Enumerating Regular Expressions and Their Languages

Jonathan Lee[⋆] and Jeffrey Shallit[⋆⋆]

School of Computer Science, University of Waterloo,
Waterloo, ON N2L 3G1, Canada
`jwlee@alumni.uwaterloo.ca`
`shallit@graceland.uwaterloo.ca`

Abstract. We discuss enumeration of regular expressions and the distinct languages they represent.

1 Introduction

Regular expressions have been studied for almost fifty years, yet many interesting and challenging problems about them remain unsolved. By a regular expression, we mean a string over the alphabet

$$\Sigma \cup \{+, *, (,), \epsilon, \emptyset\}$$

that represents a regular language. For example, `(0+10)*(1+`ϵ`)` represents the language of all strings over $\{$`0,1`$\}$ that do not contain two consecutive `1`'s.

We would like to enumerate valid regular expressions and the distinct languages they represent. Enumeration of regular languages is, generally speaking, a difficult problem. For example, define $G_k(n)$ to be the number of distinct languages accepted by nondeterministic finite automata with n states over a k-letter input alphabet. The following problem, studied by Domaratzki, Kisman, and Shallit [2], seems very difficult:

Find good upper and lower bounds for $G_k(n)$.

The analogous problem for regular expressions, however, is somewhat easier. Strangely enough, it does not seem to have been studied previously. We define $R_k(n)$ to be the number of distinct languages specified by regular expressions of length n over a k-letter alphabet. The "length" of a regular expression can be defined in several different ways [3]:

- *Ordinary length*: total number of symbols, including parentheses, $\emptyset$, ϵ, etc., counted with multiplicity.
    - `(0+10)*(1+`ϵ`)` has ordinary length 12

[⋆] Research of this author supported by an NSERC Undergraduate Student Research Award.
[⋆⋆] Research of this author supported in part by NSERC.

– *Reverse polish length*: number of symbols in a reverse polish equivalent, including a symbol • for concatenation
– Equivalently, number of nodes in a syntax tree for the expression
  - `(0+10)*(1+`ϵ`)` in reverse polish would be `010•+*`ϵ`+•`
  - This has reverse polish length 10
– *Alphabetic length*: number of symbols from Σ, not including ϵ, $\emptyset$, parens, operators
  - `(0+10)*(1+`ϵ`)` has alphabetic length 4

2 Valid Regular Expressions

In this section we introduce our basic method by counting the number of valid regular expressions of (ordinary) length n. Let $S_k(n)$ be the number of such expressions over an alphabet Σ of size k. Since a regular expression is defined over the alphabet $\{\epsilon, \emptyset, (,), +, *\} \cup \Sigma$, we immediately get the trivial upper bound $R_k(n) \le S_k(n) \le (k+6)^n$. To improve our estimate for $S_k(n)$, it becomes necessary to state more precisely what a valid regular expression is.

There is some ambiguity about the definition of a valid regular expression. For example, is the empty expression valid? How about `()` or `a**`? The first two, for example, generate errors in Grail version 2.5 [7].

Surprisingly, very few textbooks, if any, define valid regular expressions properly or formally. For example, using the definition given in Martin [6–p. 86], the expression 00 is not valid, since it is not fully parenthesized. (To be fair, after the definition it is implied that parentheses can be omitted in some cases, but no formal definition of when this can be done is given.)

Probably the best way to define valid regular expressions is with a grammar. We now present a grammar for valid regular expressions:

$$\begin{aligned}
S &\to E_+ \mid E_\bullet \mid G \\
E_+ &\to E_+ + F \mid F + F \\
F &\to E_\bullet \mid G \\
E_\bullet &\to E_\bullet G \mid GG \\
G &\to E_* \mid C \mid P \\
C &\to \emptyset \mid \epsilon \mid a \quad (a \in \Sigma) \\
E_* &\to G\,* \\
P &\to (S)
\end{aligned}$$

The meaning of the variables is as follows:

– S generates all regular expressions
– E_+ generates all unparenthesized expressions where the last operator was $+$
– $E_\bullet$ generates all unparenthesized expressions where the last operator was $\cdot$ (implicit concatenation)

- E_* generates all unparenthesized expressions where the last operator was $*$ (Kleene closure)
- C generates all unparenthesized expressions where there was no last operator (i.e., the constants)
- P generates all parenthesized expressions

Here by "parenthesized" we mean there is at least one pair of enclosing parentheses. Note this grammar allows $a**$ but disallows ().

We claim this grammar is unambiguous. Because of this, we can apply the Chomsky-Schützenberger theorem [1], which states that if a language is generated by an unambiguous CFG, then the generating function

$$f(x) = \sum_{i \geq 0} |L \cap \Sigma^i| x^i$$

is algebraic over $\mathbb{Q}(x)$.

So we look at the "commutative image" of this grammar, which replaces each terminal by x, each occurrence of the empty string ϵ by 1, and each occurrence of $|$ by $+$. This gives the following system of equations:

$$\begin{aligned} S &= E_+ + E_\bullet + G \\ E_+ &= E_+ F x + F^2 x \\ F &= E_\bullet + G \\ E_\bullet &= E_\bullet G + G^2 \\ G &= E_* + C + P \\ C &= (k+2)x \quad (k = |\Sigma|) \\ E_* &= Gx \\ P &= Sx^2 \end{aligned}$$

Now we can use Gröbner bases to find the algebraic equation satisfied by S. It is

$$(x^2 + x^3)S^2 + ((k+3)x^2 + (k+3)x - 1)S + (k+2)x = 0.$$

Solving for S, we get

$$S = \frac{-(k+3)x^2 - (k+3)x + 1 - \sqrt{D}}{2(x^2 + x^3)}.$$

where the discriminant

$$D = (k+1)^2 x^4 + 2(k^2 + 4k + 5)x^3 + (k+1)(k+3)x^2 - 2(k+3)x + 1.$$

We can expand this as a power series to get a generating function enumerating the regular expressions of length n. For example, for $k = 2$, we have

$$S = 4x + 20x^2 + 120x^3 + 716x^4 + 4356x^5 + 26880x^6 + \cdots.$$

(The 20 regular expressions of length 2 are

Table 1. Number of Valid Regular Expressions

$k =$	1	2	3	4
$n = 1$	3	4	5	6
2	12	20	30	42
3	60	120	210	336
4	297	716	1465	2682
5	1509	4356	10375	21666
6	7800	26880	74340	176736
7	40962	168068	538540	1455018
8	218052	1063156	3940280	12080862

– yz
– $y*$

where $y, z \in \{\epsilon, \emptyset, \mathtt{a}, \mathtt{b}\}$.)

The number $S_k(n)$ of valid regular expressions of length n with alphabet size k is summarized in Table 1.

The asymptotic growth rate of the coefficients of the generating function for S depends on the reciprocal of the smallest zero of the discriminant D [5]. This smallest zero is, as $k \to \infty$, asymptotically equal to

$$\frac{1}{k} - \frac{6}{k^2} + \frac{42}{k^3} - \frac{1319}{4k^4} + \frac{22463}{8k^5} - \cdots ,$$

and its reciprocal is

$$k + 6 - \frac{6}{k} - \frac{167}{4k^2} - \frac{2615}{8k^3} - \cdots .$$

For $k = 1$ this the smallest zero is about .16246250262 and for $k = 2$ it is about .13755127577.

Using Darboux's method, we can prove

Theorem 1. *We have $S_k(n) \sim c_k \alpha_k^n n^{-3/2}$ for some constant c_k, where $\alpha_1 \doteq 6.1552665$ and $\alpha_2 \doteq 7.2700161767$.*

While we have counted valid regular expressions with n symbols, we are still a long way from counting the distinct languages they represent. This is because using our definition, many languages are double-counted. To improve the bound, we can attempt to improve the grammar to weed out evidently uninteresting regular expressions, such as those containing redundant parentheses.

An unambiguous grammar for regular expressions without redundant parentheses is as follows:

$$\begin{aligned} S &\to E_+ \mid E_\bullet \mid E_* \mid C \\ E_+ &\to E_+ + F \mid F + F \\ F &\to E_\bullet \mid E_* \mid C \\ E_\bullet &\to E_\bullet G \mid GG \\ G &\to (E_+) \mid E_* \mid C \end{aligned}$$

$$\begin{aligned} C &\to \emptyset \mid \epsilon \mid a \quad (a \in \Sigma) \\ E_* &\to (E_+)* \mid (E_\bullet)* \mid E_**\mid C* \end{aligned}$$

We can mimic the analysis given for the previous grammar. For $k = 2$, we get the equation $(x^9 + x^6)S^4 + (2x^6 + 5x^5 + x^3)S^3 + (5x^4 + 5x^3 + 5x^2 - x)S^2 + (8x^2 + 5x - 1)S + 4x = 0$ which has the power series solution

$$S = 4 + 20x^2 + 116x^3 + 660x^4 + 3780x^5 + 21844x^6 + \cdots .$$

The discriminant is a polynomial of degree 30, the smallest positive root is .146378001, and the asymptotic growth rate is $O(6.832^n)$.

Maple worksheets for these examples are available at

`http://www.cs.uwaterloo.ca/~shallit/papers.html` .

Using more complicated grammars, we can dramatically improve these bounds; we do this in Section 5. For now, however, we turn to lower bounds.

3 Lower Bounds

We now turn to lower bounds on $R_k(n)$.

In the unary case ($k = 1$), we can argue as follows: consider any subset of $\{\epsilon, a, a^2, \ldots, a^{t-1}\}$. Such a subset can be denoted by a regular expression of (ordinary) length at most $t(t+1)/2$. Since there are 2^t distinct subsets, this gives a lower bound of $R_1(n) \geq 2^{\sqrt{2n}-1}$. Similarly, when $k \geq 2$, there are k^n distinct strings of length n, so $R_k(n) \geq k^n$.

However, these naive bounds can be improved somewhat using a grammar-based approach.

Consider a regular expression of the form

$$x_1(\epsilon + x_2(\epsilon + x_3(\epsilon + ...)))$$

where the x_i denote nonempty words. Every distinct choice of the x_i specifies a distinct language. Such expressions can be generated by the grammar

$$\begin{aligned} S &\to Y \mid Y(\epsilon + S) \\ Y &\to aY \mid a, \qquad a \in \Sigma \end{aligned}$$

which has the commutative image

$$\begin{aligned} S &= Y + YSx^4 \\ Y &= kxY + kx. \end{aligned}$$

The solution to this system is

$$S = \frac{kx}{1 - kx - kx^5}.$$

Once again, the asymptotic behavior of the coefficients of the power series for S depend on the zeros of $1 - kx - kx^5$. The smallest (indeed, the only) real root is, asymptotically as $k \to \infty$, given by

$$\sum_{i \geq 0} \frac{(-1)^i \binom{5i}{i}}{4i+1} k^{-(4i+1)} = \frac{1}{k} - \frac{1}{k^5} + \frac{5}{k^9} - \frac{35}{k^{13}} + \cdots .$$

The reciprocal of this series is

$$\sum_{i \geq 0} \frac{4\binom{5i+5}{i+1}}{5(5i+4)} k^{1-4i} = k + \frac{1}{k^3} - \frac{4}{k^7} + \frac{26}{k^{11}} - \frac{204}{k^{15}} + \frac{1771}{k^{19}} - \cdots .$$

For $k = 1$ the only real root of $1 - kx - kx^5$ is approximately .754877666 and for $k = 2$ it is about .4756527435. Thus we have

Theorem 2. *$R_1(n) = \Omega(1.3247^n)$ and $R_2(n) = \Omega(2.102374^n)$.*

We now turn to improving these lower bounds.

4 Better lower bounds

4.1 Trie representations of finite languages

We begin by describing how to represent non-empty finite languages not containing ϵ via a trie structure; an example is given Fig. 1.

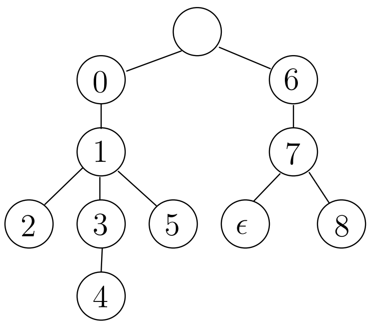

Fig. 1. Trie representation for `01(2+34+5)+6(`ϵ`+7)`

Algorithm 1 below takes as input a finite non-empty language L not containing ϵ and returns a trie in our desired format. The words in such a language L correspond to the leaf nodes of the trie for L; moreover, the concatenation of labels from the root to a leaf node gives an expression for the word associated with that leaf node. For regular languages L_1 and L_2, we write $L_2^{-1}L_1$ to denote the left quotient of L_1 by L_2; formally

$$L_2^{-1}L_1 = \{x \ : \ \text{there exists } y \in L_2 \text{ such that } yx \in L_1\}.$$

Algorithm 1 CREATE-TRIE(L)

```
Require: $\epsilon \notin L$, $L \neq \emptyset$
 1: create a tree $T$ with an unlabelled root
 2: for all $a \in \Sigma$ such that $a^{-1}L \neq \emptyset$ do
 3:    add the subtree returned by CREATE-TRIE-HELP($\{a\}^{-1}L$, $a$) as a child of the
       root of $T$
 4: end for
 5: return $T$
```

Algorithm 2 CREATE-TRIE-HELP(L, a)

```
 1: create a tree $T$ with a root labelled $a$
 2: if $L \neq \{\epsilon\}$ then {need to create children}
 3:    for all $b \in \Sigma$ such that $b^{-1}L \neq \emptyset$ do
 4:       add the subtree returned by CREATE-TRIE-HELP($\{b\}^{-1}L$, $b$) as a child of
          the root of $T$
 5:    end for
 6:    if $\epsilon \in L$ then
 7:       add a node labelled $\epsilon$ as a child of the root of $T$
 8:    end if
 9: end if
10: return $T$
```

4.2 Star-free regular expressions

We begin with the simple problem of counting the number of regular languages that may be specified by regular expressions of length n.

We develop lower bounds by specifying a context-free grammar that generates regular expressions, factoring out common prefixes in a style similar to Horner's rule. In fact, the grammar is designed so that if r is a regular expression generated by the grammar, then the structure of r mimics that of the trie for $L(r)$ — nodes with a single child correspond to concatenations while nodes with multiple children correspond to concatenations with a union. For notational convenience, we take our alphabet to be $\Sigma = \{a_0, a_1, \ldots, a_{k-1}\}$, where $k \geq 1$ denotes our alphabet size.

$$
\begin{aligned}
&S \to Y \mid Z \\
&E \to Y \mid (Z) \mid (\epsilon + S) \\
&Y \to P_i \text{ for } 0 \leq i < k \\
&Z \to P_{n_0} + P_{n_1} + \cdots + P_{n_t} \text{ where } 0 \leq n_0 < n_1 < \cdots < n_t < k \text{ for } t > 0 \\
&P_i \to a_i \mid a_i E \text{ for } 0 \leq i < k
\end{aligned}
$$

The set of regular languages represented corresponds to all non-empty finite languages over Σ not containing the empty string ϵ. We briefly describe the non-terminals:

S generates all non-empty finite languages not containing ϵ — this corresponds to Algorithm 1.

E generates all non-empty finite languages containing at least one word other than ϵ — this corresponds to line 2 of Algorithm 2.

Y generates all non-empty finite languages (not containing ϵ) whose words all begin with the same letter — this corresponds to line 2 of Algorithm 1 and line 3 of Algorithm 2 when the body of the `for` loop is executed only once.

Z generates all non-empty finite languages (not containing ϵ) whose words do not all begin with the same letter — this corresponds to line 2 of Algorithm 1 and line 3 of Algorithm 2 when the body of the `for` loop is executed more than once.

P_i generates all non-empty finite languages (not containing ϵ) whose words all begin with a_i — this corresponds to line 1 of Algorithm 2.

We remark that this grammar is unambiguous and that no regular language is represented more than once; this should be clear from the relationship between regular expressions generated by the grammar and their respective tries.

(Note that it is possible to slightly optimize this grammar in the case of ordinary length to generate expressions such as $0 + 00$ in lieu of $0(\epsilon + 0)$, but as it results in marginal improvements to the lower bound at the cost of greatly complicating the grammar, we do not do so here.)

To obtain bounds, we make use of the following result adapted from Klarner and Woodworth [5].

Theorem 3. *Suppose it is known that the coefficients of an algebraic power series $F(x)$ grow at a rate of $\Omega(\alpha^n)$ and $O(\beta^n)$ for $\alpha < \beta$. Suppose also there is a polynomial $P \in \mathbb{Z}[x, y]$ such that $P(x, F) = 0$ and whose discriminant δ with respect to y has only one root γ in the interval $[1/\beta, 1/\alpha]$. Then modulo some sub-exponential growth rate, the coefficients of $F(x)$ grow asymptotically like $1/\gamma^n$.*

Proof. We may assume that F has a radius of convergence $0 < R < \infty$. Klarner and Woodworth deduce that R is a singularity of F, so it suffices to show that γ is the smallest positive singularity of F; that is, R. Suppose there exists a smallest positive singularity $\gamma' < \gamma$. By assumption, $\gamma' < 1/\beta$ so $1/\gamma' > \beta$. However, γ' determines the radius of convergence of F, contradicting the fact that the coefficient growth is $O(\beta^n)$. It remains to show that γ is indeed a singularity. Since the coefficient growth is $\Omega(\alpha^n)$, there must be a singularity less than $1/\alpha$. By assumption, this must be γ. ■

Table 2 lists the lower bounds obtained through this grammar.

Remark 1. By virtue of Theorem 3, these lower bounds were obtained by bootstrapping off the trivial bounds of $\Omega(k^n)$, $\Omega(k^{n/2})$ and $\Omega(k^n)$ for the ordinary, reverse polish and alphabetic length cases, respectively.

Asymptotic analysis for alphabetic length. We first state a version of the Lagrange implicit function theorem as a simplification of [4–Theorem 1.2.4]. If $f(t)$ is a power series in t, we write $[t^n]f(t)$ to denote the coefficient of t^n in $f(t)$.

Table 2. Lower bounds for $R_k(n)$ with respect to length measure and alphabet size

	ordinary	reverse polish	alphabetic
1	$\Omega(1.3247^n)$	$\Omega(1.2720^n)$	$\Omega(2^n)$
2	$\Omega(2.5676^n)$	$\Omega(2.1532^n)$	$\Omega(6.8284^n)$
3	$\Omega(3.6130^n)$	$\Omega(2.7176^n)$	$\Omega(11.1961^n)$
4	$\Omega(4.6260^n)$	$\Omega(3.1806^n)$	$\Omega(15.5307^n)$
5	$\Omega(5.6264^n)$	$\Omega(3.5834^n)$	$\Omega(19.8548^n)$
6	$\Omega(6.6215^n)$	$\Omega(3.9451^n)$	$\Omega(24.1740^n)$

Lemma 1. *Let R be a commutative ring of characteristic 0 and take $\phi(\lambda) \in R[[\lambda]]$ such that $[\lambda^0]\phi$ is invertible. Then there exists a unique formal power series $w(t) \in R[[t]]$ such that $[t^0]w = 0$ and $w = t\phi(w)$. For $n \geq 1$,*

$$[t^n]w(t) = \frac{1}{n}[\lambda^{n-1}]\phi^n(\lambda) .$$

Due to the simplicity of alphabetic length, the problem of enumerating regular languages in this case may be interpreted as doing so for rooted k-ary trees, where each internal node is marked with one of two possible colours. We thus investigate how our lower bound varies with k.

More specifically, consider a regular expression r generated by the grammar from the previous section and its associated trie. Colour each node with a child labelled ϵ black and all other nodes white. After deleting all nodes marked ϵ, call the resultant tree $T(r)$. This operation is reversible, and shows that we may put the expressions of alphabetic length n in correspondence with the k-ary rooted trees with $n+1$ vertices where every non-root internal node may assume one of two colours. In order to estimate the latter, we first prove a basic result:

Lemma 2. *There are $\frac{1}{n}\binom{kn}{n-1}$ k-ary trees of n nodes. Moreover, the expected number of leaf nodes among k-ary trees of n nodes is asymptotic to $(1-1/k)^k n$ as $n \to \infty$.*

Proof. Fix $k \geq 1$. For $n \geq 1$, let a_n denote the number of k-ary rooted trees with n vertices and consider the generating series:

$$f(t) = \sum_{n \geq 1} a_n t^n .$$

By the recursive structure of k-ary trees, we have the recurrence:

$$f(t) = t(1 + f(t))^k .$$

Thus, by the Lagrange implicit function theorem, we have

$$\begin{aligned} a_n &= [t^n]f(t) \\ &= \frac{1}{n}[\lambda^{n-1}](1+\lambda)^{kn} \\ &= \frac{1}{n}\binom{kn}{n-1} . \end{aligned}$$

We now calculate the number of leaf nodes among all k-ary rooted trees with n vertices. Let $b_{n,m}$ denote the number of k-ary rooted trees with n vertices and m leaf nodes and c_n the number of leaf nodes among all k-ary rooted trees with n vertices. Consider the bivariate generating series:

$$g(s,t) = \sum_{n,m \geq 1} b_{n,m} s^m t^n \,.$$

By the recursive structure of k-ary trees, we have the recurrence:

$$g(s,t) = t(s - 1 + (1 + g(s,t))^k) \,.$$

The Lagrange implicit function theorem once again yields

$$\begin{aligned} c_n &= \frac{\partial}{\partial s}[t^n] g(s,t)\bigg|_{s=1} \\ &= \frac{\partial}{\partial s}\frac{1}{n}[\lambda^{n-1}](s - 1 + (1 + g(s,t))^k)^n\bigg|_{s=1} \\ &= \frac{1}{n}[\lambda^{n-1}]\ \frac{\partial}{\partial s}(s - 1 + (1+\lambda)^k)^n\bigg|_{s=1} \\ &= [\lambda^{n-1}](1+\lambda)^{k(n-1)} \\ &= \binom{k(n-1)}{n-1} . \end{aligned}$$

Thus, the expected number of leaf nodes among n-node trees is

$$\begin{aligned} \frac{c_n}{a_n} &= \frac{n\binom{k(n-1)}{n-1}}{\binom{kn}{n-1}} \\ &= \frac{n(kn-n+1)(kn-n)\cdots(kn-k-n+2)}{(kn)(kn-1)\cdots(kn-k+1)} \\ &\sim n\left(\frac{k-1}{k}\right)^k \quad \text{as } n \to \infty \text{ for fixed } k. \end{aligned}$$

■

We wish to find a bound on the expected number of subsets of non-root internal nodes among all k-ary rooted trees with n nodes, where a subset corresponds to those nodes marked black. Fix $k \geq 2$. Since the map $x \mapsto 2^x$ is convex, for every $\varepsilon > 0$ and sufficiently large n, Jensen's inequality (e.g., [8–Thm. 3.3]) applied to the lemma above implies the following lower bound on the number of subsets:

$$2^{(1-(1-1/k)^k-\varepsilon)n} \,.$$

Since $-(1-1/k)^k > -1/e$ for $k \geq 1$, we may choose $\varepsilon > 0$ such that

$$-(1-1/k)^k - \varepsilon > -1/e \,.$$

This yields a lower bound of

$$2^{(1-1/e)n} \,.$$

Assuming $k \geq 2$, we now estimate $\binom{kn}{n-1}$. By Stirling's formula, we have that as $n \to \infty$,

$$\begin{aligned}\binom{kn}{n-1} &= \frac{(kn)!}{n!\,((k-1)n)!}\frac{n}{(k-1)n+1} \\ &\sim \frac{\sqrt{2\pi kn}\,(kn/e)^{kn}}{\sqrt{2\pi n}\,(n/e)^n\,\sqrt{2\pi(k-1)n}\,((k-1)n/e)^{(k-1)n}}\frac{n}{(k-1)n+1} \\ &= \Theta\left(\left(\frac{k^k}{(k-1)^{k-1}}\right)^n\right).\end{aligned}$$

Putting our two bounds together, we have the following lower bound as $n \to \infty$ on the number of star-free regular expressions of alphabetic length n:

$$\Omega\left(\left(\frac{2^{(1-1/e)}k^k}{(k-1)^{k-1}}\right)^n\right) \text{ where the implied constants depend only on } k.$$

4.3 General regular languages

We now turn our attention to enumerating regular languages in general; that is, we allow for regular expressions with Kleene stars.

Our grammars for this section are based on the those for the star-free cases. Due to the difficulty of avoiding specifying duplicate regular languages, we settle for a "small" subset of regular languages. For simplicity, we only consider taking the Kleene star closure of singleton alphabet symbols.

Recall the trie representation of a star-free regular expression written in our common prefix notation. With this representation, we may mark nodes with stars while satisfying the following conditions:

- each starred symbol must have a non-starred parent other than the root;
- a starred symbol may not have a sibling or an identically-labelled parent (disregarding the lack of star) with its own sibling; and
- a starred symbol may not have an identically-labelled child (disregarding the lack of star).

The first condition eliminates duplicates such as

$$\texttt{0*11*0*1*0*} \leftrightarrow \texttt{0*1*0*11*0*};$$

the second eliminates those such as

$$\texttt{01*} \leftrightarrow \texttt{0(}\epsilon\texttt{+11*)} \text{ and } \texttt{0(1+2*1)} \leftrightarrow \texttt{02*1}$$

and the third eliminates those such as

$$\texttt{0*0} \leftrightarrow \texttt{00*}.$$

In this manner, we end up with starred tries such as in Fig. 2.

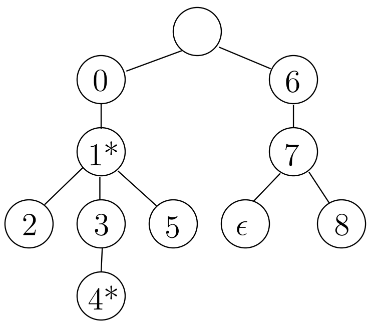

Fig. 2. Trie representation for `01*(2+34*+5)+6(`ϵ`+7)`

Algorithm 3 CREATE-STAR-TRIE(L)

```
Require: $\epsilon \notin L$, $L \neq \emptyset$
 1: create a tree $T$ with an unlabelled root
 2: for all $a \in \Sigma$ such that $\{a\}^{-1}L \neq \emptyset$ do
 3:   add the subtree returned by CREATE-STAR-TRIE-HELP($\{a\}^{-1}L$, $a$) as a child
      of the root of $T$
 4: end for
 5: return $T$
```

Algorithm 4 CREATE-STAR-TRIE-HELP(L, a)

```
 1: create a tree $T$ with a root labelled $a$
 2: if exists $b \in \Sigma$ such that $\{b^n\}^{-1}L \cap (\epsilon + (\Sigma \setminus \{b\})\Sigma^*) \neq \emptyset$ for all $n \geq 0$ then {need
    a starred child labelled $b^*$}
 3:   attach a child labelled $b^*$ to the root of $T$
 4:   if $L \neq b^*$ then {starred child will be an internal node}
 5:     for all $c \in \Sigma \setminus \{b\}$ such that $\{c\}^{-1}L \neq \emptyset$ do {determine children}
 6:       add the subtree returned by CREATE-STAR-TRIE-HELP($\{c\}^{-1}L$, $c$) as a
          child of the node labelled $b^*$
 7:     end for
 8:     if $b \in L$ then
 9:       add a node labelled $\epsilon$ as a child of the node labelled $b^*$
10:     end if
11:   end if
12: else
13:   for all $b \in \Sigma$ such that $\{b\}^{-1}L \neq \emptyset$ do {need an unstarred child labelled $b$}
14:     add the subtree returned by CREATE-STAR-TRIE-HELP($\{b\}^{-1}L$, $b$) as a
        child of the root of $T$
15:   end for
16: end if
17: if $\epsilon \in L$ and the root of $T$ has at least one unstarred child then
18:   add a node labelled $\epsilon$ as a child of the root of $T$
19: end if
20: return $T$
```

Algorithm 3 illustrates how to recreate such a starred trie from the language it specifies.

Let T be any starred trie satisfying the conditions above. Then T represents a regular expression, which in turn specifies a certain language. We now show that when the algorithm is run with that language as input, it returns the trie T by arguing that at each step of the algorithm when a particular node (matched with language L if the root and aL otherwise) is being processed, the correct children are determined.

We first consider children of the root. By the original trie construction (for finite languages without ϵ), no such children may be labelled ϵ. Thus, by the first star condition, the only children may be unstarred alphabet symbols. Thus, line 2 of Algorithm 3 suffices to find all children of the root correctly.

Now consider a non-root internal node, say labelled a. By the third star condition, a starred node may not have a child labelled with the same alphabet symbol, so if a has a child labelled b^*, then

$$\{b^n\}^{-1}L \cap (\epsilon + (\Sigma \setminus \{b\})\Sigma^*) \text{ is non-empty for all } n \geq 0.$$

Conversely, by the second condition, a starred node may not have an identically-labelled parent that has ϵ as a sibling, so if

$$\{b^n\}^{-1}L \cap (\epsilon + (\Sigma \setminus \{b\})\Sigma^*)$$

is non-empty for all $n \geq 0$, then a must have a child labelled b^*. By the second star condition, a starred node may not have siblings, so the algorithm need not check for other children once a starred child is found. This shows that line 2 of Algorithm 4 correctly finds all children in the case of a starred child.

Assuming a has a starred child b^*, then by the third condition, line 5 of Algorithm 4 correctly determines all children of b^*.

Otherwise, a has no starred children, and line 13 of Algorithm 4 suffices to find all children.

We give a grammar that generates expressions meeting these conditions. As before, we take our alphabet to be $\{a_0, a_1, \ldots, a_{k-1}\}$.

$$\begin{aligned}
S &\to Y \mid Z \\
E &\to Y \mid (Z) \mid (\epsilon + Y') \mid (\epsilon + Z) \\
E_i &\to Y_i \mid (Z_i) \mid (\epsilon + Y_i') \mid (\epsilon + Z_i) \text{ for } 0 \leq i < k \\
Y &\to P_i \text{ for } 0 \leq i < k \\
Y' &\to P_i' \text{ for } 0 \leq i < k \\
Y_i &\to P_j \text{ for } 0 \leq i, j < k \text{ and } i \neq j \\
Y_i' &\to P_j' \text{ for } 0 \leq i, j < k \text{ and } i \neq j \\
Z &\to P_{n_0}' + P_{n_1}' + \cdots + P_{n_t}' \text{ where } 0 \leq n_0 < n_1 < \cdots < n_t < k \text{ for } t > 0 \\
Z_i &\to P_{n_0}' + P_{n_1}' + \cdots + P_{n_t}' \text{ as above, but with } n_j \neq i \text{ for all } 0 \leq j \leq t \\
P_i &\to a_i \mid a_i E \mid a_i a_j^* \mid a_i a_j^* E_j \text{ for } 0 \leq i, j < k \\
P_i' &\to a_i \mid a_i E \mid a_i a_j^* \mid a_i a_j^* E_j \text{ for } 0 \leq i, j < k \text{ and } i \neq j
\end{aligned}$$

We describe the non-terminals.

S generates all expressions — this corresponds to Algorithm 3.

E, E_i generate expressions that may be concatenated to non-starred and starred alphabet symbols, respectively. The non-terminal E corresponds to lines 2 and 13 while E_i corresponds to line 5 of Algorithm 4. These act the same as S except for the introduction of parentheses to take precedence into account and restriction that no prefixes of the form $\epsilon + aa^*$ are generated, used to implement the second condition.
Additionally, E_i has the restriction that its first alphabet symbol produced may not be a_i — this is used to implement the third condition.

Y, Y', Y_i, Y_i' generate expressions whose prefix is an alphabet symbol. As a whole, these non-terminals correspond to Algorithm 4, and may be considered degenerate cases of Z and Z_i; that is, trivial unions.
The tick-mark signifies that expressions of the form aa^* for $a \in \Sigma$ are disallowed, used to implement the second condition. The subscripted i signifies that the initial alphabet symbol may not be a_i, used to implement the third condition.

Z, Z_i generate non-trivial unions of expressions beginning with distinct alphabet symbols — Z corresponds to line 2 of Algorithm 3 and line 13 of Algorithm 4, while Z_i corresponds to line 5 of Algorithm 4.
The subscripted i signifies that none of initial alphabet symbols may be a_i, used to implement the third condition.

P_i, P_i' generate expressions beginning with the specified alphabet symbol a_i. They correspond to line 1 of Algorithm 4.
The tick-mark signifies that expressions may not have the prefix $a_i a_i^*$, used to implement the second condition.

Since the algorithm correctly returns a trie when run on the language represented by the trie, the correspondence between the algorithm and the grammar gives us the following result.

Theorem 4. *The grammar above is unambiguous and the generated regular expressions represent distinct regular languages.*

Table 3 lists the improved lower bounds for $R_k(n)$.

Remark 2. These lower bounds were obtained via Theorem 3, boot-strapping off the bounds in Table 2.

Table 3. Improved lower bounds for $R_k(n)$ with respect to length measure and alphabet size

	ordinary	reverse polish	alphabetic
1	$\Omega(1.3247^n)$	$\Omega(1.2720^n)$	$\Omega(2^n)$
2	$\Omega(2.7799^n)$	$\Omega(2.2140^n)$	$\Omega(7.4140^n)$
3	$\Omega(3.9582^n)$	$\Omega(2.8065^n)$	$\Omega(12.5367^n)$
4	$\Omega(5.0629^n)$	$\Omega(3.2860^n)$	$\Omega(17.6695^n)$
5	$\Omega(6.1319^n)$	$\Omega(3.6998^n)$	$\Omega(22.8082^n)$
6	$\Omega(7.1804^n)$	$\Omega(4.0693^n)$	$\Omega(27.9500^n)$

5 Better upper bounds

Turning our attention back to upper bounds, we develop grammars for regular expressions such that every regular language is represented by at least one shortest regular expression generated by the grammar, where a regular expression R of length n is said to be shortest if there are no expressions R' of length less than n with $L(R) = L(R')$.

In increasing order of precedence, the operations on regular languages are union, concatenation and Kleene-star closure, which we denote by the symbols $+$, $\bullet$ and $*$, respectively. In our grammars for this section, we will denote these by the non-terminals A, B and C, respectively.

As $+$ and $\bullet$ are associative, we will consider them to be variadic operators taking at least 2 arguments and impose the condition that in any parse tree (see Fig. 3), neither of them are permitted to have themselves as children.

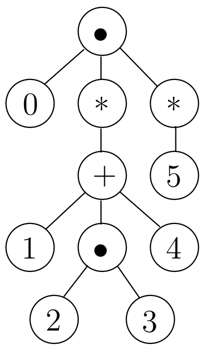

Fig. 3. Parse tree for `0(1+23+4)*5*`

Taking into mind associativity, we start with the following unambiguous grammar as our basis for regular expressions — note that this is different from the grammar given previously.

$S \to A \mid B \mid C \mid D \mid \epsilon \mid \emptyset$ (start symbol)

$A \to A_0 + A_0 \mid A_0 + A$ (union)
$A_0 \to B \mid C \mid D \mid \epsilon$

$B \to B_0\, B_0 \mid B_0\, B$ (concatenation)
$B_0 \to (A) \mid C \mid D$

$C \to (A)^* \mid (B)^* \mid D^*$ (Kleene-star closure)

$D \to a$ for $a \in \Sigma$ (alphabet symbols) .

We claim that each regular language has at least one shortest regular expression that is generated by the grammar; our immediate aim is to modify the grammar to generate fewer regular expressions per regular language while still generating at least one shortest one.

There are a few possible optimizations for removing duplicate regular languages.

– In all cases, by the commutativity of $+$ (viewed as a variadic operator), we may impose the condition that its operands appear in the following order:
  1. the symbol ϵ;
  2. starred expressions (those generated by C);
  3. concatenated expressions (those generated by B); then
  4. singleton alphabet symbols (those generated by D).

  Also, given that for languages $L_1, L_2, \ldots, L_n$ we have

$$(L_1 + L_2 + \cdots L_n)^* = (L_1^* + L_2^* + \cdots + L_n^*)\,,$$

  we impose the restriction that whenever $+$ is an operand of $*$, none of its own operands may be $*$ or ϵ. Otherwise, at most one operand may be ϵ, provided none of the other operands are $*$. These conditions prevent expressions such as $(a^*)^*$, $(\epsilon + a)^*$ and $\epsilon + a^*$ when a^* will suffice.
  We implement these two conditions by modifying the "A"-productions to:

$$\begin{aligned}
A &\to \epsilon + A_B \mid C + A_C \mid B + A_B \mid D + A_D \qquad \text{(union)}\\
A' &\to B + A_B \mid D + A_D\\
A_C &\to C \mid C + A_C \mid A_B\\
A_B &\to B \mid B + A_D \mid A_D\\
A_D &\to D \mid D + A_D\,.
\end{aligned}$$

  In addition, we modify the "C"-production to:

$$C \to (A')^* | (B)^* | D^* \qquad \text{(Kleene-star closure)}.$$

– In the unary cases, by the commutativity of $\bullet$ (viewed as a variadic operator), we may impose a similar condition as for $+$; namely, that its operands appear in the following order:
  1. starred expressions (those generated by C);
  2. united expressions (those generated by A); then
  3. single alphabet symbols (those generated by D).

  We also impose the condition that at most one such operand may be a starred expression. For if we have an operand of the form a^* for $a \in \Sigma$, all other starred expressions are redundant. Otherwise, due to precedence, we may assume we have operands of the form (r_1) and (r_2) for regular expressions r_1 and r_2, so with respect to ordinary length,

$$|(r_1 + r_2)^*| = |(r_1)^*(r_2)^*| - 2\,.$$

Note that in the alphabetic and reverse polish length cases, this condition is implied by the one we will state next.
We implement these conditions by modifying the "B"-productions to:

$$\begin{aligned} B &\to CB_A \mid (A)B_A \mid DB_D \qquad \text{(concatenation)} \\ B_A &\to (A) \mid (A)B_A \mid B_D \\ B_D &\to D \mid DB_D \,. \end{aligned}$$

– In the alphabetic and reverse polish measure of length cases, we impose the restriction that no two adjacent operands of $\bullet$, the concatenation operator, are starred expressions. For if r_1, r_2 denote regular expressions, then $r_1^* r_2^* = (r_1 + r_2)^*$. Furthermore, with respect to alphabetic length,

$$|(r_1 + r_2)^*| = |r_1^* r_2^*|;$$

and with respect to reverse polish length,

$$|(r_1 + r_2)^*| = |r_1^* r_2^*| - 1.$$

We implement these conditions by modifying the "B"-productions to:

$$\begin{aligned} B &= (A)B_0 \mid CB_1 \mid DB_0 \qquad \text{(concatenation)} \\ B_0 &= (A) \mid C \mid D \mid (A)B_0 \mid CB_1 \mid DB_0 \\ B_1 &= (A) \mid D \mid AB_0 \mid DB_0 \,. \end{aligned}$$

With these enhancements, we obtain improved upper bounds on $R_k(n)$, as listed in Table 4.

Table 4. Improved upper bounds for $R_k(n)$ with respect to length measure and alphabet size

	ordinary	reverse polish	alphabetic
1	$O(2.9090^n)$	$O(2.7037^n)$	$O(21.7527^n)$
2	$O(4.2198^n)$	$O(3.9675^n)$	$O(62.9522^n)$
3	$O(5.3182^n)$	$O(4.6899^n)$	$O(94.4282^n)$
4	$O(6.4068^n)$	$O(5.2957^n)$	$O(125.9043^n)$
5	$O(7.4736^n)$	$O(5.8276^n)$	$O(157.3804^n)$
6	$O(8.5261^n)$	$O(6.3074^n)$	$O(188.8564^n)$

Remark 3. In the case of reverse polish length, the least positive root was less than $1/(k+4)$, meaning it could be safely ignored by Theorem 3. In the ordinary unary case, the bound was boot-strapped from the bound obtained from relaxing the operand ordering of $+$; in the ordinary non-unary cases, the restriction disallowing ϵ and starred expressions to be siblings was dropped due to apparent computational infeasibility.

By discarding the productions for the non-terminal C altogether, we obtain upper bounds for the star-free analogue of $R_k(n)$, as shown in Table 5.

Table 5. Improved upper bounds for star-free $R_k(n)$ with respect to length measure and alphabet size

	ordinary	reverse polish	alphabetic
1	$O(2.4702^n)$	$O(2.4495^n)$	$O(14.6032^n)$
2	$O(3.5051^n)$	$O(3.3096^n)$	$O(29.2063^n)$
3	$O(4.5681^n)$	$O(3.9837^n)$	$O(43.8095^n)$
4	$O(5.6208^n)$	$O(4.5579^n)$	$O(58.4126^n)$
5	$O(6.6629^n)$	$O(5.0670^n)$	$O(73.0158^n)$
6	$O(7.6969^n)$	$O(5.5292^n)$	$O(87.6189^n)$

Remark 4. For the unary reverse polish and alphabetic length cases, it can be shown directly that the lower bounds given in Table 2 are indeed upper bounds as well.

6 Exact Enumerations

Tables 6 to 11 give exact numbers for the number of regular languages representable by a regular expression of length n, but not by any of length less than n.

Table 6. Star-free ordinary cases

	1	2	3	4
1	3	4	5	6
2	1	4	9	16
3	2	11	33	74
4	3	28	117	336
5	3	63	391	1474
6	5	156	1350	6560
7	5	358	4546	28861
8	8	888	15753	128720
9	9	2194	55053	578033
10	14	5665	196185	

Table 7. Star-free reverse polish cases

	1	2	3	4
1	3	4	5	6
3	2	7	15	26
5	3	25	85	202
7	5	109	589	1917
9	9	514	4512	20251
11	14	2641	37477	
13	24	14354	328718	
15	41	81325	231152	
17	71	475936		
19	118			

Table 8. Star-free alphabetic cases

	1	2	3	4
0	2	2	2	2
1	2	4	6	8
2	4	24	60	112
3	8	182	806	2164
4	16	1652	13182	51008
5	32	16854	242070	1346924
6	64	186114		

Table 9. General ordinary cases

	1	2	3	4
1	3	4	5	6
2	2	6	12	20
3	3	17	48	102
4	4	48	192	520
5	5	134	760	2628
6	9	397	3090	13482
7	12	1151	12442	68747
8	17	3442	51044	354500
9	25	10527	211812	
10	33	32731	891228	

Table 10. General reverse polish cases

	1	2	3	4
1	3	4	5	6
2	1	2	3	4
3	2	7	15	26
4	2	13	33	62
5	3	32	106	244
6	4	90	361	920
7	6	189	1012	3133
8	7	580	3859	13529
9	11	1347	11655	48388
10	15	3978	43431	208634

We explain how these numbers were obtained. Using the upper bound grammars described previously, a dynamic programming approach was taken to produce (in order of increasing regular expression size) the regular expressions generated by each non-terminal. To account for duplicates, each regular expression was transformed into a DFA, minimized and relabelled via a breadth-first search to produce a canonical representation. Using these representations as hashes, any regular expression matching a previous one generated by the same non-terminal was simply ignored.

Table 11. General alphabetic cases

	1	2	3	4
0	2	2	2	2
1	3	6	9	12
2	6	56	150	288
3	14	612	3232	9312
4	30	7923	82614	357911
5	72	114554		
6	155	1768133		
7	343			
8	731			
9	1600			
10	3407			

A Commutative images

We provide the commutative images for the grammars described. We do not explicitly provide images for the star-free cases, as they may be obtained from those below by simply ignoring the images of sentential forms containing a star.

We also set one of $\delta_o, \delta_r, \delta_a$ to 1, depending on whether the ordinary, reverse polish or alphabetic case is being considered, respectively — all others are set to 0. As usual, k denotes the alphabet size.

A.1 Lower bounds

$$\begin{aligned}
S &= Y + Z \\
E &= Y + Zx^{2\delta_o} + Y'x^{4\delta_o+2\delta_r} + Zx^{4\delta_o+2\delta_r} \\
E_N &= Y_N + Z_N x^{2\delta_o} + Y'_N x^{4\delta_o+2\delta_r} + Z_N x^{4\delta_o+2\delta_r} \\
Y &= kP_N \\
Y' &= kP'_N \\
Y_N &= (k-1)P_N \\
Y'_N &= (k-1)P'_N \\
Z &= \sum_{i=2}^{k} \binom{k}{i} P'^{i}_N x^{(\delta_o+\delta_r)(i-1)} \\
Z_N &= \sum_{i=2}^{k-1} \binom{k-1}{i} P'^{i}_N x^{(\delta_o+\delta_r)(i-1)} \\
P_N &= x + Ex^{1+\delta_r} + kx^{2+\delta_o+2\delta_r} + kE_N x^{2+\delta_o+3\delta_r} \\
P'_N &= x + Ex^{1+\delta_r} + (k-1)x^{2+\delta_o+2\delta_r} + (k-1)E_N x^{2+\delta_o+3\delta_r} .
\end{aligned}$$

A.2 Upper bounds

$$\begin{aligned}
S &= A + B + C + D + 2(\delta_o + \delta_r)x \qquad \text{(start symbol)}\\
A &= A_B x^{2(\delta_o+\delta_r)} + CA_C x^{\delta_o+\delta_r} + BA_B x^{\delta_o+\delta_r} + DA_D x^{\delta_o+\delta_r} \qquad \text{(union)}\\
A' &= BA_B x^{\delta_o+\delta_r} + DA_D x^{\delta_o+\delta_r}\\
A_C &= C + CA_C x^{\delta_o+\delta_r} + A_B\\
A_B &= B + BA_D x^{\delta_o+\delta_r} + A_D\\
A_D &= D + DA_D x^{\delta_o+\delta_r}\\
C &= A' x^{3\delta_o+\delta_r} + B x^{3\delta_o+\delta_r} + D x^{\delta_o+\delta_r} \qquad \text{(Kleene-star closure)}\\
D &= kx \qquad \text{(alphabet symbols)}
\end{aligned}$$

In the unary cases:

$$\begin{aligned}
B &= CB_A x^{\delta_r} + AB_A x^{2\delta_o+\delta_r} + DB_D x^{\delta_r} \qquad \text{(concatenation)}\\
B_A &= Ax^{2\delta_o} + AB_A x^{2\delta_o+\delta_r} + B_D\\
B_D &= D + DB_D x^{\delta_r}
\end{aligned}$$

In the ordinary, non-unary cases:

$$\begin{aligned}
B &= AB_0 x^2 + CB_0 + DB_0 \qquad \text{(concatenation)}\\
B_0 &= Ax^2 + C + D + AB_0 x^2 + CB_0 + DB_0
\end{aligned}$$

In the non-ordinary, non-unary cases:

$$\begin{aligned}
B &= AB_0 x^{\delta_r} + CB_1 + DB_0 \qquad \text{(concatenation)}\\
B_0 &= A + C + D + AB_0 x^{\delta_r} + CB_1 x^{\delta_r} + DB_0 x^{\delta_r}\\
B_1 &= A + D + AB_0 x^{\delta_r} + DB_0 x^{\delta_r}
\end{aligned}$$

References

1. N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pp. 118–161. North Holland, Amsterdam, 1963.
2. M. Domaratzki, D. Kisman, and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *J. Automata, Languages, and Combinatorics* **7** (2002), 469–486.
3. K. Ellul, B. Krawetz, J. Shallit, and M.-w. Wang. Regular expressions: new results and open problems. To appear, *J. Autom. Lang. Combin.*, 2003.
4. I. P. Goulden and D. M. Jackson. *Combinatorial Enumeration.* Wiley, 1983.
5. D. A. Klarner and P. Woodworth. Asymptotics for coefficients of algebraic functions. *Aequationes Math.* **23** (1981), 236–241.
6. J. C. Martin. *Introduction to Languages and the Theory of Computation.* McGraw-Hill, 3rd edition, 2003.
7. D. Raymond and D. Wood. *Grail*: a C++ library for automata and expressions. *J. Symbolic Comput.* **17** (1994), 341–350.
8. W. Rudin. *Real and Complex Analysis.* McGraw-Hill, 1966.