# T U M

I N S T I T U T F Ü R I N F O R M A T I K

# Provably Shorter Regular Expressions from Deterministic Finite Automata

Hermann Gruber Markus Holzer

TUM-I0805
Februar 08

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-02-I0805-0/1.-FI


Druck: Institut für Informatik der
Technischen Universität München

# Provably Shorter Regular Expressions from Deterministic Finite Automata

*Hermann Gruber*
Institut für Informatik, Ludwig-Maximilians-Universität München,
Oettingstraße 67, 80538 München, Germany
gruberh@tcs.ifi.lmu.de

*Markus Holzer*
Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching, Germany
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 alpbabetic 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 [8].

## 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* [18]. While algorithms accomplishing these tasks have been known for a long time, see, e.g., [21], there has been a renewed interest in these classical problems during the last few years. For instance, new algorithms for converting regular expressions into finite automata outperforming classical algorithms have been found only recently, as well as a matching lower bound of $\Omega(n \cdot \log^2 n)$ on the minimum number of transitions required by any equivalent nondeterministic finite automaton. That lower bound is, however, only reachable for growing alphabets, and a better algorithm is known for constant alphabet size, see [26] for the current state of the art.

Somewhat less is known about the converse direction, namely of converting finite automata into regular expressions. Apart from the fundamental nature of the problem, some applications of converting finite automata into regular expressions lie in control flow normalization, including uses in software engineering such as automatic translation of legacy code [23]. All known algorithms covering the general case are based on the classical ones, which are compared in the survey [25]. The drawback is that all of these (structurally similar) algorithms return expressions of exponential size in the worst case, and Ehrenfeucht and Zeiger exhibit a family of languages over a growing alphabet of size $n^2$ for which such an exponential blow-up is inevitable [7]. For alphabets of constant size, the existence of superpolynomial lower bounds remained open until very recently [9, 10, 12]. The best of these lower bounds imply that, in the worst case, size $2^{\Theta(n)}$ may be needed for a minimum regular expression equivalent to a binary $n$-state deterministic finite automaton.

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 algorithm: The authors of the seminal work [21] 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 [20]:

> "... 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., [4, 8, 13, 15, 22]. 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 [8] a recursive algorithm for converting planar $n$-state finite automata into regular expressions with a nontrivial 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 [19]. 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 [2], but we will show how the idea of using separators can be generalized and implemented simply as a divide-and-conquer elimination strategy. The difficulty, when applying this idea is that on the one hand large 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. Both 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. Thus, both presented heuristics significantly outperform the classical McNaughton-Yamada algorithm. As a side result, we identify a structural restriction on the transition structure of the given finite automaton that guarantees a polynomial upper bound on the resulting regular expression. This is the class of digraphs of

bounded undirected treewidth, thus identifying a large class of finite automata with a feasible conversion problem. The new insights on the conversion problem can be successfully 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]. This result thus settles a question stated in [8], by complementing previous lower bounds from [9, 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 [5].

## 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 [16]. In particular, let $\Sigma$ be a finite alphabet and $\Sigma^*$ the set of all words over the alphabet $\Sigma$, including the empty word $\varepsilon$. The length of a word $w$ is denoted by $|w|$, where $|\varepsilon| = 0$. A *(formal) language* over the alphabet $\Sigma$ is a subset of $\Sigma^*$. Apart from the usual set operations, such as union, intersection, and complement (with respect to $\Sigma^*$), the following operations on languages are commonly considered: The *concatenation* of two languages $L_1$ and $L_2$, denoted by $L_1 \cdot L_2$, is defined as the set $\{ vw \mid v \in L_1 \text{ and } w \in L_2 \}$, and for $i \geq 1$, the $i$th power of a language $L$, denoted by $L^i$, is defined as the $i$-fold concatenation of $L$ with itself. By convention, $L^0 = \{\varepsilon\}$. The *(Kleene) star* of a language $L$ is given by $L^* = \bigcup_{i \geq 0} L^i$.

A *nondeterministic finite automaton* (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ 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 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 $\delta$ is extended as usual 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 *an incomplete deterministic* or simply a *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 (deterministic or nondeterministic) finite automata are *equivalent*, if they accept the same language. Without loss of generality we assume throughout this paper, that every finite automata has useful states only, i.e., every state is accessible from the initial state and co-accessible form 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 $\Sigma$ be an alphabet. The regular expressions over $\Sigma$ are defined recursively in the usual way:[2] $\emptyset$, $\varepsilon$, and

[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 $1\,\text{MByte} = 2^{10}\,\text{KByte}$ disk space, while the smaller one needs only 1 KByte, we still have $\log \frac{m}{n} = 10$

[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.

every letter $a$ with $a \in \Sigma$ is a regular expression; 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 $\Sigma$, denoted by $\mathrm{alph}(r)$, is defined as the total number of occurrences of letters of $\Sigma$ 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. Since there is evidence that equivalence of regular expressions is difficult to determine algorithmically [27], 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 [1], and the remaining three 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 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 of edge $E \subseteq V \times V$. If the edge relation $E$ is symmetric, the graph is said to be *symmetric*. Intuitively, a symmetric graph is obtained by forgetting the orientation of the original edges in $G$. We say that an edge $e = (u, v)$ *enter* (*leaves*, respectively) the vertex $v$ ($u$, respectively). For a vertex $v$, the *indegree* of $v$ is the number of edges entering $v$, and is denoted by $\mathrm{indeg}(v)$. Similar, its *outdegree* is number of edges leaving $v$, and is denoted by $\mathrm{outdeg}(v)$.

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 $E$. A subgraph $H$ is called *induced* if furthermore for each edge $(u, v) \in E$ with $u, v \in U$, the pair $(u, v)$ is also an edge in $E$. In the latter case, graph $H$ is referred to as the subgraph of $G$ induced by $U$, and denoted by $G[U]$.

Finally, a *hammock* is a digraph $G = (V, E)$ having two distinguished vertices $s$ and $t$ satisfying the properties (1) $\mathrm{indeg}(s) = 0$ and $\mathrm{outdeg}(t) = 0$, 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 $H = (Q, E, s, t)$.

# 3 Conversion Algorithms—The State Elimination Technique

There are several algorithms to convert a finite automaton into a regular expression. One of the most well known ones is the McNaughton-Yamada algorithm and the state elimination technique. Although these algorithms are somehow different, they produce roughly the same results, which may considerable vary with the ordering of states that is used. Before we briefly recall the state elimination technique, which is the basic algorithm for our considerations, we

want to make the following simple observation.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. With $A$ 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

$$E = \{\, (p,q) \in Q^2 \mid q \in \delta(p,a), \text{ for some } a \in \Sigma \,\} \cup \{(s, q_0\}\} \cup \{\, (q,t) \mid q \in F \,\}.$$

Obviously, the assumption that $A$ has only useful states is no restriction at all. 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 nothing other than a (possibly empty) sequence of edges. To this end define the substitution $\sigma : E \to \Sigma^*$ by $(p,q) \mapsto \{\, a \in \Sigma \mid q \in \delta(p,a) \,\}$, if $p, q \in Q$, $(s, q_0) \mapsto \{\varepsilon\}$, and $(q,t) \mapsto \{\varepsilon\}$, for $q \in F$, which naturally extends to words and languages over $E$. Then a straightforward induction on the walk length now shows that walks in $H(A)$ are mapped under $\sigma$ to computation paths in the finite automaton, and that applying $\sigma$ to the language yields the language $L(A)$ accepted by the finite automaton, i.e., $L(A) = \sigma(L_{st}^Q)$. Here $L_{xy}^Z$, 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 $y$. For instance, for $(x,y) \in E$, the walk $(x,y)$ is in $L_{xy}^{\emptyset}$, since it does not have internal vertices at all, and the walk $(x,y)(y,z)$ with $(x,y) \in E$ and $(y,z) \in E$ belongs to $L_{xz}^{\{y\}}$. This definition naturally extends to $L_{XY}^Z$ 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 folklore lemma (see also [16]). Because the proof is very short, we reproduce it here.

**Lemma 1.** *Let $\Gamma$ and $\Sigma$ be finite alphabets, and let $r$ be a regular expression over $\Gamma$. 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 $a \in \Sigma$.*

*Proof.* The proof is an easy induction on the number of operator occurrences in $r$. In the base case, $r = \emptyset$, $r = \varepsilon$, or $r = a$, for some $a \in \Sigma$. In the first two case, the substitution does not apply, and in the other case, $r_a$ describes the language $\rho(L(r))$. For the induction step, it is readily verified that substitution commutes with (both finite and infinite) union and with concatenation, i.e., $\rho(L_1 \cdot L_2) = \rho(L_1) \cdot \rho(L_2)$, and $\rho(\bigcup_i L_i) = \bigcup_i \rho(L_i)$. This also implies that substitution commutes with the star operator, and thus we can always "push" the substitution inside the respective outermost operation occurring in $r$, be it concatenation, union, or star. □

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.

The state elimination technique is an optimized version of the McNaughton-Yamada algorithm, avoiding the unnecessary computation of subexpressions. Both algorithms compute regular expressions $r_{jk}^S$ satisfying $L(r_{jk}^S) = L_{jk}^S$, for every $j, k \in Q \cup \{s,t\}$ and $S \subseteq Q$, for the hammock $H(A) = (Q, E, s, t)$ that is associated to a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$. Recall, that $L_{jk}^S$ refers to the set of all paths form vertex $j$ to $k$ whose internal vertices are all

in $S$. Then we are interested in the expressions $r^Q_{st}$ since $L(A) = \sigma(L(r^Q_{st}))$. To this end we have to fix an ordering on the states and hence on the vertices of $H(A)$. To this end it is convenient to write any (total) order on a finite set $S$ as a word, where the relative position of the letters naturally specifies the order. The (unique) ordering of the emptyset is denoted by $\varepsilon$. Thus, the superscript $S$ in $r^S_{jk}$ refers to the total order induced by the set $S$ and is assumed to be a word—hence $L(r^\varepsilon_{jk}) = L^\emptyset_{jk}$. So we fix an order on $Q$ and let $S$ be a prefix of $Q$, i.e., the order induced by $S$ is compatible with the order on $Q$. Moreover assume that $i$ is the next vertex in the order after the vertices in $S$—in the terminology of words $S \cdot i$ is a prefix of $Q$. Since any path from vertex $j$ to $k$ whose internal vertices are in $S \cup \{i\}$ can be written as

$$L^{S\cup\{i\}}_{jk} = L^S_{jk} + L^S_{ji} \cdot (L^S_{ii})^* \cdot L^S_{ik},$$

we are led to define the identity

$$r^{S\cdot i}_{jk} = r^S_{jk} + r^S_{ji} \cdot (r^S_{ii})^* \cdot r^S_{ik} \tag{1}$$

on regular expressions, for every $j, k \in Q \cup \{s, t\}$ and $Si$ prefix of the ordered set $Q$. To complete the description of this algorithm the terminating cases are defined to be (1) $r^\varepsilon_{jj} = (j, j) + \varepsilon$, if $(j, j) \in E$, and $r^\varepsilon_{jj} = \varepsilon$, otherwise, and moreover, (2) for $j \neq k$ we define $r^\varepsilon_{jk} = (j, k)$, if $(j, k) \in E$, and $r^\varepsilon_{jk} = \emptyset$, otherwise. Hence for an ordered subset $S$ of $Q$, let $R^S = (r^S_{jk})_{j,k\in Q\cup\{s,t\}}$ denote the *regular expression matrix* obtained after eliminating $S$.

The state elimination technique computes the regular expressions $r^S_{jk}$, for any prefix $S$ of $Q$, in sequence, which is induced by the order on $Q$. Since we are only interested in the expression $r^Q_{st}$, we only need to generate intermediate expressions which are eventually needed for the computation of this expression. It is easy to see that we do not need to compute any expressions $r^S_{jk}$ with $j \in S$ or $k \in S$. The algorithm thus obtained is summarized by the rules pictorially presented in Figure 1, where trivial rules with $r^{S\cdot i}_{jk} = r^S_{jk}$ are omitted. Note that the inner vertex $i$ on the right hand side of Figure 1 is not necessarily an isolated vertex, since it may have other ingoing and outgoing edges depicted by wiggled lines. But it becomes an isolated vertex after applying the rules for all pairs $(j, k)$ with $j, k \neq i$. Afterwards, the vertex can be safely eliminated, i.e., removed, as all necessary information is represented on other edge labels not entering or leaving $i$. This explains the term *state elimination algorithm*, because during the computation of the expressions $r^S_{jk}$ 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 $j$ to $k$ in $H(A)$ with internal vertices from $S \,\}$. The graphical interpretation of the rules makes the algorithm suitable for manual conversion as well, at least for reasonably small and well-structured instances of finite automata. A further slight enhancement on the algorithm concerns the usage of the similarity relations introduced earlier. With a straightforward implementation one can ensure that $r^S_{jk} = \emptyset$ if and only if $L^S_{jk} = \emptyset$.

The size of the regular expression resulting from applying the McNaughton-Yamada algorithm has been analyzed in [8]. There it was shown that the algorithm produces a regular expression of alphabetic width at most $|\Sigma| \cdot n \cdot 4^n$. Because of the optimization to generate intermediate expressions only if they are eventually needed in the final expression, state elimination is better by a factor of $n$.

**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 alpbabetic width at most $|\Sigma| \cdot 4^n$.*

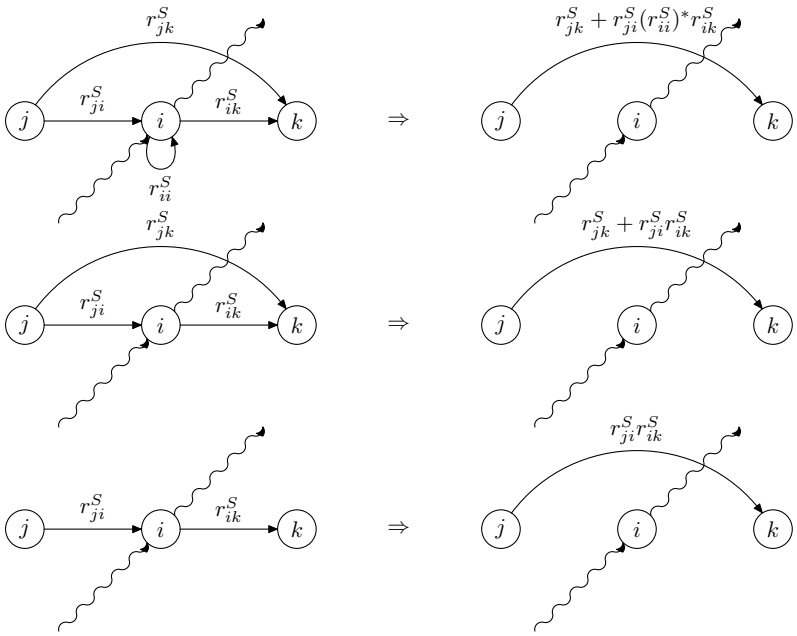


Figure 1: Schematic drawing of the improved substitution rules.

*Proof.* Let $H(A) = (Q, E, s, t)$ be the hammock associated with the finite automaton $A = (Q, \Sigma, \delta, q_0, F)$. We apply the procedure described above. Initially, each expression $r^{\varepsilon}_{jk}$, for $j, k \in Q \cup \{s, t\}$, has alphabetic width at most one and by Equation (1) each elimination step, going from expression $r^S_{jk}$ to $r^{Si}_{jk}$, for $j, k \in Q \cup \{s, t\}$, $S \subseteq Q$, and state $i$, increases the alphabetic width of the intermediate expressions each by a factor of at most 4. By induction, we obtain that the alphabetic width of $r^Q_{st}$ is at most $4^{|Q|}$, and $L(A) = \sigma(L(r^Q_{st}))$, for the appropriate substitution $\sigma : E \to \Sigma^*$ defined previously. Applying $\sigma$ increases the expression size by factor of at most $|\Sigma|$. This proves the stated claim. □

The algorithm just described requires to specify an *elimination ordering* of the state set $Q$, which prescribes in which order the states are eliminated. As observed already in the seminal work [21], the choice of the elimination ordering can greatly influence the size of the resulting regular expression. In the following sections, we will discuss some strategies for choosing a good elimination ordering.

# 4 Choosing a Good Elimination Ordering

This section is organized as follows: First, we give a brief account on two groups of algorithms for choosing good elimination orderings, which we call iterative and divide-and-conquer strategies. Then we present a lemma that proves useful for designing algorithms in both groups. Finally we present three algorithms for choosing good elimination orderings yielding nontrivial performance guarantees, one of which gives polynomial-size regular expressions for a restricted yet large class of finite automata.

## 4.1 Review of Previous Approaches

We briefly outline the main ideas underlying previous accounts on choosing elimination orderings. For details and correctness proofs, see the respective references. The solutions 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, the others are based on the divide-and conquer paradigm, suggesting a recursive implementation.

### 4.1.1 Iterative Strategies

Already the early work [21] 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. A more sophisticated approach, presented in [4], exploits the fact that the vertex degrees in the underlying graph change during state elimination. A new ordering of the remaining states is recomputed after each elimination step. Furthermore, a more elaborated weight function than just the vertex degree is used. This weight function takes into account the indegree, the outdegree and the current length of each intermediate expression. In a particular case, namely acyclic finite automata having a directed series-parallel transition structure, we can always find a vertex of degree 1 and eliminating this vertex preserves the structure of the automaton. Repeating this process for the residual graph gives a regular expression whose size is linear in the number of edges in the underlying digraph, see [22]. In [13], this idea is integrated into an iterative heuristic for the general case. The latter algorithm iteratively searches for series-parallel substructures, and for cycles that are—in some precise sense—easy to break up, in order to reach an acyclic transition structure as quickly as possible. Transforming acyclic finite automata into regular expressions is quasipolynomial in the worst case [7, 12], using an algorithm that is not based on state elimination, see [7, 8, 12] for more information.

### 4.1.2 Divide-and-Conquer Strategies

Divide-and-conquer strategies typically identify states whose elimination would affect the structure of the residual graph in an unfavorable way, and places these states to the very end of the elimination ordering. For instance, a state elimination algorithm was proposed recently [15] that recursively decomposes, if possible, a given finite automaton $A$ into subautomata $A_1$ and $A_2$, such that $L(A) = L(A_1)L(A_2)$ or $L(A) = L(A_1) \cup L(A_2)$. The respective start and accepting state of the subautomata are chosen to be eliminated last, and the procedure continues recursively on these subautomata. Interestingly, if there exists a bridge state that separates two subautomata that are composed in series, one can actually *prove* that every optimal elimination ordering ends with that state. As observed in [15], sometimes the iterative approach from [4] explained previously behaves too greedily and chooses to eliminate a bridge state first, thus resulting in a suboptimal elimination ordering. Admittedly, not many finite automata do not admit such a nice directed series-parallel decomposition needed for this algorithm.

The larger class of planar finite automata is encompassed in [8], i.e., those having a planar transition structure. There a recursive algorithm for converting planar $n$-state finite automata into regular expressions with a nontrivial performance guarantee of $2^{O(\sqrt{n})}$ was presented. As proved in [10], this bound is asymptotically optimal for the planar case. The mentioned algo-

rithm exploits the separator theorem for planar graphs [19]. That procedure is seemingly more difficult to implement than a mere state elimination strategy. But we will show in this paper how the idea of using separators can be generalized, and, with the aid of Lemma 3, implemented simply as a divide-and-conquer elimination strategy.

## 4.2 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 disjoint connected components.

**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 disjoint connected 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 $|T_1| + |T_2|$ while distinguishing three cases. The induction is rooted at $|T_1| + |T_2| = 0$. For the case $T_2$ is empty, we have in general $r_{jk}^{T_1 T_2} = r_{jk}^{T_1} \cong r_{jk}^{T_1} + r_{jk}^{\varepsilon}$, 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

$$r_{jk}^{T_1 T_2} \cong r_{jk}^{T_1 T} + r_{jt}^{T_1 T} \cdot (r_{tt}^{T_1 T})^* \cdot r_{tk}^{T_1 T}. \tag{2}$$

For the first of the four subexpressions on the right-hand side, we have $|T| = n - 1$, and the induction hypothesis applies:

$$r_{jk}^{T_1 T} \cong r_{jk}^{T_1} + r_{jk}^{T}.$$

For the last three subexpressions, we have $r_{jt}^{T_1 T} \cong r_{jt}^{T}$, 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 $r_{jt}^{T_1} \cong r_{jt}^{\varepsilon}$, 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 $r_{jt}^{\varepsilon}$; 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 the four subexpression congruences we just found into Congruence (2), we get

$$r_{jk}^{T_1 T_2} \cong r_{jk}^{T_1} + r_{jk}^{T} + r_{jt}^{T} (r_{tt}^{T})^* r_{tk}^{T} = r_{jk}^{T_1} + r_{jk}^{T_2},$$

which is the desired result. □

## 4.3 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*

$$r_{jk}^I \cong r_{jk}^\varepsilon + \sum_{i \in I} r_{ji}^\varepsilon \cdot (r_{ii}^\varepsilon)^* \cdot r_{ik}^\varepsilon.$$

*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 disjoint connected components $H[T]$ and $H[\{t\}]$. Thus, Lemma 3 is applicable, and

$$r_{jk}^I \cong r_{jk}^T + r_{jk}^t = r_{jk}^T + r_{jk}^\varepsilon + r_{jt}^\varepsilon \cdot (r_{tt}^\varepsilon)^* \cdot r_{tk}^\varepsilon.$$

By induction hypothesis, $r_{jk}^T \cong r_{jk}^\varepsilon + \sum_{i \in T} r_{ji}^\varepsilon \cdot (r_{ii}^\varepsilon)^* \cdot r_{ik}^\varepsilon$. Since the notion of similarity allows to suppress the double appearance of $r_{jk}^\varepsilon$ in a sum of subexpressions, the expression $r_{jk}^T + r_{jk}^t$ is similar to the right hand side of the congruence stated in 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 $r_{jk}^{SI}$ obtained after elimination of $SI$ holds*

$$r_{jk}^{SI} \cong r_{jk}^S + \sum_{i \in I} r_{ji}^S \cdot \left(r_{ii}^S\right)^* \cdot r_{ik}^S. \tag{3}$$

*Proof.* We break up the generation of the expression $r_{jk}^{SI}$ into two phases, by first eliminating $S$ followed by eliminating $I$. We take a look at the second phase first. For $j \neq k$, define $n_{jk}^\varepsilon$ as the expression $(j, k)$ if there is an edge from $j$ to $k$ in $H^S$ and $\emptyset$ otherwise. For $j = k$, the definition is almost the same, but we add in either case $\varepsilon$ as a summand to the expression. By eliminating $I$ in $H^S$, we get a regular expression matrix whose entries $n_{jk}^I$ each denote the set of walks from $j$ to $k$ in $H^S$. By Lemma 4, we have $n_{jk}^I \cong n_{jk}^\varepsilon + \sum_{i \in I} n_{ji}^\varepsilon \cdot (n_{ii}^\varepsilon)^* \cdot n_{ik}^\varepsilon$. From the iterated substitution scheme of the state elimination algorithm, it is readily seen that the regular expression $r_{jk}^{SI}$ resulting from eliminating $SI$ in the original hammock $H$ can be obtained by replacing, for each $\ell$ and $m$ satisfying $\ell, m \in (Q \setminus S) \cup \{s, t\}$, each occurrence of $n_{\ell m}^\varepsilon$ with an occurrence of $r_{\ell m}^S$. □

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 $H^{I_1 I_2 \ldots I_k}$. The cardinality 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 [28]. 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, see, e.g., [14].

**Theorem 7.** *Let $A$ be an $n$-state* deterministic *finite automaton with input alphabet $\Sigma$. 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 $c = \frac{2|\Sigma| \cdot 2|\Sigma|^2 \cdot 2|\Sigma|^4}{(2|\Sigma|+1)(2|\Sigma|^2+1)(2|\Sigma|^4+1)}$. The ordering is obtained by the strategy of iteratively eliminating independent sets.*

*Proof.* Let $H = H(A) = (Q, E, s, t)$ be the hammock associated with the automaton $A$. Let $\ell = |\Sigma|$. The number of edges in the induced subgraph $H[Q]$ is at most $\ell n$, since every state in $A$ has at most $\ell$ outgoing transitions. Thus the average degree in the symmetric closure of $H[Q]$ is at most $2\ell$. By Turán's Theorem, we can choose an independent set $I_1$ of size $n/(2\ell+1)$ in the first round. Let $H_k = H^{I_1 I_2 \ldots I_k}$ denote the hammock obtained after the $k$th elimination round. In order to apply Turán's Theorem for the induced subgraph $H_k - \{s, t\}$, we estimate the maximum outdegree $d_k$ of the latter graph. For $k = 0$, we have already seen that $d_k \leq 2\ell$, for $H_0 = H(A)$. For $k \geq 1$, the hammock $H_k$ is obtained from $H_{k-1}$ by replacing each edge whose tail is in $I_k$ by at most $d_{k-1}$ new edges. Thus each vertex $v$ in $H_k - \{s, t\}$ that formerly had outdegree $d$ in $H_{k-1}$ now has, after eliminating $I_k$, outdegree at most $d_k \leq d \cdot d_{k-1} \leq d_{k-1}^2$. Thus, by induction, $d_k \leq \ell^{2^k}$. Turán's Theorem then asserts that we choose a set $I_k$ independent in $H_k - \{s, t\}$ that contains a fraction of $1/(2\ell^{2^k}+1)$ of all vertices in that digraph. With $Q_k$ denoting the set of internal vertices in $H_k$, we obtain in this way $|Q_{k+1}| \leq \left(1 - \frac{1}{2\ell^{2^k}+1}\right)|Q_k|$. The number of remaining internal vertices after three rounds is then

$$|Q_3| \leq \frac{2\ell \cdot 2\ell^2 \cdot 2\ell^4}{(2\ell+1)(2\ell^2+1)(2\ell^4+1)} \cdot |Q_0| = c \cdot n.$$

Eliminating the states in $I_k$ increases the size of each intermediate expression by a factor of at most $1+3|I_k| = O(n)$ by Lemma 4. Thus, the first three rounds can only incur a polynomial size blow-up of at most $O(n^3)$, while already eliminating a constant fraction of the total number of states.

For simplicity, beginning with round four, we continue by eliminating independent sets of size 1 in each round, which is equivalent to applying the state elimination algorithm to the Hammock $H_4$, which has $c \cdot n$ internal vertices. The expressions $r_{jk}^{I_1 I_2 I_3}$ resulting from eliminating $I_1 I_2 I_3$ each have size $O(n^3)$. By eliminating the $c \cdot n$ remaining internal vertices the size of each the intermediate expressions increases by a factor of at most 4 for each vertex. Hence, we end up with a regular expression of alphabetic width $n^{O(1)} \cdot 4^{c \cdot n}$. Applying the substitution $\sigma$ increases the expression size by a factor of at most $|\Sigma|$. Thus we have a regular expression of alphabetic width $|\Sigma| \cdot n^{O(1)} \cdot 4^{c \cdot n}$ for $L(A)$. □

For the particular 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, $O(2.602^n)$ for the size of a regular expression obtained from a deterministic $n$-state automaton. This appears reasonable at once in presence of a worst-case lower bound of $\gamma^n$, even for the case of binary alphabets, which was proved recently in [10]. Here, $\gamma > 1$ is a fixed constant[3] that is independent of $n$. In the sequel, we will develop a different algorithm that even achieves $O(1.742^n)$ for binary alphabets. In order to do this, we first devise an algorithm for converting finite automata of small treewidth into much shorter regular expressions.

## 4.4 Converting Finite 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 graphs of treewidth $t$, every nontrivial subset of the vertex set admits a small balanced $k$-way separator of size at most $t+1$, for some $k$ [24]. An elementary observation on sets of real numbers shows that we can always find such a separator with $k = 3$ by grouping the $k$ disjoint subgraphs together in a suitable manner:

**Lemma 9.** *Let $A = \{a_1, a_2, \ldots, a_k\}$ be a finite set of real numbers satisfying $a_i \leq \frac{1}{2}\sum_{a \in A} a$, for $1 \leq i \leq k$. Then $A$ can be partitioned into three subsets $A_1$, $A_2$, and $A_3$ such that $\sum_{a \in A_j} a \leq \frac{1}{2}\sum_{a \in A} a$, for $1 \leq j \leq 3$.*

*Proof.* By induction on $k$. The base case $k \leq 3$ is immediate. For $k > 3$, assume $a_1$ and $a_2$ are the smallest elements in $A$. If $a_1 + a_2 \geq \frac{1}{2}\sum_{a \in A} a$, let $A_1 = \{a_1\}$, $A_2 = \{a_2\}$, and $A_3 = \{a_3, a_4, \ldots, a_k\}$. This partition obviously satisfies the requirements. Otherwise, the set $A' = \{a_1 + a_2, a_3, a_4, \ldots a_k\}$, which is of cardinality $k - 1$, admits such a partition by induction hypothesis, and so does $A$. □

Together with the mentioned result from [24], we thus have:

**Lemma 10.** *Let $G = (V, E)$ be a digraph of undirected treewidth at most $t$. Then for every subset $S$ of $V$, there exists a balanced 3-way separator of size at most $t + 1$.* □

[3] By tracking the size of the constants used in the chain reductions used in that proof, one can deduce a concrete value for the constant $\gamma$. 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 [3], for the vertex expansion of $\ell$-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 DFA to $m = 10n$ whilst preserving 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 parameter $\gamma$ from below away from 1.

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

**Theorem 11.** *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an $n$-state* nondeterministic *finite automaton, $H$ its associated hammock, and let $t$ 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 $|\Sigma| \cdot n^{2t+2+\log 3}$. The ordering on the states is obtained by the strategy of recursively computing balanced 3-way separators.*

*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 10 for each set of states $S \subseteq Q$, we can find a balanced 3-way separator $X$, such that $|X| \leq t+1$, 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 the ordering, and so on.

Since for each $S \subseteq Q$ the alphabetic width of $r_{jk}^{S}$ is at most $4^{|X|} \sum_{i=1}^{3} \operatorname{alph}(r_{jk}^{T_i})$, for some $X$, $T_1$, $T_2$, and $T_3$ with $|X| \leq t+1$ and $|T_i| \leq \frac{1}{2}S$, for $1 \leq i \leq 3$. Moreover, the alphabetic width of $r_{jk}^{S}$ is bounded above by the recurrence

$$R(1) \leq 1 \quad \text{and} \quad R(n) \leq 4^{t+1} \cdot 3 \cdot R\left(\frac{n}{2}\right), \quad \text{for } n \geq 2.$$

We obtain $R(n) \leq 4^{(t+1)\log n} 3^{\log n}$. To conclude the proof, we use simple algebra to derive

$$R(|Q|) \leq n^{2t+2+\log 3}.$$

Applying the substitution $\sigma$ increases the expression size by a factor of at most $|\Sigma|$. Thus we have a regular expression of alphabetic width $|\Sigma| \cdot n^{2t+2+\log 3}$ for the language $L(A)$. □

## 4.5 Improved Kernelization

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. It has been shown recently that every undirected graph with bounded average degree has a large induced subgraph of treewidth at most two [6, Theorem 10]:

**Theorem 12.** *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 $\frac{3n}{\overline{d}+1}$.* □

**Remark 1.** *We note that the statement in [6, Theorem 10] only asserts that the graph found by the algorithm is planar, but it is explicitly mentioned there that this graph is in fact of treewidth at most two.*[4] *We also note that the lower bound given in the above mentioned work is in fact more complicated, giving a slightly better bound on non-integer values of $\overline{d}$.*

[4] In [6], they use the term series-parallel for graphs of treewidth at most two. In the literature there are several non-equivalent definition of this term; we reserve the use of this term to acyclic digraphs obtained by iterative use of the series and parallel composition operations starting from mutually disjoint edges.

This gives rise to an algorithm with improved performance guarantee for converting finite automata with relatively few transitions into regular expressions:

**Theorem 13.** *Let $A$ be an $n$-state* deterministic *finite automaton with input alphabet $\Sigma$. 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 $c = \frac{2|\Sigma|-2}{2|\Sigma|+1}$. The ordering on the states is obtained by the following strategy: First identify a large induced subgraph of undirected treewidth at most two and then recursively apply the balanced 3-way separator strategy on that induced subgraph.*

*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 $\ell$, where $\ell = |\Sigma|$, so the average degree of its undirected version is at most $2\ell$.

By Theorem 12, we can find a subset $S$ of $Q$ having size $\frac{3n}{2\ell+1} = (1-c) \cdot n$ 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 $(r^{\varepsilon}_{jk})_{j,k}$, whose rows $j$ and columns $k$ range over the set $(Q \setminus S) \cup \{s, t\}$. The algorithm from the proof of Theorem 11 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 $n^{O(1)} \cdot 4^{c \cdot n}$. Again, we finally apply the substitution $\sigma$ 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 special case of a binary alphabet, we obtain that the maximum blow-up arising in the conversion from deterministic finite automata to regular expressions is at most $|\Sigma| \cdot n^{O(1)} \cdot 4^{2/5 \cdot n}$, where $4^{2/5} \doteq 1.741$.

# 5 Regular Language Operations and Regular Expression Size

In this section, we apply state elimination strategies to questions related to the descriptional complexity of language operations on regular expressions. Both nondeterministic and deterministic state complexity of most basic operations on regular languages are well understood, and tight bounds have been obtained in many cases during the last two decades. The effect of these operations on regular expression size was apparently first discussed in [8]. The operations union, concatenation and star are syntactically supported by regular expressions and thus the alphabetic width of the result is linear in the size of the operand(s). Easy examples show this is optimal in the worst case. Although the reversal operation is not built in syntactically, a simple inductive argument shows that reversal does not change the alphabetic width in any case.

More interesting are other operations, e.g., intersection and complement, for which some upper and lower bounds can be obtained by combining known upper and lower bounds on

nondeterministic state complexity of these operations with the fact that every regular expression can be converted into a nondeterministic finite automaton of (almost) the same size, and the size blow-up for the conversion in the other direction is at most exponential. However, there remains an exponential gap between the upper bounds thus obtained and the lower bounds coming from nondeterministic state complexity. Narrowing these gaps is an open problem stated in [8]. Only very recently, some of the lower bounds have been significantly improved [9, 10], giving, for instance, a roughly doubly-exponential lower bound on the alphabetic width of the complement operation, even for binary alphabets. Here we take a closer look at the intersection operation, where we obtain an improved upper bound that matches the best known lower bound in the case both operands are of nearly the same size. We also consider a less common regularity preserving operation, namely taking the first half of a language, whose descriptional complexity was studied recently for the deterministic finite automaton model of description [5].

For studying these operations, we rely on separation properties of a certain graph product, namely the categorical product (also known as Kronecker product), defined as follows.

**Definition 14.** *For two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the* categorical product $G_1 \times G_2$ *is defined as the directed graph whose vertex set is $V = V_1 \times V_2$ and which has an edge $((u_1, u_2), (v_1, v_2))$ whenever both $(u_1, v_1)$ is an edge in $G_1$ and $(u_2, v_2)$ is an edge in $G_2$.*

The following lemma is immediate from the definition of balanced 3-way separators (Definition 8) and the definition of categorical product:

**Lemma 15.** *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 $(G_1 \times G_2)[S_1 \times S_2] - (X \times S_2)$ falls apart into the mutually disjoint subgraphs $G[T_i \times S_2]$, for $1 \leq i \leq 3$, with $0 \leq |T_i \times S_2| \leq \frac{1}{2}|(S_1 \setminus X) \times S_2|$.*

The following theorem gives an upper bound on the resulting alphabetic width of the intersection operation, which nicely contrasts with the lower bound of $2^{\Omega(\min\{m,n\})}$, given two regular expressions of alphabetic width $m$ and $n$, respectively, obtained recently in [10].

**Theorem 16.** *Let $L_1$ and $L_2$ be regular languages over a common alphabet $\Sigma$, with alphabetic width at most $m$ and $n$, respectively. Then*

$$\operatorname{alph}(L_1 \cap L_2) \leq |\Sigma| \cdot 2^{O\left(1+\log \frac{m}{n}\right) \min\{m,n\}}.$$

*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 directed graph of the underlying transition structure has undirected treewidth at most two, e.g., by using the recursive scheme given in [17]—this nondeterministic finite automaton will in general have $\varepsilon$-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 $H = (Q, E, s, t)$, with $|Q| \leq m - 1$.

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 $H$.

We proceed in a similar way as in the proof of Theorem 11, 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 Lemma 15. This is done as follows: If $|S_1| < |S_2|$, then exchange the roles of $G_1$ and $G_2$, and that 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 10 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 Lemma 15 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 $\operatorname{alph}(L_1 \cap L_2) \leq |\Sigma| \cdot \operatorname{alph}(r_{st}^{S_1 \times S_2})$, define the function

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

An easy observation is that for the degenerate case, where $S_1$ and $S_2$ have both at most one element, we have $A(1, 1) \leq 1$. An upper bound is obtained thus by solving the recurrence

$$A(\beta, \eta) = \begin{cases} A(\eta, \beta) & \text{for } 1 < \beta < \eta, \\ 1 & \text{for } \beta = \eta = 1, \\ 3 \cdot A\left(\left\lfloor \frac{\beta}{2} \right\rfloor, \eta\right) \cdot 4^{3\eta} & \text{otherwise} \end{cases} \tag{4}$$

In order to analyze this recurrence, assume without loss of generality that initially $\beta \geq \eta$. We observe first that when evaluating $A$, the parameters $\beta$ and $\eta$ are eventually alternatingly decreased in each recursive call: Namely, for $1 < \eta \leq \beta < 2\eta$, we obtain

$$A(\beta, \eta) \leq 3 \cdot 4^{3\eta} A\left(\frac{\beta}{2}, \eta\right) = 3 \cdot 4^{3\eta} A\left(\eta, \frac{\beta}{2}\right) \leq 3^2 \cdot 4^{3(\eta + 1/2\beta)} A\left(\frac{\eta}{2}, \frac{\beta}{2}\right).$$

Then the parameter pair $\left(\frac{\eta}{2}, \frac{\beta}{2}\right)$ is switched again, and we have halved both parameters in the recurrent expression. By iterating the above procedure for $\lfloor \log \beta \rfloor$ times, and generously overestimating the finite geometric sums generated during this process, we obtain

$$A(\beta, \eta) \leq 3^{2 \log \beta} 4^{6\eta + 3\beta} A(1,1) = 2^{O(\eta)}, \quad \text{for } \eta \leq \beta < 2\eta,$$

and the recurrence satisfies the desired upper bound in this case.

It remains to reduce the case $\beta \geq 2\eta$ to the balanced case we just discussed. Let $x = \lfloor \log \left(\frac{\beta}{\eta}\right) \rfloor$. When iterating the recurrence given in Equation (4) for $x$ times, the first parameter always remains larger than the second, and we get $A(\beta, \eta) \leq 3^x \cdot 4^{3x\eta} A(2^{-x}\beta, \eta) = 2^{O(\log(\frac{\beta}{\eta}))\eta} A(2^{-x}\beta, \eta)$. Let $y$ denote the fractional part of $\log \frac{\beta}{\eta}$. Then $2^{-x-y} = \frac{\eta}{\beta}$, or $2^{-x}\beta = 2^y \eta$, and using $0 \leq y < 1$, we get $\eta \leq 2^{-x}\beta < 2\eta$, and the case we have already studied applies, so we have $A(2^{-x}\beta, \eta) = 2^{O(\eta)}$. Putting these together, we obtain, for ,

$$A(\beta, \eta) = 2^{O(\log \frac{\beta}{\eta})\eta} \cdot 2^{O(\eta)} = 2^{O(1 + \log \frac{\beta}{\eta})\eta}, \quad \text{for } 2 < 2\eta \leq \beta,$$

and the proof is nearly completed. Again applying the substitution $\sigma$ increases the alphabetic width by at most a factor of $|\Sigma|$. This proves the desired result. □

Essentially the same technique as for intersection applies for the first-half operation on languages $L \subseteq \Sigma^*$, which is defined as

$$\frac{1}{2}L = \{\, x \in \Sigma^* \mid \text{there exists } y \in \Sigma^* \text{ with } |x| = |y| \text{ such that } xy \in L \,\}.$$

Observe that the trivial construction produces an upper bound of $|\Sigma| \cdot 2^{O\left(n^2\right)}$.

**Theorem 17.** *Let $L \subseteq \Sigma^*$ be a regular language of alphabetic width at most $n$. Then*

$$\operatorname{alph}\left(\frac{1}{2}L\right) = |\Sigma| \cdot 2^{O(n)}.$$

*Proof.* As before, we convert a regular expression of size $n$ describing $L$ into an equivalent nondeterministic finite automaton $A$ with at most $n+1$ states such that the digraph of the underlying transition structure has undirected treewidth at most two. W.l.o.g. we may assume that $A$ has only one accepting state, i.e, $A = (Q, \Sigma, \delta, q_0, \{f\})$. Then we define a nondeterministic finite automaton $A^R$ by reversing all transitions, and exchanging initial and final state. Obviously, the automaton $A^R$ accepts the reverse of the language $L(A)$.

Now we construct an automaton $B$ accepting $\frac{1}{2}L$ as follows: We apply the same product construction to $A$ and $A^R$ as for language intersection in the proof of Theorem 16, with the only modification that the set of accepting states is now $\{\, (q,q) \mid q \in Q \,\}$. The intuition behind this construction is that whenever automaton $A$ can accept a word $xy \in \Sigma^*$ with $|x| = |y|$, automaton $B$ can, while reading input $x$, starting from the initial state $q_0$ in the first component, simultaneously guess and "read" the reversed word $y^R$ starting from state $f$, by simply simulating the automaton $A^R$ on input $x$. The key point is that automaton $A^R$ behaves identically on the words $y^R$ and $x$, since they are of the same length. The construction ensures that $x$ is accepted if there is a word $y$ of the same length such both simulated computation paths meet in the middle, in the same state $q$.

When restricting our attention the above construction to the transition structure, the resulting digraph is (a subgraph of) the categorical product of two digraphs, each having $n+1$ vertices and undirected treewidth at most two. Hence the same analysis as carried out in the proof of Theorem 16 applies, for the special case that the two factor graphs have the same number $n+1$ of vertices. This gives an upper bound of $|\Sigma| \cdot 2^{O\left(1+\log \frac{n}{n}\right) \min\{n,n\}} = |\Sigma| \cdot 2^{O(n)}$, as desired. □

Thus, the technique used above is applicable for certain language operations that can be implemented on nondeterministic finite automata using a 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. For this operation, essentially the same lower bound as for intersection is obtained in [10]. The standard construction for this operation is based on a different product construction for nondeterministic finite automata, and the resulting underlying graph is isomorphic to the *cartesian* product of the underlying factor graphs. However, we are not aware of any counterpart to Lemma 15 for cartesian products of graphs.

# 6 Conclusion and Further Research

The present paper aims at deepening our understanding of the descriptional capacity of regular expressions. In particular, we studied the problem of finding good elimination orderings for the state elimination algorithm, which is one of the most popular algorithms for converting finite automata into regular expressions. In this context, we suggested two new algorithms for finding such orderings, one computing the ordering in an iterative manner, while the other uses a divide-and-conquer approach. Both will result in regular expressions that are provably way much shorter than the best known upper bound known previously, at least if the given finite automaton is deterministic and the alphabet is fixed. In particular, to our knowledge, these are the first two conversion algorithms for which a worst-case guarantee below $O(4^n)$ could be found. The latter upper bound is obtained in some sense trivial as it is attained for *any* choice of an elimination ordering. All previous conversion algorithms either covered only special cases, such as acyclic or planar finite automata, or, probably because of difficulties in analyzing the respective algorithm theoretically, were only evaluated in practice on a small set of benchmark instances. It would be interesting to see how the newly found algorithms compare to the existing ones and to each other, on a suitably chosen set of benchmark instances.

As a spinoff of the design of the second presented algorithm, we proved that a large class of finite automata admits equivalent regular expressions of polynomial size, namely those whose underlying digraphs have bounded undirected treewidth. The theorem generalizes a previous result for planar finite automata from [8]. We applied some of the techniques we found for the conversion problem to questions regarding the effect of language operations on the descriptional complexity of regular expressions; in particular, we gave a new upper bound on the minimum required regular expression size for the intersection of two regular languages. This bound is optimal in the case that both operands require roughly the same expression size, and is almost tight in the general case. That result (almost) settles an open question stated in [8]. There are many other language operations whose effect on descriptional complexity is exactly determined in the (non)deterministic finite automaton model; it would be instructive to have some

corresponding, if only asymptotically tight, bounds on how they affect the required regular expression size. In this direction, the authors have just started an investigation on some language operations related to language quotients, which are shown to result in regular expressions of polynomial size [11].

# References


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

[2] J. A. Brzozowski and E. J. McCluskey, Jr. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, 2:67–76, 1963.

[3] F. R. K. Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional Conference Series in Mathematics*. American Mathematical Society, 1997.

[4] 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.

[5] M. Domaratzki. State complexity of proportional removals. *Journal of Automata, Languages and Combinatorics*, 7(4):455–468, 2002.

[6] K. Edwards and G. Farr. Planarization and fragmentability of some classes of graphs. *Discrete Mathematics*, 2007. to appear.

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

[8] 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.

[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. Technischer Bericht TUM-I0725, Institut für Informatik, Technische Universität München, Boltzmannstraße 3, D-85748 Garching bei München, Germany, December 2007.

[11] H. Gruber and M. Holzer. Language operations with regular expressions of polynomial size. Manuscript, February 2008.

[12] H. Gruber and J. Johannsen. Optimal lower bounds on regular expression size using communication complexity. In R. Amadio, editor, *Proceedings of the* 11*th International Conference Foundations of Software Science and Computation Structures*, volume 4962 of *LNCS*, pages 273–286, Budapest, Hungary, March–April 2008. Springer.

[13] S. Gulan and H. Fernau. Local elimination strategies in automata for shorter regular expressions. Student Research Forum at SOFSEM '08, High Tatras, Slovakia, 2008.

[14] M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[15] Y.-S. Han and D. Wood. Obtaining shorter regular expressions from finite-state automata. *Theoretical Computer Science*, 370(1-3):110–120, 2007.

[16] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[17] L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):140–162, 2003.

[18] 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.

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

[20] H. V. McIntosh. REEX: A CONVERT program to realize the McNaughton-Yamada analysis algorithm. Technical Report AIM-153, MIT Artificial Intelligence Laboratory, January 1968.

[21] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRA Transactions on Electronic Computers*, 9(1):39–47, 1960.

[22] 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* 10*th Conference on Implementation and Application of Automata*, volume 3845 of *LNCS*, pages 349–350, Sophia Antipolis, France, June 2005. Springer.

[23] P. H. Morris, R. A. Gray, and R. E. Filman. Goto removal based on regular expressions. *Journal of Software Maintenance*, 9(1):47–66, 1997.

[24] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[25] J. Sakarovitch. The language, the expression, and the (small) automaton. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *Proceedings of the* 10*th Conference on Implementation and Application of Automata*, volume 3845 of *LNCS*, pages 15–30, Sophia Antipolis, France, June 2005.

[26] G. Schnitger. Regular expressions and NFAs without $\varepsilon$-transitions. In B. Durand and W. Thomas, editors, *Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 432–443, 2006.

[27] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary Report. In *ACM Symposium on Theory of Computing*, pages 1–9. ACM, 1973.

[28] P. Turán. On an extremal problem in graph theory (in Hungarian). *Matematicko Fizicki Lapok*, 48:436–452, 1941.