and co-accessible from some accepting state—this assumption is compatible with the definition of deterministic finite automata given above.

It is well known that finite automata and regular expressions are equally powerful, i.e., for every finite automaton on can construct an equivalent regular expression. Let Σ be an alphabet. The regular expressions over Σ are defined recursively in the usual way:[2] $\emptyset$, ϵ, and every letter a with $a \in \Sigma$ is a regular expression, and if r_1 and r_2 are regular expressions, then $(r_1 + r_2)$, $(r_1 \cdot r_2)$, and $(r_1)^*$ are also regular expressions. The language defined by a regular expression r, denoted by $L(r)$, is defined as follows: $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(a) = \{a\}$, $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$, and $L(r_1^*) = L(r_1)^*$. The *size* or *alphabetic width* of a regular expression r over the alphabet Σ, denoted by $\mathrm{alph}(r)$, is defined as the total number of occurrences of alphabet symbols of Σ in r. For a regular language L, we define its alphabetic width, $\mathrm{alph}(L)$, as the minimum alphabetic width among all regular expressions describing L. As with finite automata, the notion of equivalence is defined based on equality of the described language.

In the remainder of this section we fix some basic notations from graph theory. A *directed graph*, or *digraph*, $G = (V, E)$ consists of a finite set of vertices V with an associated set $E \subseteq V \times V$ of edges. If the edge relation E is symmetric, the digraph is said to be *symmetric*. Intuitively, a symmetric digraph is obtained by forgetting the orientation of the original edges in G. A digraph $H = (U, F)$ is a *subdigraph*, or simply *subgraph*, of a digraph $G = (V, E)$, if $U \subseteq V$ and for each edge $(u, v) \in F$ with $u, v \in U$, the pair (u, v) is an edge in E. For a subset $U \subseteq V$, the *subgraph induced by* U is the subgraph $G[U] = (U, E \cap (U \times U))$.

[2] 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.

Finally, a *hammock* is a digraph $G = (V, E)$ having two distinguished vertices s and t satisfying the properties (1) that the indegree of s and the outdegree of t is zero, and (2) for every vertex v in G, there is both a path from s to v and a path from v to t. Here s is referred to as the *start vertex* and t as the *terminal vertex* of the hammock G. The remaining set of vertices $Q = V \setminus \{s, t\}$ is called the set of *internal vertices*. It is thus convenient to specify a hammock as a 4-tuple $H = (Q, E, s, t)$. With the finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ we naturally associate a hammock $H(A) = (Q, E, s, t)$, where s and t are designated vertices not appearing in Q that play the role of the initial and a single final state, and $E = \{(s, q_0)\} \cup \{\, (q, t) \mid q \in F \,\} \cup \{\, (p, q) \in Q^2 \mid q \in \delta(p, a), \text{ for some } a \in \Sigma \,\}$. Due to the "dualism" of computations in A and walks in $H(A)$ one can reconstruct the language accepted by A from the walks in $H(A)$—a walk in $H(A)$ is a (possibly empty) sequence of edges along a path, with repeated edges allowed. To this end define the substitution $\sigma : E \to \Sigma^*$ by $(p, q) \mapsto \{\, a \in \Sigma \mid q \in \delta(p, a) \,\}$, if $p, q \in Q$, $(s, q_0) \mapsto \{\epsilon\}$, and $(q, t) \mapsto \{\epsilon\}$, for $q \in F$, which naturally extends to words and languages over E. It is easy to see that $L(A) = \sigma(L^Q_{st})$; here L^Z_{xy}, for $x, y \in Q \cup \{s, t\}$ and $Z \subseteq Q$, refers to the set of all walks in $H(A)$ from vertex x to vertex y whose *internal* vertices are all in Z—the internal vertices of a walk denote those that are visited by the walk after the leaving x and before entering y. This notion naturally extends to L^Z_{XY} for sets $X, Y \subseteq Q \cup \{s, t\}$ and $Z \subseteq Q$. The above definitions are particularly useful in connection with regular expressions because of the following well-known fact (see also [21]).

Lemma 1. *Let Γ and Σ be finite alphabets and r be a regular expression over Γ. Moreover, let $\rho : \Gamma \to 2^{\Sigma^*}$ be a regular substitution, i.e., a substitution satisfying $\rho(a) = L(r_a)$, for some regular expression r_a, for each $a \in \Gamma$. Then a regular expression describing $\rho(L(r))$ is obtained from r by substituting r_a for each letter $a \in \Sigma$.* $\square$

Thus, it suffices to describe the conversion to regular expressions from finite automata on the basis of the associated digraphs, which we will do in the forthcoming. Because our proofs and algorithmic ideas are mainly drawn from graph theory, this proves to be notationally more convenient.

3 Choosing a Good Elimination Ordering for the State Elimination Technique

The state elimination technique is an optimized version of the McNaughton-Yamada algorithm, avoiding the unnecessary computation of subexpressions. A detailed description of the state elimination algorithm can be found in [21]. Here we only introduce the necessary background and notations. For the hammock $H(A) = (Q, E, s, t)$ that is associated to a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ both algorithms compute regular expressions r^S_{jk} from the *regular expression matrix* $R^S = (r^S_{jk})_{j,k \in Q \cup \{s,t\}}$ satisfying $L(r^S_{jk}) = L^S_{jk}$, for every $j, k \in Q \cup \{s, t\}$ and a fixed ordering $S \subseteq Q$; it is convenient to write a total order on a finite set as a word, where the relative positions of the letters specify the order.

Since any path from vertex j to k whose internal vertices are in $S \cup \{i\}$ can be written as $L_{jk}^{S\cup\{i\}} = L_{jk}^{S} + L_{ji}^{S} \cdot (L_{ii}^{S})^{*} \cdot L_{ik}^{S}$, we are led to define the identity $r_{jk}^{S\cdot i} = r_{jk}^{S} + r_{ji}^{S} \cdot (r_{ii}^{S})^{*} \cdot r_{ik}^{S}$ on regular expressions, for every $j, k \in Q \cup \{s, t\}$ and $S \cdot i$ prefix of the ordered set Q, which is the basic recurrence of both algorithms. After applying these rules for all pairs (j, k) with $j, k \neq i$ for an inner vertex i it becomes isolated and thus can safely be eliminated. This explains the term *state elimination algorithm*, because during the computation of the expressions r_{jk}^{S} we are led to the hammock $H^{S}(A) = (Q \setminus S, E^{S}, s, t)$, with $E^{S} = \{ (j,k) \mid$ there is a path from vertex j to k in $H(A)$ with internal vertices from $S \}$. Observe that the choice of the elimination order on Q can greatly influence the size of the resulting regular expression r_{st}^{Q}. A further slight enhancement on the algorithm concerns the usage of the similarity relation.[3] With a straightforward implementation one can ensure that $r_{jk}^{S} = \emptyset$ if and only if $L_{jk}^{S} = \emptyset$.

The size of the regular expression resulting from applying the McNaughton-Yamada algorithm has been analyzed in [7]. There it was shown that the algorithm produces a regular expression of alphabetic width at most $|\Sigma| \cdot n \cdot 4^{n}$. Here, state elimination is better by a factor of n, paradoxically because we *enlarged* the automaton, in adding an $(n+1)$th state as single final state.

Theorem 2. *Let A be an n-state finite automaton. Then the state elimination algorithm produces for any ordering on the states a regular expression describing $L(A)$ of alphabetic width at most $|\Sigma| \cdot 4^{n}$.* $\square$

Previous accounts on choosing elimination orderings can be naturally put into two groups: In the first group, we find algorithms that have a tail-recursive specification, and are most easily implemented by an iterative program [3, 6, 7, 11, 17, 18], the others are based on the divide-and conquer paradigm [7, 12], suggesting a recursive implementation. We present a lemma that proves useful for designing algorithms in both groups. The lemma gives rise to two algorithms for choosing good elimination orderings yielding nontrivial performance guarantees for deterministic finite automata, one of which gives polynomial-size regular expressions for a restricted yet large class of finite automata.

3.1 The Main Lemma

As before, for a finite automaton A, let $H(A) = (Q, E, s, t)$ be the hammock associated with A, and let S denote a subset of Q. We begin with an observation on the expressions r_{jk}^{S} resulting from eliminating S in case the induced subgraph $H[S]$ falls apart into mutually disjoint components.

[3] Two regular expressions r and s are called *similar*, in symbols $r \cong s$, if r and s can be transformed into each other by repeatedly applying one of the following rules to their subexpressions: (1) $r + r \cong r$, (2) $(r + s) + t \cong r + (s + t)$, (3) $r + s \cong s + r$, (4) $r + \emptyset \cong r \cong \emptyset + r$, (5) $r \cdot \emptyset \cong \emptyset \cong \emptyset \cdot r$, (6) $r \cdot \epsilon \cong r \cong \epsilon \cdot r$, and (7) $\emptyset^{*} \cong \epsilon \cong \epsilon^{*}$. The first three rules above define the notion of similarity introduced by Brzozowski [1], and the remaining three have been added because of their usefulness in the context of converting regular expressions into finite automata.

Lemma 3. *Let $H = (Q, E, s, t)$ be a hammock. Assume $S \subseteq Q$ can be partitioned into two sets T_1 and T_2 such that the induced subgraph $H[S]$ falls apart into mutually disconnected components $H[T_1]$ and $H[T_2]$. Let j and k be vertices with $j, k \in (Q \setminus (T_1 \cup T_2)) \cup \{s, t\}$. Then for the expression obtained by elimination of the the vertices in T_1 followed by elimination of the vertices in T_2 holds $r_{jk}^{T_1 \cdot T_2} \cong r_{jk}^{T_1} + r_{jk}^{T_2}$.*

Proof. We prove the statement by induction on $|T_1| + |T_2|$. The induction is rooted at $|T_1| + |T_2| = 0$. For the case T_2 is empty, we have in general $r_{jk}^{T_1 T_2} = r_{jk}^{T_1} \cong r_{jk}^{T_1} + r_{jk}^{\epsilon}$, as desired. For the induction step, let $|T_1| + |T_2| = n$, with $T_2 \neq \emptyset$. Let t be the last element in T_2, that is, $T_2 = Tt$ for some prefix T of T_2. Then $r_{jk}^{T_1 T_2} \cong r_{jk}^{T_1 T} + r_{jt}^{T_1 T} \cdot (r_{tt}^{T_1 T})^* \cdot r_{tk}^{T_1 T}$. Since $|T_1| + |T| = n - 1$, for the first of the four subexpressions on the right-hand side the induction hypothesis applies: $r_{jk}^{T_1 T} \cong r_{jk}^{T_1} + r_{jk}^{T}$. For the last three subexpressions, we claim that $r_{jt}^{T_1 T} \cong r_{jt}^{T}$, as well as $(r_{tt}^{T_1 T})^* \cong (r_{tt}^{T})^*$, and $r_{tk}^{T_1 T} = r_{tk}^{T}$. We only prove the first congruence, the others are dealt with in a similar manner. It suffices to prove $r_{jt}^{T_1} \cong r_{jt}^{\epsilon}$, since the both sides of the former congruence are obtained from the latter by eliminating T, and state elimination preserves similarity of expressions. If there is an edge $(j, t) \in E$, then it is already described by r_{jt}^{ϵ}. It only remains to show that no further words are introduced by eliminating T_1. So we may as well assume that $(j, t) \notin E$ and prove the congruence for this case. This can be done as follows: Consider the subgraph $H[S]$. By assumption of the lemma, $t \in T_2$ is not reachable from any vertex in T_1, thus no walk from j to t can visit a vertex in T_1, and since there is no direct connection from j to t, the language $L_{jt}^{T_1}$ is empty. Every regular expression describing the empty set is similar to $\emptyset$, hence $r_{jt}^{T_1} \cong \emptyset$, provided $(j, t) \notin E$. This completes the proof of the congruence for this subexpression. Plugging in the four subexpression congruences we just found that $r_{jk}^{T_1 T_2} \cong r_{jk}^{T_1} + r_{jk}^{T} + r_{jt}^{T}(r_{tt}^{T})^* r_{tk}^{T} = r_{jk}^{T_1} + r_{jk}^{T_2}$. □

3.2 Eliminating Independent Sets

The following theorem shows that eliminating an independent set from the vertex set before eliminating the remaining vertices produces intermediate regular expressions which are short and easy to understand.

Lemma 4. *Let $H = (Q, E, s, t)$ be a hammock. Assume $I \subseteq Q$ is an independent set in H. Let j and k be vertices with $j, k \in (Q \setminus I) \cup \{s, t\}$. Then for the regular expression r_{jk}^{I} obtained after elimination of I holds $r_{jk}^{I} \cong r_{jk}^{\epsilon} + \sum_{i \in I} r_{ji}^{\epsilon} \cdot (r_{ii}^{\epsilon})^* \cdot r_{ik}^{\epsilon}$.*

Proof. By induction on $|I|$, making repeated use of Lemma 3: The statement holds true in the case $|I| = 1$. For $|I| > 1$, in the notation of Lemma 3, set $S = I$, let t be the last element in I, and assume that T is a suitable prefix such that $I = Tt$. Then $H[I]$ falls apart into mutually disjoint components $H[T]$ and $H[\{t\}]$. Thus, Lemma 3 is applicable, and $r_{jk}^{I} \cong r_{jk}^{T} + r_{jk}^{t} = r_{jk}^{T} + r_{jk}^{\epsilon} + r_{jt}^{\epsilon} \cdot (r_{tt}^{\epsilon})^* \cdot r_{tk}^{\epsilon}$. By induction hypothesis, $r_{jk}^{T} \cong r_{jk}^{\epsilon} + \sum_{i \in T} r_{ji}^{\epsilon} \cdot (r_{ii}^{\epsilon})^* \cdot r_{ik}^{\epsilon}$. Since the notion of similarity allows to suppress the multiple appearance of r_{jk}^{ϵ} in a sum of subexpressions, the expression $r^T_{jk} + r^\epsilon_{jk} + r^\epsilon_{jt} \cdot (r^\epsilon_{tt})^* \cdot r^\epsilon_{tk}$ is similar to the right hand side of the congruence in the statement of the lemma. □

The next observation is that we can use Lemma 4 repeatedly.

Lemma 5. *Let $H = (Q, E, s, t)$ be a hammock, and let S be an ordered subset of Q Assume $I \subseteq Q \setminus S$ is an independent set in H^S. Let j and k be vertices with $j, k \in (Q \setminus (S \cup I)) \cup \{s, t\}$. Then for the regular expression r^{SI}_{jk} obtained after elimination of SI holds $r^{SI}_{jk} \cong r^S_{jk} + \sum_{i \in I} r^S_{ji} \cdot \left(r^S_{ii}\right)^* \cdot r^S_{ik}$.* □

This gives an algorithm for computing a good elimination ordering as follows: Choose a large independent set I_1 in $H = H(A)$, then choose an independent set I_2 in H^{I_1}, choose an independent set I_3 in $H^{I_1 I_2}$, and so on. To estimate the performance of the independent set elimination approach, we have to find a large independent set I_{k+1} in the hammock $H^{I_1 I_2 \cdots I_k}$. The cardinality of the maximum independent set in some intermediate graph G^S obtained after eliminating $S = I_1 I_2 \ldots I_k$ can be estimated using Turán's Theorem from graph theory [20]. The latter gives an estimate in terms of the *average degree* of a symmetric digraph $G = (V, E)$, the latter being defined as $\overline{d}(G) = |E|/|V|$—recall that each unordered pair $\{u, v\}$ forming an "undirected edge" is counted as two edges in E.

Theorem 6 (Turán). *If G is a symmetric digraph of average degree $\overline{d}$ with n vertices, then G has an independent set of size at least $n/(\overline{d} + 1)$.*

In spite of the well known fact that finding a maximum independent set is computationally hard, the proof of the above theorem implies that such a large independent set can also be found efficiently using a simple greedy algorithm. Due to lack of space we have to omit the proof of the following theorem.

Theorem 7. *Let A be an n-state deterministic finite automaton with input alphabet Σ. Then there exists an ordering on the states such that the state elimination algorithm produces a regular expression describing $L(A)$ of alphabetic width at most $|\Sigma| \cdot n^{O(1)} \cdot 4^{c \cdot n}$, where $c = \frac{2|\Sigma| \cdot 2|\Sigma|^2 \cdot 2|\Sigma|^4}{(2|\Sigma|+1)(2|\Sigma|^2+1)(2|\Sigma|^4+1)}$.* □

For the case of a binary alphabet, we have $c = \frac{1024}{1485}$ and $4^c \doteq 2.601$, thus giving a worst-case upper bound of, say, $O(2.602^n)$. This appears reasonable at once in presence of a worst-case lower bound of γ^n for the case of deterministic finite automata over binary alphabets, proved recently in [10]. Here, $\gamma > 1$ is a fixed constant[4] that is independent of n.

[4] By tracking the size of the constants used in the chain of reductions used in that proof, one can deduce a concrete value for the constant γ. For alphabets of size $\ell \geq 3$, we get expression size at least $2^{\frac{\sqrt{\ell}(n-1)}{3 \cdot 2 \cdot (\ell+1)^2}}$, for infinitely many values of n. Here we exploited the fact from spectral graph theory that, using definitions and notation from [2], for the vertex expansion of ℓ-regular Ramanujan graphs G holds $g_G \geq h_G \geq \lambda_1/2 \geq \frac{\sqrt{\ell}}{2(\ell+1)}$, in particular for $\ell = 3$. Using a binary encoding that increases the size of the input deterministic finite automaton to $m = 10n$ whilst preserving

3.3 From Automata of Small Treewidth to Regular Expressions

We show that finite automata whose transition structure forms a graph of bounded undirected treewidth can be converted into regular expressions of polynomial size.

Definition 8. *Let $G = (V, E)$ be a digraph, and let $S \subseteq V$ be a set of vertices. A set of vertices X is a* balanced k-way separator *for S if the induced subgraph $G[S \setminus X]$ falls apart into k mutually disjoint subgraphs $G[T_i]$, for $1 \leq i \leq k$, with $0 \leq |T_i| \leq \frac{1}{2}|S \setminus X|$.*

It is known that for digraphs of undirected treewidth w, every nontrivial subset of the vertex set admits a small balanced k-way separator of size at most $w + 1$, for some k [19]. An elementary observation on sums of integers shows that we can always set $k = 3$, by grouping the disjoint subgraphs together in a suitable manner. Together with the mentioned result from [19], we thus have:

Lemma 9. *Let $G = (V, E)$ be a digraph of undirected treewidth at most w. Then for every subset S of V, there exists a balanced 3-way separator of size at most $w + 1$.* □

This separation property can be used to convert finite automata of small undirected treewidth into relatively short regular expressions:

Theorem 10. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an n-state* nondeterministic *finite automaton, H its associated hammock, and let w denote the undirected treewidth of $H[Q]$. Then the there exists a ordering on the states such that the state elimination algorithm produces a regular expression describing language $L(A)$ of alphabetic width at most $|\Sigma| \cdot n^{2w+2+\log 3}$.*

Proof. We devise a recursive algorithm for finding an elimination ordering such that the size of the resulting regular expression obeys the desired bound as follows: By Lemma 9 for each set of states $S \subseteq Q$, we can find a balanced 3-way separator X, such that $|X| \leq w + 1$, and the induced subgraph $H[S \setminus X]$ falls apart into three mutually disjoint subgraphs $H[T_i]$, for $1 \leq i \leq 3$. For each of the individual sets T_i, Lemma 3 ensures that for *every ordering*, $r_{jk}^{T_1 T_2 T_3} \cong \sum_{i=1}^{3} r_{jk}^{T_i}$, for all $j, k \in (Q \setminus (T_1 \cup T_2 \cup T_3)) \cup \{s, t\}$. Then we recursively compute an ordering for each T_i, placing a separator for $H[T_i]$ at the end of that ordering, and so on.

Since for each $S \subseteq Q$ the alphabetic width of r_{jk}^{S} is at most $4^{|X|} \sum_{i=1}^{3} \text{alph}(r_{jk}^{T_i})$, for some X, T_1, T_2, and T_3 with $|X| \leq w + 1$ and $|T_i| \leq \frac{1}{2}|S|$, for $1 \leq i \leq 3$. Moreover, the alphabetic width of the expression r_{jk}^{S} is bounded above by the recurrence $R(1) \leq 1$ and $R(n) \leq 4^{w+1} \cdot 3 \cdot R\left(\frac{n}{2}\right)$, for $n \geq 2$. We obtain $R(n) \leq 4^{(w+1)\log n} 3^{\log n}$. Applying the substitution σ increases the expression size by a factor of at most $|\Sigma|$. Thus we have an expression of alphabetic width $|\Sigma| \cdot n^{2w+2+\log 3}$ for the language $L(A)$. □

star-height, the very same lower bound (but still in terms of $n = \frac{1}{10}m$) is proved for binary alphabets. Thus we obtain $\gamma \doteq 1.013$ for alphabet size at least 3, and $\gamma \doteq 1.001$ for binary alphabets. This estimate is most likely very loose, since the main goal in in [10] was merely to bound the value of γ away from 1.

3.4 Eliminating Subgraphs of Small Treewidth

Now we present a fusion of our previous ideas: Instead of an independent set, we look for a large induced subgraph whose structure is "simple" in the sense that eliminating the states in the subgraph leads to a regular expression of moderate size. As we have seen in the previous section, one such example are induced subgraphs of small undirected treewidth. A very recent result states that every graph with bounded average degree has a large induced subgraph of treewidth at most two [5]:

Theorem 11. *Let G be a connected graph with average degree at most $\overline{d} \geq 2$. Then there is a polynomial-time algorithm which finds an induced subgraph with undirected treewidth at most two of size at least $\frac{3n}{\overline{d}+1}$.* □

This gives rise to an algorithm with improved performance guarantee.

Theorem 12. *Let A be an n-state deterministic finite automaton with input alphabet Σ. Then there exists a ordering on the states such that the state elimination algorithm produces a regular expression describing $L(A)$ of alphabetic width at most $|\Sigma| \cdot n^{O(1)} \cdot 4^{c \cdot n}$, where $c = \frac{2|\Sigma|-2}{2|\Sigma|+1}$.*

Proof. Let $H = H(A) = (Q, E, s, t)$ be the hammock associated with the automaton A. Note that the average outdegree of $H[Q]$ is at most $|\Sigma|$ so the average degree of its undirected version is at most $2|\Sigma|$. By Theorem 11, we can find a subset S of Q having size $\frac{3n}{2|\Sigma|+1} = (1-c) \cdot n$, for suitably chosen c, such that the induced subgraph $H[S]$ has undirected treewidth at most 2. The remaining states in $Q \setminus S$ are placed at the end of the elimination ordering. We set up a regular expression matrix $(r^{\epsilon}_{jk})_{j,k}$, whose rows j and columns k range over the set $(Q \setminus S) \cup \{s, t\}$. The algorithm from the proof of Theorem 10 can be used to compute an elimination ordering for S such that the set of walks from j to k using internal states only from $H[S]$ is described by the regular expression r^{S}_{jk}. One observes that, since this ordering does not depend on j or k, that the same result is obtained by eliminating S from the larger graph H by using that very ordering. As the size of the intermediate expressions after this phase is bounded by $|S|^{6+\log 3}$, and eliminating the remaining states in $Q \setminus S$ incurs a blow-up by a factor of $4^{c \cdot n}$, we obtain that the alphabet with of r^{Q}_{st} is at most $n^{O(1)} \cdot 4^{c \cdot n}$. Finally we apply the substitution σ to obtain a regular expression for $L(A)$ that has alphabetic width at most $|\Sigma| \cdot n^{O(1)} \cdot 4^{c \cdot n}$, for $c = (2|\Sigma| - 2)/(2|\Sigma| + 1)$. □

In the case of a binary input alphabet, we obtain that the maximum blow-up arising in the conversion from deterministic finite automata to regular expressions is at most $n^{O(1)} \cdot 4^{2/5 \cdot n}$, where $4^{2/5} \doteq 1.741$.

4 Language Operations and Regular Expression Size

Studying descriptional complexity of language operations on regular expressions was first suggested in [7]. Lower bounds for the intersection and shuffle, and a tight lower bound for complementation were found recently in [8–10]. We are able to contrast these negative results with a comparable upper bound for intersection. A similar approach works for the half-removal operation.

Theorem 13. *Let $L_1, L_2 \subseteq \Sigma^*$ be regular languages with alphabetic width at most m and n, respectively. Then* $\mathrm{alph}(L_1 \cap L_2) \leq |\Sigma| \cdot 2^{O\left(1+\log \frac{m}{n}\right)\min\{m,n\}}$. *Note that this bound is best possible for the case $m = \Theta(n)$ and $|\Sigma| = O(1)$.*

Proof. A regular expression of size m can be converted into an equivalent nondeterministic finite automaton A with at most $m+1$ states such that the digraph of the underlying transition structure has undirected treewidth at most two [13]—this nondeterministic finite automaton will in general have ϵ-transitions, but these do not cause any trouble when we treat them just like normal transitions. The construction ensures that the transition structure of the that automaton is a hammock with at most $m-1$ internal vertices.

Let A_1 and A_2 be finite automata thus obtained from suitable regular expressions of alphabetic width m and n describing the languages L_1 and L_2, respectively. Moreover, let Q_1 and Q_2 denote their respective state sets of A_1 and A_2, respectively. By applying the standard product construction for the intersection of regular languages, we obtain a nondeterministic finite automaton $A_1 \times A_2$ with $(m+1)(n+1)$ states accepting the language $L_1 \cap L_2$, by appropriately defining the initial state of $A_1 \times A_2$ and the accepting states of the product automaton. With G_1 and G_2 denoting the digraphs underlying each transition structure of the automata, the digraph underlying $A_1 \times A_2$ is (a subgraph of) the categorical product $G_1 \times G_2$. Let $H = H(A_1 \times A_2)$ denote the hammock associated with finite automaton $A_1 \times A_2$, where s and t are the distinguished vertices of H. The following claim is immediate from the definition of balanced 3-way separators (Definition 8) and the definition of categorical product:

Claim. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be digraphs, and $S_1 \subseteq V_1$, $S_2 \subseteq V_2$. Assume X is a balanced separator for S_1, such that the digraph $G[S_1 \setminus X]$ falls apart into the mutually disjoint subgraphs $G[T_i]$, for $1 \leq i \leq 3$, with $0 \leq |T_i| \leq \frac{1}{2}|S_1 \setminus X|$. Then $X \times S_2$ is a balanced 3-way separator for $S_1 \times S_2$ in the product graph $G_1 \times G_2$, and the digraph $(G_1 \times G_2)[(S_1 \setminus X) \times S_2]$ falls apart into the mutually disjoint subgraphs $G[T_i \times S_2]$, for $1 \leq i \leq 3$, with $0 \leq |T_i \times S_2| \leq \frac{1}{2}|(S_1 \setminus X) \times S_2|$. □

We proceed in a similar way as in the proof of Theorem 10, by recursively computing regular expressions $r_{jk}^{S_1 \times S_2}$ for $S_1 \subseteq Q_1$, $S_2 \subseteq Q_2$ and all $j, k \in ((Q_1 \times Q_2) \setminus (S_1 \times S_2)) \cup \{s, t\}$. This time we always choose a suitable separator according to the above stated claim. This is done as follows: If $|S_1| < |S_2|$, then exchange the roles of G_1 and G_2, and of S_1 and S_2, respectively. This is admissible by the symmetry of the categorical product. Afterwards, choose a 3-way separator X for $G_1[S_1]$ of size at most 3—recall that, by Lemma 9 such a separator exists, since both factor graphs have undirected treewidth at most 2. Let T_1, T_2, and T_3 be the disjoint subgraphs constituting $G_1[S_1 \setminus X]$ as given by Definition 8. Eliminating $(S_1 \setminus X) \times S_2$ gives regular expressions $r_{jk}^{(S_1 \setminus X) \times S_2}$, with j and k ranging over all states not in $(S_1 \setminus X) \times S_2$.

By the above claim and Lemma 3, we have $r_{jk}^{(S_1 \setminus X) \times S_2} \cong \sum_{i=1}^{3} r_{jk}^{T_i \times S_2}$. To recursively assign an elimination ordering to each of the subsets $T_i \times S_2$, we find next a balanced 3-way separator for the larger of the two graphs $G_1[T_i]$ and $G_2[S_2]$, which amounts to a corresponding separator in the product graph $G_1 \times G_2[T_i \times S_2]$, and recursively proceed to assign elimination orderings to such subsets until the subset sizes reach the value 1.

In order to get an upper bound on $\mathrm{alph}(L_1 \cap L_2) \le \mathrm{alph}(r_{st}^{S_1 \times S_2})$, define

$$A(\beta, \eta) = \max_{\substack{S_1 \subseteq V_1, |S_1| \le \beta \\ S_2 \subseteq V_2, |S_2| \le \eta}} \{ \, \mathrm{alph}(r_{jk}^{S_1 \times S_2}) \mid j, k \in ((Q_1 \times Q_2) \setminus (S_1 \times S_2)) \cup \{s, t\} \, \}.$$

An easy observation is that for the degenerate case, where S_1 and S_2 have both at most one element, we have $A(1, 1) \le 4$. An upper bound is obtained thus by solving the recurrence $A(\beta, \eta) = A(\eta, \beta)$, if $1 < \beta < \eta$, $A(\beta, \eta) = 4$, if $\beta = \eta = 1$, and $A(\beta, \eta) = 3 \cdot A\left(\left\lfloor \frac{\beta}{2} \right\rfloor, \eta\right) \cdot 4^{3\eta}$, otherwise. This leads to the stated bound on $\mathrm{alph}(L_1 \cap L_2)$. The analysis of the recurrence is omitted. $\square$

Basically the same technique can be used for the half-removal operation, defined as $\frac{1}{2}L = \{ \, x \in \Sigma^* \mid \text{there exists } y \in \Sigma^* \text{ with } |x| = |y| \text{ such that } xy \in L \, \}$. The state complexity of this operation was studied in [4]. The theorem reads as follows—due to lack of space we omit the proof.

Theorem 14. *Let $L \subseteq \Sigma^*$ be a regular language of alphabetic width at most n. Then* $\mathrm{alph}\left(\frac{1}{2}L\right) \le |\Sigma| \cdot 2^{O(n)}$. $\square$

Thus, the technique used above is applicable for certain language operations that can be implemented on nondeterministic finite automata using a special kind of product construction. But there are also limitations: For instance, the authors failed to use the above technique to produce a nontrivial upper bound for the shuffle of two regular languages.

References

1. J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.
2. F. R. K. Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional Conference Series in Mathematics.* American Mathematical Society, 1997.
3. M. Delgado and J. Morais. Approximation to the smallest regular expression for a given regular language. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *Proceedings of the 9th Conference on Implementation and Application of Automata*, volume 3317 of *LNCS*, pages 312–314, Kingston, Ontario, Canada, July 2004. Springer.
4. M. Domaratzki. State complexity of proportional removals. *Journal of Automata, Languages and Combinatorics*, 7(4):455–468, 2002.

5. K. Edwards and G. E. Farr. Planarization and fragmentability of some classes of graphs. *Discrete Mathematics*, 308(12):2396–2406, 2008.
6. A. Ehrenfeucht and H. P. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.
7. K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.
8. W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science*, LNCS, Turoń, Poland, August 2008. Springer. To appear.
9. W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In S. Albers and P. Weil, editors, *Proceedings of the 25th Symposium on Theoretical Aspects of Computer Science*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 325–336, Bordeaux, France, February 2008. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.
10. H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. In L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walkuwiewicz, editors, *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, Reykjavik, Iceland, July 2008. Springer. Accepted for publication.
11. H. Gruber and J. Johannsen. Optimal lower bounds on regular expression size using communication complexity. In R. Amadio, editor, *Proceedings of the 11th International Conference Foundations of Software Science and Computation Structures*, volume 4962 of *LNCS*, pages 273–286, Budapest, Hungary, March–April 2008. Springer.
12. Y.-S. Han and D. Wood. Obtaining shorter regular expressions from finite-state automata. *Theoretical Computer Science*, 370(1-3):110–120, 2007.
13. L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):140–162, 2003.
14. S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, Annals of Mathematics Studies, pages 3–42. Princeton University Press, 1956.
15. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.
16. H. V. McIntosh. REEX: A CONVERT program to realize the McNaughton-Yamada analysis algorithm. Technical Report AIM-153, MIT Artificial Intelligence Laboratory, January 1968.
17. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRA Transactions on Electronic Computers*, 9(1):39–47, 1960.
18. J. J. Morais, N. Moreira, and R. Reis. Acyclic automata with easy-to-find short regular expressions. In J. Farré, I. Litovsky, and S. Schmitz, editors, *Proceedings of the 10th Conference on Implementation and Application of Automata*, volume 3845 of *LNCS*, pages 349–350, Sophia Antipolis, France, June 2005. Springer.
19. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
20. P. Turán. On an extremal problem in graph theory (in Hungarian). *Matematicko Fizicki Lapok*, 48:436–452, 1941.
21. D. Wood. *Theory of Computation.* John Wilet & Sons, 1987.