International Journal of Foundations of Computer Science
Vol. 24, No. 8 (2013) 1255–1279
© World Scientific Publishing Company
DOI: 10.1142/S0129054113500330

PROVABLY SHORTER REGULAR EXPRESSIONS FROM FINITE AUTOMATA*

HERMANN GRUBER[†,‡] and MARKUS HOLZER[§]

Institut für Informatik, Universität Giessen
Arndtstrasse 2, D-35392 Germany
[‡] *hermann.gruber@informatik.uni-giessen.de*
[§]*holzer@informatik.uni-giessen.de*

Received 3 June 2009
Accepted 17 April 2013
Communicated by Masami Ito

Based on recent results from extremal graph theory, we prove that every n-state binary deterministic finite automaton can be converted into an equivalent regular expression of size $O(1.742^n)$ using state elimination. Furthermore, we give improved upper bounds on the language operations intersection and interleaving on regular expressions.

Keywords: Regular expression; state elimination; cycle rank; intersection; interleaving.

1. Introduction

A famous theorem due to Kleene [25] states that the regular languages admit two equivalent characterizations of entirely different nature, namely as the languages accepted by finite automata on the one hand, and as those described by regular expressions on the other hand. There are a few classical algorithms for converting finite automata into regular expressions. Those algorithms look different at first glance [6, 7, 28]. But, as Sakarovitch [32] pointed out, all of these approaches are more or less reformulations of the same underlying algorithmic idea: they can be recast as variations of the standard state elimination algorithm. The latter is found in most textbooks on automata theory, see, e.g., [36].

All of these algorithms have an upper bound of roughly 4^n on the size of the resulting regular expressions, the number n being the number of states in the given finite automaton. The desire to obtain shorter regular expressions than the 4^n upper bound can be traced back to the work by McNaughton and Yamada [28]. They observed that the choice of the ordering in which the states are eliminated

*This is a completely revised and expanded version of a paper presented at the 12th Conference on Developments in Language Theory (DLT) held in Kyoto, Japan, September 16–19, 2008.
[†]First author's current address: knowledgepark AG, Leonrodstrasse 68, D-80636 München, Germany.

would largely affect the size of the resulting regular expression. Subsequently, several heuristics for choosing good elimination orderings have been proposed [8, 18, 20, 28]. But none of these yields a provable increase in performance. Such improved upper bounds have been obtained by imposing severe restrictions on the structure of the given finite automata: significantly better upper bounds are known for directed acyclic [12], unary [13], planar [13], and directed acyclic series-parallel [29] finite automata. These improvements are partly attained using specialized algorithms that are different from state elimination. To summarize, previous approaches resulted either in heuristics for the general case without guaranteed benefit, or in algorithms with provable performance that apply only to constrained automata.

In the present work, we will perform a balancing act between these two extremes. The first main result is that deterministic finite automata over binary alphabets can be converted into regular expressions of size $O(1.742^n)$ by state elimination. The corresponding elimination ordering can be found in polynomial time. In fact, the same result applies more generally to given nondeterministic finite automata if the density of transitions is sufficiently low. Our result nicely contrasts with a classical result by Ehrenfeucht and Zeiger [12], who obtained a lower bound of $\Omega(2^n)$ if we allow alphabets of growing size, and with the recently obtained lower bound of $\Omega(c^n)$ for some $c > 1$ for binary alphabets [17]. Note that both lower bounds apply to given deterministic finite automata as input.

We will also identify a graph connectivity measure that lends itself for a nice parametrization. Namely, every n-state finite automaton of undirected cycle rank at most c can be converted into a regular expression of size about $4^c \cdot n$, instead of 4^n. We will use this parametrization to give significantly improved upper bounds on the cost of performing the intersection and interleaving operations on regular expressions. Interestingly, we can show that the size of the resulting expression is chiefly governed by the size of the *smaller* operand. If both operands are of roughly the same size, the new upper bounds are asymptotically optimal, as witnessed by the matching lower bounds from [17].

2. Definitions

We briefly recall some basic notions in formal language and automata theory — for a thorough treatment, the reader might want to consult a textbook such as [22]. In particular, let Σ be a finite alphabet and Σ^* be 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 $|\varepsilon| = 0$.

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, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq S$ is the set of accepting states. The *language accepted* by a finite automaton A is defined as $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \,\}$, where the transition function δ is extended to a function from $\delta : Q \times \Sigma^* \to 2^Q$ in the natural way, i.e., $\delta(q, \varepsilon) = \{q\}$, and

$\delta(q, aw) = \bigcup_{p \in \delta(q,a)} \delta(p, w)$, for $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$. A nondeterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ is a (*partial*) *deterministic finite automaton*, 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 (deterministic or nondeterministic) finite automata are *equivalent*, if they accept the same language.

It is well known that finite automata and regular expressions are equally powerful, i.e., for every finite automaton one can construct an equivalent regular expression and *vice versa*. The regular expressions over Σ are defined recursively in the usual way:[a] $\emptyset$, ε, and every letter a with $a \in \Sigma$ are regular expressions; and when 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: $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$, $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$, and $L(r_1^*) = L(r_1)^*$.

The *size*, or *alphabetic width*, of a regular expression r over the alphabet Σ, denoted by $\text{alph}(r)$, is defined as the total number of occurrences of letters of Σ in r. For a regular language L, we define its alphabetic width, $\text{alph}(L)$, as the minimum alphabetic width among all regular expressions describing L. Several other measures for the size of a regular expression have been proposed, see [13, 21]; it is known that they are all related by a linear factor. A concept of different nature is star height [11]. It measures the nesting depth of stars, rather than the length of a regular expression. It is straightforward to give an infinite family of regular languages of star height 1 over some fixed alphabet, while there are only finitely many regular expressions of any given length.

As with finite automata, the notion of equivalence is defined based on equality of the described language. Since there is evidence that equivalence of regular expressions is difficult to determine algorithmically [34], we use the weaker notion of *similarity* which is much easier to apply: 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) $r \cdot \varepsilon \cong r \cong \varepsilon \cdot r$, and (7) $\emptyset^* \cong \varepsilon \cong \varepsilon^*$. The first three rules above define the notion of similarity introduced by Brzozowski [5], and the remaining four have been added because of their usefulness in the context of converting regular expressions into finite automata.

In the remainder of this section we fix some notations from graph theory. A *directed graph*, or *digraph*, $G = (V, E)$ consists of a finite set of vertices V with an associated set of edges $E \subseteq V \times V$. If the edge relation E is symmetric, the graph is said to be *undirected*. Intuitively, an *undirected graph*, or just *graph*, is obtained from a digraph by forgetting the orientation of the original edges. Further

[a]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.

we assume familiarity with basic notions from elementary graph theory, such as (average) degree, (induced) subgraphs, paths, walks, and connected components. Definitions for missing notions are found in standard textbooks such as [9].

Treewidth is a measure of the structural complexity of graphs that recently gained popularity, first studied (albeit under a different name) by Halin [19]. This concept remained largely unnoticed until it was rediscovered by two independent groups of researchers [2, 31]. These researchers noticed that many hard computational problems became tractable for graphs of small treewidth. The definition reads as follows [9]:

Definition 1. *Let $G = (V, E)$ be a graph, and assume $\mathcal{V} = \{U_1, U_2, \ldots, U_r\}$ is a collection of subsets of V. A tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ is called a* tree decomposition, *if all of the following conditions hold:*

(1) *The collection $\mathcal{V}$ covers the vertex set of the graph G, in the sense that $V = \bigcup_{U \in \mathcal{V}} U$.*
(2) *For every edge $(u, v) \in E$, there is a tree node $U \in \mathcal{V}$ such that both u and v are in U.*
(3) *If two tree nodes U_1 and U_2 are connected in the tree by a path, then $U_1 \cap U_2$ is a subset of each tree node visited along this path.*

The *width* of a tree decomposition $(\mathcal{V}, \mathcal{E})$ is defined as $\max\{\, |U| - 1 \mid U \in \mathcal{V}\}$, and the *treewidth* of a graph G is defined as the minimum width among all tree decompositions for G and denoted by $tw(G)$.

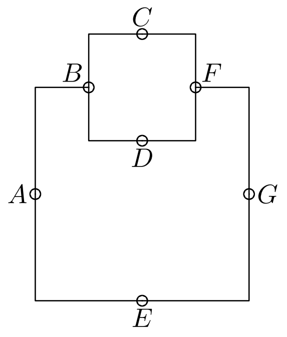
We illustrate the concept of treewidth on a small example.

Example 2. *Especially in electrical networks, we often encounter graphs that are recursively built from atomic units using series and parallel composition. An instance of such a series-parallel graph is depicted in Fig.* 1(*a*)*. Series-parallel graphs always admit a tree decomposition of width at most* 2*, compare* [9]*. Informally, the tree decomposition given in Fig.* 1(*b*) *exhibits the following pattern: a parallel composition gives rise to a tree node of cardinality* 2*, whose children are each of size* 3*. A series composition gives rise to a tree node of cardinality* 3*, whose children are each of size* 2*.*

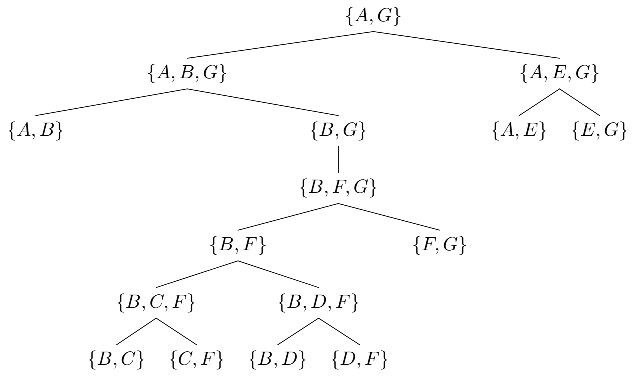
Intuitively, the treewidth of a graph measures its structural similarity to a tree. Not by coincidence, the treewidth of a tree is at most 1, whereas the treewidth of a complete graph on n vertices equals $n - 1$. The latter is the maximum possible treewidth among graphs on n vertices.

We shall need two further complexity measures on graphs — here for a (di)graph $G = (V, E)$ and $U \subseteq V$, the subgraph of G induced by U is denoted by $G[U]$.

Definition 3. *Let $G = (V, E)$ be a graph and $U \subseteq V$ be a set of vertices. A set of vertices S is a* balanced separator *for U if every component of $G[U \setminus S]$ contains*



(a) A series-parallel graph...



(b) ... and a tree decomposition.

Fig. 1. An example of a series-parallel graph and a tree decomposition of width 2 for it.

at most $\frac{1}{2}|U|$ *vertices. The* separator number *of* G*, denoted by* $s(G)$*, is defined as the maximum size, among all vertex subsets* $U \subseteq V$*, of the smallest balanced separator for* U*. Formally, let*

$$s(G) = \max_{U \subseteq V} \min_{S \subseteq U} \{\, |S| \mid S \textit{ is a balanced separator for } U \,\}.$$

Finally, we recall the notion of cycle rank, suggested by Eggan and Büchi in the course of investigating the star height of regular languages [11]. This concept was originally defined for directed graphs, compare [11, 17]. In this paper, we shall restrict our attention to the undirected cycle rank.

Definition 4. *The* (undirected) cycle rank *of a graph* $G = (V, E)$*, denoted by* $cr(G)$*, is inductively defined as follows:*

(1) *If* G *has no edges, then* $cr(G) = 0$*.*
(2) *If* G *consists of a single vertex with a self-loop, then* $cr(G) = 1$*.*
(3) *If* G *is connected and* $|V| \geq 2$*, then* $cr(G) = 1 + \min_{v \in V}\{cr(G - v)\}$*.*
(4) *f* G *is not connected, then* $cr(G)$ *equals the maximum cycle rank among all connected components of* G*.*

If we consider (undirected) graphs simply as symmetric digraphs, the notions of cycle rank and undirected cycle rank coincide on graphs. Nevertheless, we sometimes use the term *undirected cycle rank* to stress the difference between this and the concept that applies to digraphs. We note that undirected cycle rank is studied in the literature under many different names, such as ordered chromatic number, vertex ranking, tree-depth or minimum elimination tree height, see, e.g., [3, 4, 24, 30].

The following relation between separator number, treewidth, and undirected cycle rank is known.

Lemma 5 (Bodlaender *et al.* [4]) *Let G be an undirected graph. Then*

$$s(G) - 1 \leq tw(G) \leq cr(G) \leq 1 + s(G) \cdot \log n.$$

We mention that this result was stated in [4] in terms of elimination tree height, which is equal to the undirected cycle rank plus one for loop-free graphs.

3. Conversion Algorithms — The State Elimination Scheme

There are several algorithms to convert a finite automaton into a regular expression. Among the most well known ones are the McNaughton-Yamada algorithm and the state elimination technique. Although these are sometimes considered as different algorithms, at the core level they are essentially the same [32]. We briefly recall the state elimination technique, which is the basic algorithm for our considerations. In order to do this, we need to refine the notion of the computation relation of an NFA.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA. A triple $(p, a, q) \in Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ with $q \in \delta(p, a)$ is called a *transition* of A, which is more conveniently written as $p \xrightarrow{a} q$. For a subset U of the state set Q of the finite automaton A and an input word $w \in \Sigma^*$, we say that A can go on input w from state j *through* U to state k, if there is a computation on input w taking A from state j to k, without going through any state outside U. Here, by "going through a state," we mean both entering and leaving. More formally, A can go on input w from j through U to k if one of the following three cases applies:

- $j = k$ and $w = \varepsilon$, or
- $w \in \Sigma \cup \{\varepsilon\}$ and A has a transition $j \xrightarrow{w} k$, or
- $w = xa$ for some $x \in \Sigma^*$ and some $a \in \Sigma$ and there is a state r in U such that both A has a transition $r \xrightarrow{a} k$ and A can go on input x from j through U to r.

With the *rôles* of j, k and U fixed as above, we now define the language L_{jk}^U as the set of input words on which the automaton A can go from j to k through U. Then the essential fact is that $L(A) = \bigcup_{f \in F} L_{q_0 f}^Q$, which can be shown by induction on the length of a computation.

Now we present an algorithm scheme that became known as *state elimination.* Without loss of generality, we will assume that the given NFA A is *normalized* in the sense that A has state set $Q \cup \{s, t\}$ where s is the initial state and has no incoming transitions, and t is the sole accepting state and has no outgoing transitions. Both algorithms compute regular expressions r_{jk}^U satisfying $L(r_{jk}^U) = L_{jk}^U$, for every $j, k \in Q \cup \{s, t\}$ and $U \subseteq Q$. Recall, that L_{jk}^U refers to the set of all words on which A can go from state j to state k through U. Then we are interested in the expressions r_{st}^Q since $L(A) = L(r_{st}^Q)$. To this end we have to fix an ordering on the states of the automaton A. It is convenient to write any (total) order on a finite set U as a word, where the relative position of the letters naturally specifies the order. The (unique) ordering of the empty set is denoted by ε. Thus, the superscript U in r_{jk}^U refers to the total order induced by the set U and is assumed to be a word. In particular, we have $L(r^{\varepsilon}_{jk}) = L^{\emptyset}_{jk}$.

So we fix an order on Q. Let U be a prefix of Q, i.e., the order induced by U is compatible with the order on Q. Moreover we assume that i is the next state in the order after the states in U — in other words, that $U \cdot i$ is a prefix of Q. Observe that the set of words on which A can go from state j to state k through $U \cup \{i\}$ obeys

$$L^{U \cup \{i\}}_{jk} = L^{U}_{jk} \cup L^{U}_{ji} \cdot (L^{U}_{ii})^* \cdot L^{U}_{ik}. \tag{1}$$

So, we are led to the identity

$$r^{U \cdot i}_{jk} = r^{U}_{jk} + r^{U}_{ji} \cdot (r^{U}_{ii})^* \cdot r^{U}_{ik} \tag{2}$$

on regular expressions, for every $j, k \in Q \cup \{s, t\}$ and every $U \cdot i$ that is prefix of the ordered set Q. To complete the description of this algorithm the base cases are defined to be

$$r^{\varepsilon}_{jj} = \varepsilon + \sum_{\substack{j \xrightarrow{a} j \\ a \in \Sigma \cup \{\varepsilon\}}} a$$

and

$$r^{\varepsilon}_{jk} = \sum_{\substack{j \xrightarrow{a} k \\ a \in \Sigma \cup \{\varepsilon\}}} a, \text{ for } j \neq k.$$

Hence for an ordered subset U of Q, let $R^U = (r^U_{jk})_{j,k \in Q \cup \{s,t\}}$ denote the *regular expression matrix* obtained after eliminating U. The state elimination algorithm computes the regular expressions r^U_{jk}, for any prefix U of Q.

Since we are only interested in the expression r^Q_{st}, one only needs to generate those intermediate expressions that are eventually needed for the final result. As already observed by Brzozowski and McCluskey [6] one does not need to compute any expressions r^U_{jk} with $j \in U$ or $k \in U$. That is, after the state i has been added to the set U, we can discard both the ith row and the ith column from the regular expression matrix, hence the name "state elimination." A further slight enhancement of the algorithm concerns the use of the similarity relations we introduced earlier. With a straightforward implementation one can ensure that $r^U_{jk} = \emptyset$ if and only if $L^U_{jk} = \emptyset$. To illustrate the above description, we give a small example.

Example 6. *Imagine a software buffer supporting the actions "a" ("add work item") and "b" ("remove work item"), with a total capacity of n items. Let L_n denote the set of action sequences that result in an empty buffer and never cause the buffer to exceed its capacity. For illustration, a minimum DFA for L_4 is depicted in Fig. 2. The following two regular expressions*

$$(ab)^* + (ab)^* aa\, (ab + aa(ba)^* bb + bb(ab)^* aa + ba)^*\, bb(ab)^*$$

and

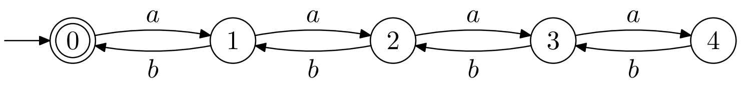


Fig. 2. A minimal DFA accepting L_4.

$$\left(a\left(a\left(a(ab)^*b\right)^*b\right)^*b\right)^*$$

both denote the language L_4*.*

Indeed, both expressions can, after normalizing the automaton, be obtained by state elimination. The first expression is obtained by eliminating the states $1, 3, 0, 4, 2$*, in this order. The second expression is obtained by eliminating the states in the order* $4, 3, 2, 1, 0$*. The relevant parts of the computation for the first expression are illustrated in Tables* 1*,* 2*, and* 3*; the analogous computation for the second expression is left as an easy exercise to the reader.*

The size of the regular expression resulting from applying the state elimination algorithm to a — not necessarily normalized — automaton has been analyzed in [13].

Table 1. The initial matrix with regular expression entries.

from \ to	s	t	0	1	2	3	4
s	ε	$\emptyset$	ε	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
t	$\emptyset$	ε	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
0	$\emptyset$	ε	ε	a	$\emptyset$	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$	b	ε	a	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$	$\emptyset$	b	ε	a	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	b	ε	a
4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	b	ε

Table 2. The matrix obtained after eliminating states 1 and 3.

from \ to	s	t	0	2	4
s	ε	$\emptyset$	ε	$\emptyset$	$\emptyset$
t	$\emptyset$	ε	$\emptyset$	$\emptyset$	$\emptyset$
0	$\emptyset$	ε	$\varepsilon + ab$	aa	$\emptyset$
2	$\emptyset$	$\emptyset$	bb	$\varepsilon + ab + ba$	aa
4	$\emptyset$	ε	$\emptyset$	b	$\varepsilon + ba$

Table 3. The matrix obtained after eliminating states 1, 3, 0, and 4.

from \ to	s	t	2
s	ε	$(ab)^*$	$(ab)^*aa$
t	$\emptyset$	ε	$\emptyset$
2	$\emptyset$	$bb(ab)^*$	$\varepsilon + ab + ba + aa(ab)^*bb + bb(ab)^*aa$

Their analysis is based on the equation $L(A) = \bigcup_{f \in F} L^Q_{q_0 f}$. This produces a regular expression of alphabetic width at most $|\Sigma| \cdot n \cdot 4^n$. In the form described above, state elimination normalizes the automaton first. Compared to the result from [13], we now save a factor of n:

Theorem 7. *Let A be an n-state NFA with input alphabet Σ. Then the state elimination algorithm produces, for every ordering on the states, a regular expression describing $L(A)$ of alphabetic width at most $|\Sigma| \cdot 4^n$.*

The state elimination algorithm we just saw should be called an *algorithm scheme* rather than an algorithm, since there remains some degree of indeterminacy: in each round, a new state is added to the working set U. Thus the order in which the states are eliminated, one after another, forms a parameter of an algorithm scheme, and every such *elimination ordering* gives rise to a different instance of that scheme. Already in 1960, McNaughton and Yamada observed that the choice of an elimination ordering can greatly influence the resulting regular expression size [28]. The running time of state elimination is in $O(|r| \cdot n^3)$, where r is the regular expression produced. The choice of an elimination ordering thus also affects the performance of the algorithm.

4. Undirected Cycle Rank and Elimination Orderings

We start our investigation of elimination orderings with some bad news: namely, by the work of Ehrenfeucht and Zeiger [12] there are normalized deterministic finite automata with $n + 2$ states, for which *every* elimination ordering will give a regular expression of the same size 4^n. This imposes severe limitations on what we can expect, in the worst case, from the state elimination scheme.

Nevertheless, a few heuristics to choose good elimination orderings have been proposed in the literature. McNaughton and Yamada [28] proposed to identify the states that "bear the most traffic," i.e., those vertices in the underlying graph with the highest degree, and to eliminate these states at last. Put another way, this amounts to ordering the states with respect to their degree. Delgado and Morais took this idea further [8]: instead of the vertex degree, a more elaborated weight function is used. This weight function takes into account the indegree, the outdegree and the current size of each intermediate expression. Since the respective values of these parameters change during state elimination, the weight function is recomputed after each elimination step, and the state of least weight is eliminated next.

Moreira and Reis [29] identified a class of finite automata for which the conversion problem is tractable, namely those whose underlying structure is an acyclic series-parallel digraph. Gulan and Fernau [18] proposed to search for such acyclic series-parallel digraphs as substructures in general finite automata. Han and Wood [20] suggested to identify serial or parallel composition patterns along which a given finite automaton could be recursively decomposed. All of the above heuristics can be easily implemented by means of elimination orderings. Of course, we cannot expect to find such nice substructures in too many finite automata: for instance, although the finite automaton from Example 6 has extremely simple structure, both of these two heuristics fail on that input; the same applies to Example 11 below.

Ellul *et al.* [13] encompassed the class of planar finite automata. They give a recursive algorithm with a nontrivial performance guarantee based on the planar separator theorem [26]. That procedure is seemingly more difficult to implement than a mere state elimination strategy. Below we will simplify and generalize the idea of using separators, and recast it in terms of state elimination.

Before we study the quest for good elimination orderings in more detail, we now proceed to the first main technical lemma of this section.

Lemma 8. *Let A be a normalized NFA with state set $\{s,t\}\cup Q$, and let G be the digraph underlying the transition structure of A. Assume $U\subseteq Q$ can be partitioned into two sets T_1 and T_2 such that the induced subgraph $G[U]$ falls apart into the mutually disconnected graphs $G[T_1]$ and $G[T_2]$. Let j and k be vertices with $j,k\in\{s,t\}\cup Q\setminus U$. Then for the expression $r_{jk}^{T_1\cdot T_2}$ obtained by elimination of the vertices in T_1 followed by elimination of the vertices in T_2 holds*

$$r_{jk}^{T_1\cdot T_2}\cong r_{jk}^{T_1}+r_{jk}^{T_2}. \tag{3}$$

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 $r_{jk}^{T_1T_2}=r_{jk}^{T_1}\cong r_{jk}^{T_1}+r_{jk}^{\varepsilon}$, as desired.

For the induction step, let $|T_1|+|T_2|=n$, with $T_2\neq\emptyset$. Let τ be the last element in T_2, that is, $T_2=T\tau$ for some prefix T of T_2. Then

$$r_{jk}^{T_1T_2}\cong r_{jk}^{T_1T}+r_{j\tau}^{T_1T}\cdot(r_{\tau\tau}^{T_1T})^{*}\cdot r_{\tau k}^{T_1T}. \tag{4}$$

Since $|T_1|+|T|=n-1$, for the first of the three subexpressions on the right-hand side the induction hypothesis applies: $r_{jk}^{T_1T}\cong r_{jk}^{T_1}+r_{jk}^{T}$. For the last three subexpressions, we claim that $r_{j\tau}^{T_1T}\cong r_{j\tau}^{T}$, as well as $\left(r_{\tau\tau}^{T_1T}\right)^{*}\cong\left(r_{\tau\tau}^{T}\right)^{*}$, and $r_{\tau k}^{T_1T}=r_{\tau k}^{T}$. We only prove the first and the second similarity congruence, because the third is proved symmetrically to the first.

It suffices to prove $r_{j\tau}^{T_1}\cong r_{j\tau}^{\varepsilon}$, since both sides of the congruence $r_{j\tau}^{T_1T}\cong r_{j\tau}^{T}$ are obtained from the mentioned one by eliminating T, and state elimination preserves similarity of expressions. If $L_{j\tau}^{\varepsilon}$ is nonempty, then these words are already described by $r_{j\tau}^{\varepsilon}$. It only remains to show that no further words are introduced by eliminating T_1. So, for simplicity of exposition, we may as well assume that $L_{j\tau}=\emptyset$ and prove the congruence for this case. This can be done as follows: consider the subgraph $G[U]$. By assumption of the lemma, $\tau\in T_2$ is not reachable from any vertex in T_1, thus the automaton cannot go from j to τ through any state in T_1 on any input at all, and since there is no direct connection from j to τ either, the language $L_{j\tau}^{T_1}$ is empty. Every regular expression describing the empty set is similar to the expression $\emptyset$, hence $r_{j\tau}^{T_1}\cong\emptyset$. This completes the proof of the congruence for this subexpression. Similar to above, the second congruence will follow once we have established that $r^{T_1}_{\tau\tau} \cong r^{\varepsilon}_{\tau\tau}$. This is easy to see since any computation path from τ to τ through T_1 (other than those implied by the empty input ε, or by a single self-loop directly returning back to τ itself) would witness that τ is connected to the subgraph T_1; but since τ is in T_2 this would contradict our assumption.

By plugging the three congruences for the subexpressions into Congruence 4, we obtain

$$r^{T_1T_2}_{jk} \cong r^{T_1}_{jk} + \left[r^{T}_{jk} + r^{T}_{j\tau}\left(r^{T}_{\tau\tau}\right)^{*} r^{T}_{\tau k}\right].$$

The right hand side is of course similar to $r^{T_1}_{jk} + r^{T_2}_{jk}$, by definition of the state elimination scheme. □

Remark. To prevent potential misunderstandings, we recall that, by our notational convention, Congruence 3 assumes an *arbitrary but fixed ordering* on T_1T_2, which naturally induces an ordering on all of its subsets, including T_1 and T_2. Equivalently, one can assume each an arbitrary but fixed ordering for the two subsets T_1 and T_2. Then these jointly define an ordering on T_1T_2.

The next theorem identifies a parameterized restriction on the transition structure of finite automata that gives rise to a large class of tractable instances of the conversion problem. This parameter is the undirected cycle rank as given in Definition 4. For convenience, we speak of the undirected cycle rank of a finite automaton A to refer to the undirected cycle rank of the underlying graph.[b]

Theorem 9. *Let A be an n-state NFA, and let c be a positive integer. If A has undirected cycle rank at most c, then there is a regular expression for $L(A)$ having size at most $|\Sigma| \cdot 4^c \cdot n$.*

The theorem is indeed a special case of the following slightly more general technical lemma, which we will prove instead:

Lemma 10. *Let A be a normalized NFA with state set $\{s, t\} \cup Q$, let G be its underlying graph, and let c be a positive integer. Let $U \subseteq Q$ be such that $G[U]$ has undirected cycle rank at most c. Then there is an elimination ordering for U which yields, for each pair j, k of states not in U, a regular expression r^{U}_{jk} of size at most $|\Sigma| \cdot 4^c \cdot |U|$.*

Proof. We prove the statement by induction on the number of states in the set U. In the base case $U = \{u\}$, the (undirected) cycle rank of $G[U]$ is at most c, and with the aid of Eq. (2), it is readily verified that

$$\mathrm{alph}\left(r^{U}_{jk}\right) \le 4 \cdot |\Sigma| \le |\Sigma| \cdot 4^c \cdot |U|;$$

recall that c is a positive integer.

[b]Beware that the undirected cycle rank of a finite automaton can largely differ from its *directed* cycle rank, the latter being used in [17].

For the induction step, we consider two cases: if the graph $G[U]$ is disconnected, then it falls apart into the components $C_1, C_2, \ldots, C_\ell$, each having at most $|U| - 1$ vertices. By induction hypothesis, for each component C_i, there is an ordering for C_i such that for each pair of states j, k not in U, the regular expression $r_{jk}^{C_i}$ resulting from this ordering satisfies

$$\operatorname{alph}\left(r_{jk}^{C_i}\right) \leq |\Sigma| \cdot |C_i| \cdot 4^c. \tag{5}$$

Here we can use the same constant c as for U, since the cycle rank of C_i is at most as large as the cycle rank of U. Fix such an ordering for each of the components $C_1, C_2, \ldots, C_\ell$, and take the sum of these expressions, that is, $\sum_{i=1}^{\ell} r_{jk}^{C_i}$. By applying Lemma 8 as often as needed, we see that the regular expression $r_{jk}^{C_1 C_2 \cdots C_\ell}$, which is obtained by eliminating according to the ordering $C_1 C_2 \cdots C_\ell$, is similar to the sum $\sum_{i=1}^{\ell} r_{jk}^{C_i}$ — recall that when applying Lemma 8, we assume arbitrary but fixed individual orderings on each set C_i. Using Inequality (5), the latter expression has alphabetic width at most

$$\sum_{i=1}^{\ell} |\Sigma| \cdot |C_i| \cdot 4^c = |\Sigma| \cdot |U| \cdot 4^c.$$

Otherwise, by the definition of cycle rank there must be a vertex u in $G[U]$ such that $G[U \setminus \{u\}]$ has cycle rank at most $c - 1$. By induction hypothesis, there is an ordering on $U \setminus \{u\}$ such that for each pair of states j, k not in $U \setminus \{u\}$, the regular expression $r_{jk}^{U \setminus \{u\}}$ resulting from this ordering satisfies

$$\operatorname{alph}\left(r_{jk}^{U \setminus \{u\}}\right) \leq |\Sigma| \cdot (|U| - 1) \cdot 4^{c-1}. \tag{6}$$

Finally, eliminating u as last state can incur a size increase by a factor of at most 4. This shows the desired inequality in the second case. □

The undirected cycle rank of a graph G can be defined equivalently as the minimum height among all elimination forests for G [4, 27]. We note that the post-order traversal of an elimination tree (of height at most k) yields the elimination ordering referred to in Lemma 10. Unfortunately, determining the undirected cycle rank is **NP**-complete in general, as proved by Pothen (cf. [3]). On the positive side, the undirected cycle rank problem can be approximated in polynomial time within a factor of $O((\log n)^{\frac{3}{2}})$ [14], and it is fixed-parameter tractable [3]. Such algorithms may be used for actually finding suitable elimination orderings, whose existence is implied by Lemma 10.

Theorem 9 has further algorithmic consequences: for instance, it is known [24] that planar graphs on n vertices have cycle rank in $O(\sqrt{n})$. With Theorem 9, one can show that an n-state planar finite automaton can be converted into a regular expression of size $2^{O(\sqrt{n})}$, just by state elimination. The same bound was obtained previously by Ellul *et al.* [13] using a more complicated algorithm. We shall see another application of Theorem 9 in the section that now follows. When investigating language operations on regular expressions in Sec. 6, we will present still more consequences of this theorem.

5. Converting DFAs into Regular Expressions: An Upper Bound

Although we cannot assume that DFAs in general have small undirected cycle rank, we can still try to look for large induced subgraphs that have small undirected cycle rank. The reason is here that after eliminating the vertex set U of an induced subgraph having undirected cycle rank c, the generated intermediate expressions are of size $4^c \cdot |U|$, in place of $4^{|U|}$. This advantage is more prominent if U is very large, while $c \ll |U|$: ideally, we want to find an induced subgraph of very low cycle rank, while being so large that it contains already a constant fraction of all vertices. If $|U| = \gamma \cdot n$, for some γ bounded away from zero, then eliminating the remaining $(1-\gamma)n$ states can increase the size of the intermediate expressions "only" by a factor of $4^{(1-\gamma)n}$. If the intermediate expressions resulting from eliminating the first, easier half are of size (say) $2^{o(n)}$, we will finally end up with a regular expression of size $2^{o(n)} \cdot 4^{(1-\gamma)n} = o(4^n)$. In such a case, we have thus reduced the original problem to a problem kernel of size $(1-\gamma)n$.

We will elaborate upon this idea in more detail in the following. To gain a bit more intuition, let us take a look at a relatively simple example first:

Example 11. *For illustrating the above said, consider the language*

$$(a_1b_1)^* \sqcup\!\sqcup (a_2b_2)^* \sqcup\!\sqcup (a_3b_3)^*,$$

where the interleaving*, or* shuffle*, of two languages L_1 and L_2 over alphabet Σ is*

$$L_1 \sqcup\!\sqcup L_2 = \{\, w \in \Sigma^* \mid w \in x \sqcup\!\sqcup y \text{ for some } x \in L_1 \text{ and } y \in L_2 \,\},$$

and the interleaving $x \sqcup\!\sqcup y$ of two words x and y is defined as the set of all words of the form $x_1y_1x_2y_2 \cdots x_ny_n$, where $x = x_1x_2\cdots x_n$, $y = y_1y_2\cdots y_n$ with $x_i, y_i \in \Sigma^*$, for $n \geq 1$ and $1 \leq i \leq n$. Note that in this definition, some of the subwords x_i and y_i can be empty.

This language can be accepted by a DFA over the state set $\{0,1\}^3$, and whose partial transition function is given such that input a_i sets the ith bit left of the rightmost bit of the current state from 0 to 1, and input b_i resets the ith bit, again counting from right to left, of the current state from 1 to 0. All other transitions are undefined. Formally, we can set out a matrix with regular expression entries as shown in Table 4.

The initial state is 000, which is also the single final state. Notice that the graph underlying this automaton is the 3-dimensional cube, with 8 vertices; and generalizing this example to $d \geq 3$, the underlying graph would be the d-dimensional hypercube, with 2^d many vertices. Normalizing this automaton amounts to adding two new states s and t and adding appropriate ε-transitions.

It is well known that the d-dimensional hypercube admits a large induced subgraph of very low undirected cycle rank: the hypercube is 2-colorable, and thus has an

Table 4. The initial matrix with regular expression entries. The rows and columns for the states s and t are omitted (all corresponding entries are equal to either ε or $\emptyset$).

from \ to	000	001	010	011	100	101	110	111
000	ε	a_1	a_2	$\emptyset$	a_3	$\emptyset$	$\emptyset$	$\emptyset$
001	b_1	ε	$\emptyset$	a_2	$\emptyset$	a_3	$\emptyset$	$\emptyset$
010	b_2	$\emptyset$	ε	a_1	$\emptyset$	$\emptyset$	a_3	$\emptyset$
011	$\emptyset$	b_2	b_1	ε	$\emptyset$	$\emptyset$	$\emptyset$	a_3
100	b_3	$\emptyset$	$\emptyset$	$\emptyset$	ε	a_1	a_2	$\emptyset$
101	$\emptyset$	b_3	$\emptyset$	$\emptyset$	b_1	ε	$\emptyset$	a_2
110	$\emptyset$	$\emptyset$	b_3	$\emptyset$	b_2	$\emptyset$	ε	a_1
111	$\emptyset$	$\emptyset$	$\emptyset$	b_3	$\emptyset$	b_2	b_1	ε

independent set that contains at least half of the vertices. Notice that an independent set always has undirected cycle rank at most 1*. For instance, if we eliminate the set* $U = \{001, 010, 100, 111\}$ *first, we end up with a matrix having entries as shown in Table* 5*. Not surprisingly, for the resulting regular expressions holds*

$$r_{jk}^{U} \cong r_{jk}^{001} + r_{jk}^{010} + r_{jk}^{100} + r_{jk}^{111}.$$

This can also be seen by applying Lemma 8 *as often as needed.*

Of course, the resulting matrix has many more nontrivial entries than before. But the largest entry has alphabetic width only 6*, although we already eliminated half of the original number of states. Continuing the elimination with the matrix from Table* 5 *we will end up with a regular expression of size at most* $6 \cdot 4^{|Q \setminus U|} \leq 1536$*,* even for the worst ordering *of the remaining states in* $Q \setminus U$*. Now we can again look for a large independent set in the resulting matrix. Yet a look at Table* 5 *shows that, in our small example, all independent sets in the resulting graph are only of size* 1*.*

While the bound 1536 *may appear large at first glance, recall that the upper bound predicted by Lemma* 7 *would be as huge as* $4^8 = 65536$*. Also, notice that the two bounds are both overestimations of the actual resulting expression size.*

The above example suggests to look out for large independent sets in the underlying graph, and to eliminate these first. We cannot expect a large independent set in the graph underlying the DFA if the alphabet size is not fixed: in the DFAs exhibiting worst-case behavior investigated by Ehrenfeucht and Zeiger [12], the underlying graph has no independent sets of size greater than 1. But once we require constant alphabet size, the graph underlying the DFA is sparse, i.e., it can have only a linear number of edges. A classical theorem in extremal graph theory due to Turán [35] states that sparse graphs do always contain large independent sets:

Theorem 12 (Turán [35]) *If* G *is an* n*-vertex graph of average degree* $\overline{d}$*, then* G *admits an independent set having at least* $\frac{n}{\overline{d}+1}$ *many vertices.*

By generalizing the analysis carried out in Example 11 to the case of an independent set U of size $|U| = \frac{n}{\overline{d}+1}$, this enables us to give a general upper bound. The

Table 5. The residual matrix with regular expression entries, after elimination of the independent set $U = \{001, 010, 100, 111\}$. The rows and columns for the states s and t are again omitted (in this particular example, all corresponding entries are still equal to either ε or $\emptyset$).

from \ to	000	011	101	110
000	$\varepsilon + a_1b_1 + a_2b_2 + a_3b_3$	$a_1a_2 + a_2a_1$	$a_1a_3 + a_3a_1$	$a_2a_3 + a_3a_2$
011	$b_1b_2 + b_2b_1$	$\varepsilon + a_3b_3 + b_1a_3 + b_2a_2$	$a_3b_2 + b_2a_3$	$a_3b_1 + b_1a_3$
101	$b_1b_3 + b_3b_1$	$a_2b_3 + b_3a_2$	$\varepsilon + a_2b_2 + b_1a_1 + b_3a_3$	$a_2b_1 + b_1a_2$
110	$b_2b_3 + b_3b_2$	$a_1b_3 + b_3a_1$	$a_1b_2 + b_2a_1$	$\varepsilon + a_1b_1 + b_2a_2 + b_3a_3$

proof of the following theorem is based on the fact that eliminating an independent set can cause only a bounded increase of the maximum outdegree of the underlying digraph, and thus we can apply Turán's Theorem again for the resulting digraph.

Theorem 13. *Let A be an n-state DFA with input alphabet Σ of constant size. Then there is an elimination ordering which yields a regular expression for $L(A)$ having size at most $n^{O(1)} \cdot 4^{\alpha n}$, with $\alpha = \frac{2|\Sigma| \cdot 2|\Sigma|^2 \cdot 2|\Sigma|^4}{(2|\Sigma|+1)(2|\Sigma|^2+1)(2|\Sigma|^4+1)}$. This ordering can be computed in time polynomial in n.*

We shall not report the details of the proof here, as we shall derive a better bound in the following. Recall that our basic idea is that we want to look for large induced subgraphs whose elimination is cheap in the sense that the intermediate expressions are of size $2^{o(n)}$. For independent sets, which have cycle rank at most 1, this intermediate size is indeed at most $O(n)$, as implied by Lemma 10. Thus we can safely look for induced subgraphs with more complex structure allowed. Alon *et al.* [1] found a generalization of Turán's theorem. Although they used the concept of degeneracy, and not treewidth, a special case of their result can be phrased in terms of treewidth as follows:

Theorem 14 (Alon *et al.* [1]) *If G is an n-vertex graph of average degree at most $\overline{d}$, with $\overline{d} \geq 2$, then G admits an induced subgraph of treewidth at most 1 that has at least $\frac{2n}{\overline{d}+1}$ many vertices.*

We point out that independent sets are exactly the induced subgraphs of treewidth zero. The following recent result, due to Edwards and Farr [10], provides a nice analog to Turán's Theorem and to Theorem 14:

Theorem 15 (Edwards and Farr [10]) *If G is an n-vertex graph of average degree at most $\overline{d}$, with $\overline{d} \geq 2$, then G admits an induced subgraph of treewidth at most 2 that has at least $\frac{3n}{\overline{d}+1}$ many vertices.*

The interpretation of these extremal theorems in terms of treewidth appears to be new, since treewidth is not mentioned in any of the papers [1, 10, 35]. Now we are ready to state the main result of this section.

Theorem 16. *Let A be an n-state DFA with input alphabet Σ. Then there is an elimination ordering which yields a regular expression for $L(A)$ having size at most $4 \cdot |\Sigma| \cdot n^7 \cdot 4^{\alpha n}$, with $\alpha = 1 - \frac{3}{2 \cdot |\Sigma| + 1}$. In particular, for $|\Sigma| = 2$, this bound is in $O(1.742^n)$.*

Proof. Assume the given DFA has state set Q. In a first step, we normalize the DFA by adding a new initial state s as well as a single new final state t and connecting them with Q appropriately. Note that, while normalizing might increase the number of edges in the undirected graph G underlying the automaton, this process does not affect the average degree of the induced subgraph $G[Q]$, since all new edges begin or end outside the set Q. Hence, the average degree of $G[Q]$ is at most $2 \cdot |\Sigma|$, both before and after the normalization.

To find a good elimination ordering, we choose to eliminate first the vertex set U of a large induced subgraph of treewidth at most 2 from $G[Q]$. Since $2 \cdot |\Sigma|$ constitutes an upper bound on the average degree of $G[Q]$, by Theorem 15 we can choose U to be of cardinality at least $\frac{3n}{2 \cdot |\Sigma|+1} = (1-\alpha)n$.

Recall from Lemma 5 that graphs on n vertices of treewidth at most k have undirected cycle rank at most $1 + (k+1)\log n$. In this way, we get $cr(G[U]) \leq 1 + 3\log|U|$. Let the latter number be denoted by c. By Lemma 10, the set U can be ordered in a way such that for all j, k not in U holds $\text{alph}\left(r_{jk}^U\right) \leq |\Sigma| \cdot 4^c \cdot |U|$.

Now we continue with eliminating the remaining states, by choosing an arbitrary ordering for the states in $\overline{U} = Q \setminus U$. During the second phase, eliminating a state in $\overline{U}$ can increase the size of the intermediate expressions by a factor of at most 4 each time. Thus, we have:

$$\text{alph}\left(r_{st}^Q\right) \leq |\Sigma| \cdot 4^c \cdot |U| \cdot 4^{\overline{U}}.$$

With $c \leq 1 + 3\log|U|$, we obtain

$$4^c = 2^{2\cdot(1+3\log|U|)} \leq 4 \cdot |U|^6.$$

Combined with the inequality $\left|\overline{U}\right| = n - |U| \leq \alpha n$, we get an upper bound of

$$\text{alph}\left(r_{st}^Q\right) \leq |\Sigma| \cdot 4 \cdot |U|^6 \cdot |U| \cdot 4^{\alpha n} \leq 4 \cdot |\Sigma| \cdot n^7 \cdot 4^{\alpha n}.$$

For $|\Sigma| = 2$, we have $4^\alpha = 4^{\frac{2}{5}} \doteq 1.7411$, and thus $n^7 \cdot 4^{\frac{2}{5}n} \in O(1.742^n)$. $\square$

The reader may find that there is no apparent reason to stop at this point. For instance, we could further generalize the above approach and look for large induced *planar* subgraphs. For, it can be proved that eliminating an induced planar subgraph still yields intermediate regular expressions of size $2^{o(n)}$. Yet it is impossible to guarantee essentially larger size when allowing planar induced subgraphs. The above mentioned work [10] discusses this issue at greater depth.

6. Alphabetic Width of Intersection and Interleaving

We turn to questions regarding the succinctness of regular expressions with respect to various language operations. This topic has recently received some attention — not only because of its fundamental nature, but also because of applications in XML processing — see, e.g., [13, 16, 17]. We shall focus on the operations intersection and interleaving. Recent results [16, 17] imply that the intersection operation has exponential cost. Similar results are shown in [15, 17] for the interleaving operation. The currently best known lower bounds are derived in [17] — here the two-parameter function $\text{alph}(\circ, m, n)$, for some binary language operation $\circ$, is defined as the maximum alphabetic width of $L(r_2) \circ L(r_2)$, where the maximum is taken over all regular expressions r_1 and r_2 having sizes at most m and n, respectively.

Theorem 17 (Gruber and Holzer [17]) *There exists a constant $c > 1$ such that for all $m \leq n$ holds $\mathrm{alph}(\cap, m, n) = c^m$ and $\mathrm{alph}(ш, m, n) = c^m$, and this holds for all alphabets of size at least 2.*

As detailed by Ellul *et al.* [13], the naïve idea of converting the operand expressions over Σ into finite automata, performing a product construction, and converting the result back to a finite automaton gives a preliminary upper bound of

$$|\Sigma| \cdot 4^{(m+1)(n+1)} \leq \max(m, n) \cdot 4^{(m+1)(n+1)} \leq d^{mn}, \text{ for some constant } d.$$

As we shall see, the easy upper bound by Ellul *et al.* [13] can be substantially improved by choosing an appropriate elimination ordering according to some graph-theoretic properties, similar to the considerations carried out in Sec. 4. We need to introduce a notion of products on graphs first.

Definition 18. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The categorical product $G_1 \otimes G_2$ is defined as the graph with vertex set $V_1 \times V_2$ and edge set $\{\, \{(u_1, u_2), (v_1, v_2)\} \mid \{u_1, v_1\} \in E_1$ and $\{u_2, v_2\} \in E_2 \,\}$.*

Observe that this product on graphs bears a striking similarity to the standard product construction on finite automata for realizing the intersection of regular languages, compare [22]. We have seen in Theorem 9, that the connectivity of the graph underlying a finite automaton can largely affect the complexity of the conversion problem into regular expressions. We thus inspect next how the connectivity of graphs evolves under taking such products.

Theorem 19. *Let G_1 and G_2 be two graphs, each having separator number at most k. Assume G_1 and G_2 have m and n vertices, respectively, with $m \leq n$. Then for the undirected cycle rank of the categorical product $G_1 \otimes G_2$ holds:*

$$cr(G_1 \otimes G_2) < k \cdot m \cdot \left(\log \frac{n}{m} + 4\right).$$

Proof. The basic idea is to search in the product graph for certain balanced separators, whose existence is implied by the presence of balanced separators in the factor graphs. If we search for a balanced separator of size at most k in the larger factor graph (say) G_2, this separator gives rise to a balanced separator of size only $k \cdot m$ in the product graph. In contrast, the separator halves the maximum size among the connected components of the larger factor graph, and also of the product graph. This idea is illustrated in Fig. 3. We proceed by recursively halving the parts of G_2, until the resulting parts become smaller than $|V_1| = m$. Then we look for separators in G_1, and proceed recursively until the size of the parts has sufficiently decreased such that we switch again, to continue working with the parts of G_2, and so on.

In order to turn this idea into a rigorous proof, we start off with a few observations regarding the categorical product of graphs:

(1) First, the categorical product is commutative in the sense that $G_1 \otimes G_2$ is isomorphic to $G_2 \otimes G_1$.

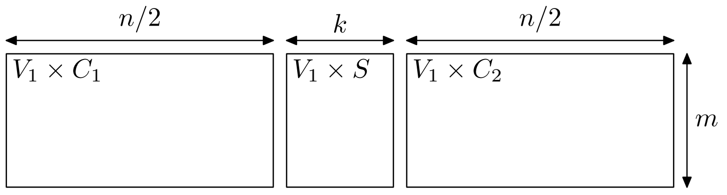


Fig. 3. Schematic drawing of the product graph $G_1 \otimes G_2$ for the case where G_1 has vertex set V_1 of size m, G_2 has vertex set $V_2 = C_1 \cup C_2 \cup S$ of size n, with C_1 and C_2 separated by S in G_2. Looking for a separator in G_2 induces a much smaller balanced separator for $G_1 \otimes G_2$ if $m \ll n$.

(2) Second, for $U_1 \subseteq V_1$ and $U_2 \subseteq V_2$, the subgraph of $G_1 \otimes G_2$ induced by $U_1 \times U_2$ is the same as the categorical product of the induced subgraphs $G_1[U_1]$ and $G_2[U_2]$. In symbols, we have the equation

$$(G_1 \otimes G_2)[U_1 \times U_2] = (G_1[U_1]) \otimes (G_2[U_2]). \tag{7}$$

(3) Third, if $S_1 \subseteq V_1$ is a balanced separator for G_1, then $S_1 \times V_2$ is a balanced separator for $G_1 \otimes G_2$. This can be seen as follows: each walk

$$\left(v_{i_1}^{(1)}, v_{j_1}^{(2)}\right) \to \cdots \to \left(v_{i_2}^{(1)}, v_{j_2}^{(2)}\right) \to \cdots \to \left(v_{i_k}^{(1)}, v_{j_k}^{(2)}\right)$$

in the product graph $G_1 \otimes G_2$ naturally projects down to a corresponding walk

$$v_{i_1}^{(1)} \to \cdots \to v_{i_2}^{(1)} \to \cdots \to v_{i_k}^{(1)}$$

in G_1. In the contrapositive, if G_1 has the property that every walk in G_1 connecting $v_{i_1}^{(1)}$ to $v_{i_k}^{(1)}$ passes through S, then $G_1 \otimes G_2$ has the property that every walk in $G_1 \otimes G_2$ connecting a vertex of the form $\left(v_{i_1}^{(1)}, w^{(2)}\right)$ to a vertex of the form $\left(v_{i_k}^{(1)}, x^{(2)}\right)$, with $w^{(2)}, x^{(2)} \in V_2$, necessarily also visits a vertex from $S \times V_2$. In particular, if $G_1 - S$ has the components $C_1, C_2, \ldots, C_\ell$, then the sets $C_1 \times V_2, C_2 \times V_2, \ldots, C_\ell \times V_2$ are subsets of pairwise different components in the product graph. If S is a balanced separator for G_1, none of the components in $G_1 - S$ can have size larger than $\frac{1}{2}m$; accordingly none of the components in $(G_1 \otimes G_2)[(V_1 \setminus S) \times V_2]$ can have size larger than $\frac{1}{2}m \cdot n$, and thus $S_1 \times V_2$ is a balanced separator for the product graph.

(4) Lastly, combining the second with the third observation, we see that if S is a balanced separator for $U_1 \subseteq V_1$ in G_1, then $S \times U_2$ is a balanced separator for $U_1 \times U_2$ in the product graph $G_1 \otimes G_2$.

At this point, we have collected enough information to put forward a recurrence. To this end, for real numbers $0 \leq \beta \leq m$ and $0 \leq \eta \leq n$, let $cr(\beta, \eta)$ denote the maximum cycle rank among the induced subgraphs $G[U_1 \times U_2]$, where the maximum is taken subject to $U_1 \subseteq V_1$, $|U_1| \leq \beta$ and $U_2 \subseteq V_2$, $|U_2| \leq \eta$. Then, clearly we have $cr(G_1 \otimes G_2) = cr(m, n)$.

Now the function $cr(\beta, \eta)$ can be bounded above using the following recurrence, as we explain below:

$$cr(\beta,\eta) \leq \begin{cases} 1 & \text{for } \beta, \eta < 2, \\ cr(\eta,\beta) & \text{for } 2 \leq \eta < \beta, \\ k\beta + cr\big(\beta, \frac{\eta}{2}\big) & \text{otherwise.} \end{cases} \tag{8}$$

The base case of the recurrence is justified by the fact that the cycle rank of any graph is of course bounded above by its number of vertices. The second case is correct because the categorical product is commutative, and taking products commutes with taking induced subgraphs of its factors in the sense of Eq. (7). Correctness of the third case of the recurrence is explained as follows: consider an induced subgraph G' of the form $G' = G_1[U_1] \otimes G_2[U_2]$, with $U_1 \leq \beta$ and $U_2 \leq \eta$, and $\beta \leq \eta$. As the graph $G_2[U_2]$ admits a balanced separator S of size at most k, the product G' admits a balanced separator of size at most $k \cdot |U_1| \leq k \cdot \beta$. Since removing a set of vertices of size $|U_1 \times S|$ from G' can decrease its cycle rank at most by that number, we have

$$cr(G') \leq |U_1 \times S| + cr(G' - S) \leq k \cdot \beta + cr(G' \setminus S).$$

If the components of $G_2[U_2 \setminus S]$ are denoted by $C_1, C_2, \dots C_\ell$, then by the definition of cycle rank

$$cr(G' - S) = \max_{1 \leq i \leq \ell} cr(G[U_1 \times C_i]).$$

Finally, because each C_i has cardinality at most $\frac{\eta}{2}$, each subgraph $G[U_1 \times C_i]$ can have cycle rank at most $cr(\frac{\eta}{2}, \beta)$. Altogether, this shows that the Recurrence (8) gives a correct upper bound.

For the analysis of this recurrence, assume without loss of generality that $\beta \leq \eta$. We observe first that when evaluating cr, eventually the parameters β and η are decreased in alternating order with each recursive call. Namely, for the parameter range $1 < \beta \leq \eta < 2\beta$, we see by partial unrolling that

$$\begin{aligned} cr(\beta,\eta) &\leq k \cdot \beta + cr\Big(\beta, \frac{\eta}{2}\Big) \\ &\leq k \cdot \beta + cr\Big(\frac{\eta}{2}, \beta\Big) \\ &\leq k \cdot \beta + \frac{1}{2}k \cdot \eta + cr\left(\frac{\eta}{2}, \frac{\beta}{2}\right) \\ &\leq k \cdot \left(\beta + \frac{1}{2}\eta\right) + cr\left(\frac{\beta}{2}, \frac{\eta}{2}\right) \\ &< 2k \cdot \beta + cr\left(\frac{\beta}{2}, \frac{\eta}{2}\right) . \end{aligned}$$

In this way, we have halved both parameter values appearing in the recurring expression. By unrolling the recurrence thus obtained and simplifying, we obtain for this range of parameter values:

$$cr(\beta,\eta) \leq \sum_{0\leq i<\log\beta} \frac{2k\cdot\beta}{2^i} + cr(1,1) < \sum_{i=0}^{\infty} \frac{2k\cdot\beta}{2^i} = 4k\cdot\beta.$$

We can thus keep for later reference that

$$cr(\beta,\eta) < 4k\cdot\beta, \text{ for } 1<\beta\leq\eta<2\beta. \tag{9}$$

It remains to reduce the case $\eta \geq 2\beta$ to the balanced case we just discussed. To this end, let $x = \left\lfloor \log\left(\frac{\eta}{\beta}\right)\right\rfloor$ denote the integer part of $\log\left(\frac{\eta}{\beta}\right)$. When applying the Recurrence (8) for x times, the second parameter always remains larger than the first, until we reach the following inequality:

$$cr(\beta,\eta) \leq k\cdot\beta\cdot x + cr(\beta, 2^{-x}\cdot\eta). \tag{10}$$

Now let y denote the fractional part of $\log\frac{\eta}{\beta}$. Then $2^{-x-y} = \frac{\beta}{\eta}$, or, equivalently, $2^{-x}\cdot\eta = 2^y\cdot\beta$. Using $0\leq y<1$, we get $\beta \leq 2^{-x}\cdot\eta < 2\beta$, and thus we can apply Inequality (9) to estimate the remaining recurrent expression on the right-hand-side of Inequality (10) and get:

$$\begin{aligned} cr(\beta,\eta) &\leq k\cdot\beta\cdot x + cr(\beta,2^{-x}\cdot\eta) \\ &< k\cdot\beta\cdot x + 4k\cdot\beta \\ &= k\cdot\beta\cdot\left(\log\frac{\eta}{\beta}+4\right). \end{aligned}$$

Since $cr(G_1\otimes G_2) = cr(m,n)$, the proof is completed. □

Now an upper bound for the alphabetic width of the intersection operation can be derived as follows:

Theorem 20. *There exists a constant d such that for all $m\leq n$, and for arbitrary alphabets,* $\text{alph}(\cap,m,n) \leq n\cdot d^{m\cdot\left(1+\log\frac{n}{m}\right)}$ *holds.*

Proof. Assume we are given two regular expressions r_1 and r_2 over a common alphabet Σ, of size m and n, respectively. These expressions can be transformed into equivalent NFAs A_1 and A_2 having $m+1$ and $n+1$ states using the conversion algorithm given in [23]. As observed by the authors in [17], the graphs G_1 and G_2 underlying the resulting automata are of treewidth at most 2, which is a feature of that particular conversion algorithm.

We apply the product construction to A_1 and A_2 to obtain an NFA $A_1\otimes A_2$ with $(m+1)(n+1)$ states accepting the intersection of $L(r_1)$ and $L(r_2)$.

Since the possibly present ε-transitions cause a minor technical issue, we briefly recall the details of this construction. Given two NFAs $A_i = (Q_i,\Sigma,q_{0,i},\delta_i,F_i)$, for $i\in\{1,2\}$, let $A_1\otimes A_2 = (Q_1\times Q_2,\Sigma,(q_{0,1},q_{0,2}),\delta,F_1\times F_2)$, where the transition function δ is defined as follows: transitions on letters $a\in\Sigma$ are given by $(q_1,s_2)\in\delta((p_1,r_2),a)$ iff both $q_1\in\delta_1(p_1,a)$ and $s_2\in\delta_2(s_2,a)$. Transitions on the empty word are given by $(q_1, s_2) \in \delta((p_1, r_2), \varepsilon)$ iff both $q_1 \in \{p_1\} \cup \delta_1(p_1, \varepsilon)$ and $s_2 \in \{r_2\} \cup \delta_2(r_2, \varepsilon)$. It is easy to see that $A_1 \otimes A_2$ accepts the intersection $L(A_1) \cap L(A_2)$.

Now we are interested in the cycle rank of the undirected graph underlying the automaton $A_1 \otimes A_2$. Observe that the latter is a subgraph of the product $G'_1 \otimes G'_2$, where the graph G'_i is obtained from G_i, for $i = 1, 2$, by adding a self-loop to each vertex. The graphs G_i have treewidth at most 2, and by Lemma 5 their separator number is at most 3 each. Adding self-loops to a graph does not increase its separator number. Thus the factors G'_1 and G'_2 still have separator number at most 3.

Now we are in a position allowing us to apply Theorem 19, and we deduce that there is some constant d' such that the product graph $G'_1 \otimes G'_2$ has cycle rank at most $d' \cdot m \log \frac{n}{m}$. Recall that Theorem 9 states that all NFAs of low cycle rank can be converted into sufficiently short regular expressions. In our case, the theorem implies that

$$\operatorname{alph}(L(A_1 \times A_2)) \le |\Sigma| \cdot 4^{d' \cdot m \log \frac{n}{m}} (m+1)(n+1)\,.$$

With $|\Sigma| \le \min\{m, n\} = m$ and $m + 1 \le 4^{\log m}$ for $m \ge 2$, this can be estimated by

$$\begin{aligned} \operatorname{alph}(L(A_1 \times A_2)) &\le (n+1) \cdot 4^{d' \cdot m \log \frac{n}{m} + 2 \log m} \\ &\le (n+1) \cdot 4^{3d' \cdot m \log \frac{n}{m}}, \end{aligned}$$

which holds for $n \ge m \ge 2$. Taking also the case $m = 1$ into account, it can be readily seen that we can find a suitable constant d such that the right-hand-side in turn is bounded above by $n \cdot d^{m \cdot \left(1 + \log \frac{n}{m}\right)}$, as desired. □

It turns out that a slight alteration of this proof gives a corresponding upper bound for the interleaving of two regular languages:

Theorem 21. *There exists a constant d such that for all $m \le n$, and for arbitrary alphabets,* $\operatorname{alph}(ш, m, n) \le n \cdot d^{m \cdot \left(1 + \log \frac{n}{m}\right)}$ *holds.*

Proof. As for the intersection operation, assume we are given two regular expressions r_1 and r_2 of size m and n, respectively. We apply a simple trick found in the textbook [33] to simulate, in a sense detailed below, the interleaving operation by means of the intersection operation.

We will consider first the case where the expressions r_1 and r_2 are over disjoint alphabets Σ_1 and Σ_2. Again, we convert first the two expressions into NFAs A_1 and A_2 having $m + 1$ and $n + 1$ states, respectively, such that both underlying graphs G_1 and G_2 have treewidth at most 2. Next, we add transitions ensuring that A_1 can consume symbols from Σ_2 without changing its state, and perform a similar construction for A_2. More precisely, for each state $p \in A_1$ and each alphabet symbol $a \in \Sigma_2$, we add transitions $p \xrightarrow{a} p$, and we perform a symmetric construction for A_2, in that we add for each state $q \in A_2$ and each alphabet symbol b in Σ_1 a transition $q \xrightarrow{b} q$. Let the A'_1 and A'_2 denote the respective NFAs resulting from this construction. It is not difficult to prove that $L(A_1') \cap L(A_2') = L(A_1) \sqcup\!\sqcup L(A_2)$, compare [33].

Notice that going, for $i = 1, 2$, from A_i to A_i' is mirrored in the underlying graphs in the mere addition of a self-loop to each vertex. Exactly this technicality was dealt with already in the proof of Theorem 20 for the intersection of NFAs, and thus the analysis carried out in that proof applies *mutatis mutandis* to the alphabetic width of the interleaving of two regular expressions that are over disjoint alphabets.

Finally, the case $\Sigma_1 \cap \Sigma_2 \neq \emptyset$ can be easily reduced to the case of disjoint alphabets: we rename the second alphabet $\Sigma_2 = \{a_1, a_2, \ldots, a_\ell\}$ into $\Sigma_2' = \{a_1', a_2', \ldots, a_\ell'\}$ and apply the construction outlined above. Then, in the resulting expression for the interleaved language, we change each occurrence of an alphabet symbol from Σ_2' back into an occurrence of the corresponding symbol from Σ_2. The final result describes the language $L(r_1) \sqcup\!\sqcup L(r_2)$, as desired. $\square$

Observe that for the parameter range where $m = \Theta(n)$, each of the two upper bounds asymptotically matches the corresponding lower bound stated in Theorem 17, that is, $\mathrm{alph}(\cap, n, \Theta(n)) = 2^{\Theta(n)}$ and $\mathrm{alph}(\sqcup\!\sqcup, n, \Theta(n)) = 2^{\Theta(n)}$. Also, the upper and lower bound are very close in general.

References

[1] Noga Alon, Jeff Kahn, and Paul D. Seymour. Large induced degenerate subgraphs. *Graphs and Combinatorics*, 3(1):203–211, 1987.

[2] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[3] Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of graphs. *SIAM Journal on Discrete Mathematics*, 11(1):168–181, 1998.

[4] Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[5] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[6] Janusz A. Brzozowski and Edward J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, EC-12(2):67–76, 1963.

[7] John H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[8] Manuel Delgado and José Morais. Approximation to the smallest regular expression for a given regular language. In Michael Domaratzki, Alexander Okhotin, Kai Salomaa, and Sheng Yu, editors, *9th Conference on Implementation and Application of Automata*, volume 3317 of *Lecture Notes in Computer Science*, pages 312–314. Springer, 2004.

[9] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 3rd edition, 2006.

[10] Keith Edwards and Graham E. Farr. Planarization and fragmentability of some classes of graphs. *Discrete Mathematics*, 308(12):2396–2406, 2008.

[11] Lawrence C. Eggan. Transition graphs and the star height of regular events. *Michigan Mathematical Journal*, 10:385–397, 1963.

[12] Andrzej Ehrenfeucht and H. Paul Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

[13] 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.

[14] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

[15] Wouter Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theoretical Computer Science*, 411(31-33):2987–2998, 2010.

[16] Wouter Gelade and Frank Neven. Succinctness of the complement and intersection of regular expressions. *ACM Transactions on Computational Logic*, 13(1):4:1–4:19, 2012.

[17] Hermann Gruber and Markus Holzer. Finite automata, digraph connectivity, and regular expression size. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walkuwiewicz, editors, 35*th International Colloquium on Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2008.

[18] Stefan Gulan and Henning Fernau. Local elimination-strategies in automata for shorter regular expressions. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, 34*th Conference on Current Trends in Theory and Practice of Computer Science, Volume II - Student Research Forum*, pages 46–57. Safarik University, Košice, Slovakia, 2008.

[19] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1–2):171–186, 1976.

[20] Yo-Sub Han and Derek Wood. Obtaining shorter regular expressions from finite-state automata. *Theoretical Computer Science*, 370(1-3):110–120, 2007.

[21] Markus Holzer and Martin Kutrib. Descriptional complexity — An introductory survey. In Carlos Martín-Vide, editor, *Scientific Applications of Language Methods*, volume 2 of *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, pages 1–58. Imperial College Press, 2010.

[22] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, 1979.

[23] Lucien Ilie and Sheng Yu. Follow automata. *Information and Computation*, 186(1):140–162, 2003.

[24] Meir Katchalski, William McCuaig, and Suzanne M. Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995.

[25] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, Annals of Mathematics Studies, pages 3–42. Princeton University Press, 1956.

[26] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[27] Robert McNaughton. The loop complexity of regular events. *Information Sciences*, 1:305–328, 1969.

[28] Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, EC-9(1):39–47, 1960.

[29] Nelma Moreira and Rogério Reis. Series-parallel automata and short regular expressions. *Fundamenta Informaticae*, 91(3-4):611–629, 2009.

[30] Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

[31] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[32] Jacques Sakarovitch. *Elements of Automata Theory*, chapter 4.3: Expressions for the behaviour of finite automata. Cambridge University Press, 2009.

[33] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory.* Cambridge University Press, 2009.

[34] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary Report. In 5*th Annual ACM Symposium on Theory of Computing*, pages 1–9. ACM, 1973.

[35] Pál Turán. On an extremal problem in graph theory (in Hungarian). *Matematikai és Fizikai Lapok*, 48:436–452, 1941.

[36] Derick Wood. *Theory of Computation.* John Wiley & Sons, 1987.