T U M

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

Language Operations with Regular Expressions of Polynomial Size

Hermann Gruber Markus Holzer

TUM-I0814
Mai 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-5-I0814-0/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©2008

Druck: Institut für Informatik der
Technischen Universität München

Language Operations with Regular Expressions of Polynomial Size

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

This work deals with questions regarding to what extent regularity-preserving language operations affect the descriptional complexity of regular expressions. Some language operations are identified which are feasible for regular expressions in the sense that the result of the operation can be represented as a regular expression of size polynomial in that of the operands. We prove that taking language quotients, in particular the prefix and suffix closures, of a regular set can incur at most a quadratic blow-up on the required expression size. The circular shift operation can cause only a cubic increase in size. For the latter operation, at least an almost quadratic blow-up can be necessary in the worst case.

1 Introduction

In the last 20 years, a large body of research on the descriptional complexity of finite automata has been developed. To the authors' knowledge, the first systematic attempt to start a parallel development for the descriptional complexity of regular expressions was presented by Ellul et al. [8] at the workshop on "Descriptional Complexity of Formal Systems" (DCFS), in 2002. In particular, they raised the question of determining how basic language operations such as complementation and intersection affect the required regular expression size. For the intersection and shuffle operations, exponential lower bounds are known, and complementation can even incur a doubly-exponential blow-up [9, 10]. In [10] it was shown that the star height of a regular language is at most logarithmic in the minimum regular expression size, and lower bounds are proved by finding families of languages for which the respective language operations incur a dramatic increase in star height. In contrast, it is well known that taking language quotients does not increase the star height [7]. This and similar language operations appear to be a natural testing ground for deepening our understanding of the descriptional complexity of regular expressions: Either one has to find some new lower bound techniques, or one has to find a non-trivial implementation of these operations on regular expressions, or both—a straightforward procedure would be to convert the expression into a finite automaton, implement the operation on a finite automaton, and convert back to a regular expression using state elimination. Yet that last step can incur an exponential blow-up in general, even over binary alphabets [10].

Here, we give polynomial upper bounds for the required expression size resulting from taking language quotients and circular shift. The descriptional complexity of these operations were already studied for various computational models, see [1, 2, 18] for the circular shift on various types of automata and grammars, and [13, 16] for language quotients of deterministic finite automata—the latter two references consider deterministic finite automata with multiple start states, but the results easily translate to state complexity results for (left) quotients.

The basic idea is to implement the operation for the special case of linear expressions [3], called single-occurrence regular expressions in [9]. These are expressions in which every alphabetic symbol occurs exactly once, which makes it easier to deal with as they can describe only local languages. To cover the general case, we study the interplay of the operations with length-preserving homomorphisms.

2 Basic definitions

We recall some basic notions in formal language theory—for a thorough treatment, the reader might want to consult a textbook such as [14]. In particular, let Σ be a finite alphabet and Σ^* the set of all words over the alphabet Σ, including the empty word λ. A *(formal) language* over the alphabet Σ is a subset of Σ^*.

Apart from the regular operations on languages, namely (finite) union, catenation, and star, we briefly recall the following operations on languages: The *reversal* of a language L, denoted by L^R, consists of all words which, when read backwards yield a word in L. The *(left) derivative* of a language L with respect to a word w, written as $w^{-1}L$, is defined as $\{\, x \mid wx \in L \,\}$, the *(left) quotient* of L with respect to a set of words W, denoted by $W^{-1}L$, is defined as $\bigcup_{w\in W} w^{-1}L$. The special case $W = \Sigma^*$ is known as the *suffix closure* of L and denoted by $\mathit{suf}(L)$. We can perform similar operations when reading words from right to left: The *right derivative* of a language L with respect to a word w is defined as $\{\, v \mid vw \in L \,\}$. This operation can be expressed using derivatives and reversal as $((w^R)^{-1}L^R)^R$; right quotients and the prefix closure $\mathit{pre}(L)$ are defined in an analogous manner. The circular (or cyclic) shift of a language, denoted by $\circlearrowleft(L)$, is given by $\{\, xw \mid wx \in L \,\}$.

Let Σ be an alphabet. The regular expressions over Σ are defined recursively in the usual way:[1] $\emptyset$, λ, and every letter a with $a \in \Sigma$ is a regular expression; and when s and t are regular expressions, then $(s+t)$, $(s\cdot t)$, and $(s)^*$ are also regular expressions. The language denoted by a regular expression r, denoted by $L(r)$, is defined as follows: $L(\emptyset) = \emptyset$, $L(\lambda) = \{\lambda\}$, $L(a) = \{a\}$, $L(s+t) = L(s) \cup L(t)$, $L(s\cdot t) = L(s) \cdot L(t)$, and $L(s^*) = L(s)^*$. Two regular expressions are called *equivalent* if they denote the same language. For a regular expression r, define $\lambda(r) = \lambda$ if $\lambda \in L(r)$, and $\lambda(r) = \emptyset$ otherwise. Likewise, for a language L, we define $\lambda(L)$ analogously.

[1]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.

The *size* or *alphabetic width* of a regular expression r over the alphabet Σ, denoted by $\mathrm{alph}(r)$, is defined as the total number of occurrences of letters of Σ in r. For a regular language L, we define its alphabetic width, $\mathrm{alph}(L)$, as the minimum alphabetic width among all regular expressions describing L. The *star height* of a regular expression r, denoted by $h(r)$ is a structural complexity measure inductively defined by

1. $h(r) = 0$, for $r \in \Sigma \cup \{\emptyset, \lambda\}$,
2. $h(s \cdot t) = h(s + t) = \max(h(s), h(t))$, and
3. $h(r^*) = 1 + h(r)$.

The star height of a regular language L is then defined as the minimum star height among all regular expressions describing L.

Let r be a regular expression. Following [15], we say that r is *reduced* if all of the following conditions hold: If r contains the symbol $\emptyset$, then $r = \emptyset$; the expression r contains no subexpression of the form st or ts, with $L(s) = \{\lambda\}$ and no subexpression of the form $(s^*)^*$; if r contains a subexpression of the form $s+t$ or $t+s$ with $L(s) = \{\lambda\}$, then $\lambda \notin L(t)$; if r contains a subexpression of the form s^*, then $L(s) \neq \{\lambda\}$. Otherwise r is called reducible. The above definition suggests some rewriting rules, such as replacing $s + \emptyset$ with s, and a few more rules, see [15]. By iteratively applying the rules to all subexpressions until none is applicable, we can *reduce* every regular expression to a reduced one.

Clearly, for every regular expression there exists an equivalent reduced regular expression with alphabetic width and star height no larger than the original expression. We will need the following relation between star height and alphabetic width of reduced regular expressions:

Lemma 1. *Let r be a reduced regular expression. Then $h(r) \leq \mathrm{alph}(r)$.*

Proof. We prove the following two statements by simultaneous induction on the total number of occurrences of operators in r: If r is a starred expression, then $h(r) \leq \mathrm{alph}(r)$, otherwise $h(r) \leq \mathrm{alph}(r) - 1$. If r contains no operators at all, then the statement clearly holds. To do the induction step, assume the statement holds for all regular expressions with at most m occurrences of operators. In the cases $r = s + t$ and $r = s \cdot t$, we have $h(r) = \max(h(s), h(t))$, and the statement holds by induction hypothesis. If r has the form $(s)^*$, then s is not a starred expression, since r is reduced. Thus by induction assumption, $h(s) \leq \mathrm{alph}(s) - 1$. Since $\mathrm{alph}(r) = \mathrm{alph}(s)$ and $h(r) = h(s) + 1$, the claimed statement also holds in this case, and the proof is completed. □

3 Linear expressions

Let r be a regular expression over the alphabet Σ. The alphabetic width of r, denoted by $\mathrm{alph}(r)$, is the total number of occurrences of symbols in r. We refer to the ith alphabetic letter in r as the ith *position*. A regular expression r over an alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ is called a *linear expression* if and only if $|\Sigma| = \mathrm{alph}(r)$ and the ith position in r is the symbol a_i. In this case, there is a straightforward bijection between positions and alphabet symbols, and here we shall often denote the used alphabet by P_r.

For two alphabets Σ and Γ, a homomorphism $h : \Sigma^* \to \Gamma^*$ is *length-preserving* or also *letter-to-letter* if it maps all symbols from Γ to symbols from Σ. It is easy to see that each regular expression r is the image of a unique linear expression $\overline{r}$ under a length-preserving homomorphism: That homomorphism maps the symbol a_i to the ith position of r. This homomorphism will be denoted by ℓ_r or just ℓ in the case r is understood from the context.

Example 2. *For the regular expression $r = ((ab)^*a)^*$, the corresponding linear expression is $\overline{r} = ((a_1a_2)^*a_3)^*$, and the length-preserving homomorphism which maps $\overline{r}$ to r is given by $\ell_r = \{a_1 \mapsto a, a_2 \mapsto b, a_3 \mapsto a\}$.*

Let Σ be an alphabet. A language $L \subseteq \Sigma^*$ is *local* if

$$L = \lambda(L) \cup (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N \Sigma^*)$$

for some $P, S \subseteq \Sigma$ and $N \subseteq \Sigma^2$. Note that in this definition, we permit the empty word to be a member of a local language. The concept of local languages is related to linear expressions as follows [4]:

Theorem 3. *For every linear expression r, the language $L(r)$ is local.*

We briefly recall the definition of the canonical derivative $d_a(r)$ of a linear expression r with respect to an alphabet symbol a, in the reformulation given in [6, Prop. 6]:

Definition 4. *Let r be a linear expression and let a be a symbol in P_r. Then the canonical derivative $d_a(r)$ is computed recursively by applying the following rules and finally reducing the expression:*

$$\begin{aligned} d_a(a) &= \lambda \\ d_a(s+t) &= \begin{cases} d_a(s) & \text{if } d_a(s) \neq \emptyset \\ d_a(t) & \text{otherwise} \end{cases} \\ d_a(s \cdot t) &= \begin{cases} d_a(s) \cdot t & \text{if } d_a(s) \neq \emptyset \\ d_a(t) & \text{otherwise} \end{cases} \\ d_a(r^*) &= d_a(r) \cdot r^* \end{aligned}$$

And $d_a(r) = \emptyset$ in all cases not covered above.

The study [6] relates the canonical derivatives of a linear expression to the original definition of derivatives for general regular expressions due to Brzozowski [5], and to the continuations introduced by Berry and Sethi [3]. The results from [6] relevant to our context are summarized in the following characterization:

Theorem 5. *Let r be a linear expression, let a be a symbol in P_r, and u a word over P_r. If the set $(ua)^{-1}L(r)$ is nonempty, then it is described by the canonical derivative $d_a(r)$.*

Thus for a reduced linear expression r, the canonical derivative $d_a(r)$ describes the language quotient $(P_r^* a)^{-1}L(r)$.

Example 6. *Consider again the linear expression $\overline{r} = ((ab)^*c)^*$ from Example 2; we now use the alphabet $\{a, b, c\}$ instead of $\{a_1, a_2, a_3\}$ to increase readability. Then*

$$\begin{aligned} d_a(\overline{r}) &= d_a((ab)^*c) \cdot ((ab)^*c)^* = d_a((ab)^*) \cdot c \cdot ((ab)^*c)^* \\ &= d_a(ab) \cdot (ab)^* \cdot c \cdot ((ab)^*c)^* = d_a(a) \cdot b \cdot (ab)^* \cdot c \cdot ((ab)^*c)^* \\ &= \lambda \cdot b \cdot (ab)^* \cdot c \cdot ((ab)^*c)^* = b(ab)^*c((ab)^*c)^*. \end{aligned}$$

A similar computation yields $d_b(\overline{r}) = (ab)^*c((ab)^*c)^*$ and $d_c(\overline{r}) = ((ab)^*c)^*$. □

Now we generalize the above notion from symbols $a \in P_r$ to sets of symbols $A \subseteq P_r$ as follows:

Definition 7. *Let r be a reduced linear expression and let A be a set of symbol in P_r. Then the canonical derivative $d_A(r)$ is computed recursively by applying the following rules and finally reducing the expression:*

$$\begin{aligned} d_A(a) &= \lambda && \text{if } a \in A \\ d_A(s+t) &= d_B(s) + d_{A\setminus B}(t) && \text{with } B = \{\, a \in A \mid d_a(s) \neq \emptyset \,\} \\ d_A(s \cdot t) &= d_B(s) \cdot t + d_{A\setminus B}(t) && \text{with } B = \{\, a \in A \mid d_a(s) \neq \emptyset \,\} \\ d_A(s^*) &= d_A(s) \cdot s^* \end{aligned}$$

And $d_A(r) = \emptyset$ in all cases not covered above.

A straightforward induction shows that the definition works as expected:

Lemma 8. *Let r be a linear expression and A a set of symbols in r. Then $L(d_A(r)) = \bigcup_{a\in A} L(d_a(r))$.* □

Example 9. *Continuing Example 6, for the expression $\overline{r} = ((ab)^*c)^*$ and $A = \{b, c\}$, the expression $d_A(\overline{r})$ computes as*

$$\begin{aligned} d_A(\overline{r}) &= d_A((ab)^*c) \cdot ((ab)^*c)^* = (d_b((ab)^*) \cdot c + d_c(c)) \cdot ((ab)^*c)^* \\ &= (d_b(ab) \cdot (ab)^* \cdot c + d_c(c)) \cdot ((ab)^*c)^* \\ &= (d_b(b) \cdot (ab)^* \cdot c + d_c(c)) \cdot ((ab)^*c)^* = (\lambda \cdot (ab)^* \cdot c + \lambda) \cdot ((ab)^*c)^* \\ &= ((ab)^*c + \lambda) \cdot ((ab)^*c)^*. \end{aligned}$$

For the last line of the above computation, we applied two different rules for reducing the expression. Note that this expression is indeed shorter than the expression $d_b(\overline{r}) + d_c(\overline{r})$ we computed in Example 6, although one might first be tempted to expect the contrary.

A similar computation yields the equalities $d_{\{a,b\}}(\overline{r}) = ((b + \lambda)(ab)^*c) \cdot ((ab)^*c)^*$, $d_{\{a,c\}}(\overline{r}) = (b(ab)^*c + \lambda) \cdot ((ab)^*c)^*$, and finally $d_{\{a,b,c\}}(\overline{r}) = ((b + \lambda)(ab)^*c + \lambda) \cdot ((ab)^*c)^*$. □

Next, we estimate the size of the expressions $d_A(r)$.

Lemma 10. *Let r be a reduced linear expression of alphabetic width $n \geq 1$ and star height h, and let A be a subset of P_r. Then the expression $d_A(r)$ has size at most $\frac{n^2-n}{2} + hn = O(n^2)$.*

Proof. First, recall that Lemma 1 proves that the claimed size is in $O(n^2)$. We prove the claim by induction on the depth $d \geq 0$ of the syntax tree of r. In the case $d = 0$, then with $\mathrm{alph}(r) \geq 1$ we must have $r = a$ for some $a \in P_r$, and the claim clearly holds. To do the induction step, we consider three cases.

If r is of the form $s+t$, then $d_A(r)$ is the expression obtained from reducing $d_B(s)+d_{A\setminus B}(t)$. Let $\mathrm{alph}(s) = k$ and $\mathrm{alph}(t) = n - k$, for some $k \geq 0$. By induction hypothesis we obtain that

$$\mathrm{alph}(d_A(r)) \leq \frac{k^2 - k}{2} + hk + \frac{(n-k)^2 - (n-k)}{2} + h(n-k).$$

By rearranging terms, we get

$$\frac{k^2 - k + (n-k)^2 - (n-k)}{2} = \frac{n^2 - n}{2} + k(k-n) \leq \frac{n^2-n}{2},$$

and thus $\mathrm{alph}(d_A(r))$ is bounded above by $\frac{n^2-n}{2} + hn$ in this case.

If r is of the form $s \cdot t$, then $d_A(r)$ is the expression obtained from reducing the expression $d_B(s)\cdot t + d_{A\setminus B}(t)$. Since r is reduced, both s and t have alphabetic width at least 1. Letting $k \geq 1$ denote the alphabetic width of s and $n - k$ the alphabetic width of t, we obtain by induction hypothesis that

$$\mathrm{alph}(d_A(r)) \leq \frac{k^2 - k}{2} + hk + \frac{(n-k)^2 - (n-k)}{2} + h(n-k) + n - k.$$

By a similar computation as for the previous case, the right hand side in the above inequality is still bounded above by $\frac{n^2-n}{2} + hn$.

Finally, if r is of the form s^*, then the depth of s is smaller than that of r, and by induction assumption $\mathrm{alph}(d_A(r)) \leq \mathrm{alph}(d_A(s)) + n \leq \frac{n^2-n}{2} + (h-1)n + n$. This covers all possible cases, and the proof is completed. $\square$

We remark that our notion of $d_A(r)$ differs from the one given in [6] in that our definition yields expressions of size $O(n^2)$, while defining d_A as $\sum_{a\in A} d_a(r)$ would be much more redundant, recall Example 9. It should be said that the actual size of these expressions is immaterial in the context of [6], but is important in the present paper.

Now we take a closer look at local languages. The following lemma is easy to see from the definition of local languages:

Lemma 11. *If $L \subseteq \Sigma^*$ is a local language, then for each $a \in \Sigma$ holds:*

$$u_1 \cdot a \cdot v_1 \in L \textit{ and } u_2 \cdot a \cdot v_2 \in L \textit{ implies } u_1 \cdot a \cdot v_2 \in L. \qquad \square$$

For a local language L and an alphabet symbol a, we may thus define $d_a(L)$ as the language quotient $d_a(L) = (\Sigma^* a)^{-1}L$. Likewise, for a set of symbols A define $d_A(L) = (\Sigma^* A)^{-1}L$. This operator overloading is perfectly consistent with the use of the notation $d_a(r)$ to denote the canonical derivative of a linear expression r with respect to an alphabet symbol a: By Theorem 5, we have $L(d_a(r)) = d_a(L(r))$.

The above characterization allows us to provide a neat formula for left quotients of local languages:

Lemma 12. *Let $L \subseteq \Sigma^*$ be a local language and let $W \subseteq \Sigma^*$ an arbitrary language. Define $A = \{\, a \in \Sigma \mid W \cap pre(L) \cap \Sigma^* a \neq \emptyset \,\}$. Then*

$$W^{-1}L = \lambda(W) \cdot L \cup \bigcup_{a \in A} d_a(L) = \lambda(W) \cdot L \cup d_A(L).$$

Proof. First of all, it follows from the definition of language quotients that $W^{-1}L = (W \cap pre(L))^{-1}L$. Second, if we decompose the set W as $\lambda(W) \cup \bigcup_{a\in\Sigma}(W \cap \Sigma^* a)$, we obtain

$$\begin{aligned} W^{-1}L &= \lambda(W)^{-1} \cdot L \cup \bigcup_{a\in\Sigma}(W \cap \Sigma^* a \cap pre(L))^{-1}L \\ &= \lambda(W) \cdot L \cup \bigcup_{a\in A}(W \cap \Sigma^* a \cap pre(L))^{-1}L, \end{aligned} \tag{1}$$

since $\bigcup_{a\in\Sigma\setminus A}(W \cap \Sigma^* a \cap pre(L))^{-1}L = (\emptyset)^{-1}L = \emptyset$. Finally, for $a \in A$, let ua be any word in $W \cap \Sigma^* a \cap pre(L)$. Then from Lemma 11 one can readily deduce that $(ua)^{-1}L = (\Sigma^* a \cap pre(L))^{-1}L = (\Sigma^* a)^{-1}L = d_a(L)$. Thus $(W \cap \Sigma^* a \cap pre(L))^{-1}L = d_a(L)$. By putting this into Equation (1), the result follows. □

Example 13. *For illustration, we compute the left quotient of the language $L = L(\overline{r})$ denoted by the linear expression $\overline{r} = ((ab)^*c)^*$ with respect to the set W denoted by the regular expression $(ab)^* + (abc)^*$. To this end, we identify the set $A = \{\, d \in \{a,b,c\} \mid W \cap \Sigma^* d \cap pre(L) \neq \emptyset \,\}$. No word in W ends with a, but the words ab and abc are both in W and prefixes of words in L, thus $A = \{b,c\}$ in our case. In general, the set A can be effectively computed provided W is represented in a machine model that effectively allows intersection with regular sets and has a decidable emptiness problem. This is the case, e.g., for the finite automaton and pushdown automaton models, see [14].*

Now by Lemma 12 holds $W^{-1}L = \lambda(W) \cdot L \cup d_{\{b,c\}}(L)$. Here we have $\lambda(W) \cdot L = L$, and a regular expression denoting $d_{\{b,c\}}(L)$ is given in the previous example (Example 9). Thus, a regular expression denoting the quotient $W^{-1}L$ is given by

$$\overline{r} + d_{\{b,c\}}(\overline{r}) = ((ab)^*c)^* + ((ab)^*c + \lambda) \cdot ((ab)^*c)^*.$$

Observe, that in case $A = \{a,c\}$ the above computation would have resulted in $\overline{r} + d_{\{a,c\}}(\overline{r}) = ((ab)^*c)^* + (b(ab)^*c + \lambda) \cdot ((ab)^*c)^*$, where $d_{\{a,c\}}(\overline{r})$ is computed in Example 6. □

Also for the circular shift of local languages, we obtain a nice characterization:

Lemma 14. *Let $L \subseteq \Sigma^*$ be a local language. Then for the circular shift $\circlearrowleft(L)$ holds*

$$\circlearrowleft(L) = \lambda(L) \cup \bigcup_{a\in\Sigma} a \cdot d_a(L) \cdot (d_a(L^R))^R.$$

Proof. The circular shift of any language L can by definition be written as $\circlearrowleft(L) = \lambda(L) \cup \bigcup_{a\in\Sigma} L(a)$, with

$$L(a) = \{\, awv \mid v, w \in \Sigma^*, vaw \in L \,\}.$$

In particular, if L is local, Lemma 11 tells us that there cannot be any dependencies between the subwords v and w in the definition of $L(a)$. Thus $L(a)$ can be rewritten as

$$L(a) = \bigcup_{\{\, v\in\Sigma^* \mid va\cdot d_a(L)\subseteq L\,\}} a\cdot d_a(L)\cdot v,$$

and since local languages are readily seen to be closed under reversal, redoing the same trick for the reversed language yields $L(a) = a \cdot d_a(L) \cdot d_a(L^R)^R$, as desired. □

Example 15. *We also compute the circular shift of the language $L = L(\overline{r})$ in our running example. By Lemma 14, we can write $\circlearrowleft(L)$ as $\lambda(L) \cup \bigcup_{a\in\Sigma} a \cdot d_a(L) \cdot (d_a(L^R))^R$. The sets $d_a(L)$, $d_b(L)$ and $d_c(L)$ are denoted by the expressions $d_a(\overline{r}) = b(ab)^*c((ab)^*c)^*$, $d_b(\overline{r}) = (ab)^*c((ab)^*c)^*$, and $d_c(\overline{r}) = ((ab)^*c)^*$, respectively, which were computed in Example 6. In a similar manner, we obtain for $\overline{r}^R = (c(ba)^*)^*$ the canonical derivatives $d_a(\overline{r}^R) = (ba)^*(c(ba)^*)^*$, $d_b(\overline{r}^R) = a(ba)^*(c(ba)^*)^*$, and $d_c(\overline{r}^R) = (ba)^*(c(ba)^*)^*$. We have $\lambda(L) = \lambda$, and straightforward rules for implementing the reversal of regular expressions yield $((ba)^*(c(ba)^*)^*)^R = ((ab)^*c)^*(ab)^*$ as well as $(a(ba)^*(c(ba)^*)^*)^R = ((ab)^*c)^*(ab)^*a$. Thus a regular expression denoting $\circlearrowleft(L)$ is given by*

$$\begin{aligned}\lambda + a\cdot b(ab)^*c((ab)^*c)^* \cdot ((ab)^*c)^*(ab)^* \\ + b\cdot (ab)^*c((ab)^*c)^* \cdot ((ab)^*c)^*(ab)^*a + c\cdot ((ab)^*c)^* \cdot ((ab)^*c)^*(ab)^*.\end{aligned}$$

Finally, we note that in an actual implementation the computational overhead for the two reversal operations carried out here could be saved by defining the concept of "canonical right derivatives" using rules analogous to those for computing canonical (left) derivatives in Definition 4. □

These characterizations immediately lend themselves to an implementation of quotient and circular shift operations on linear expressions via canonical derivatives. Using Lemma 10, we can estimate the resulting expression size as follows:

Theorem 16. *Let r be a linear expression of size n, and let $L = L(r)$. Then for set of words $W \subseteq P_r^*$, there is a regular expression of size $O(n^2)$ denoting $W^{-1}L$, and a regular expression of size $O(n^3)$ denoting the circular shift $\circlearrowleft(L)$.* □

4 The General Case

The above results allow us to compute from a given linear expression relatively small regular expressions denoting a language quotient or the circular shift of the denoted language. In this section, we investigate the interaction of (length-preserving) homomorphisms with the language operations under consideration to transfer the obtained results to the general case. The easier case is a language operation that commutes with length-preserving homomorphisms. This is the case for the circular shift:

Lemma 17. *Let ℓ be a length-preserving homomorphism, and let $L \subseteq \Sigma^*$ be a language. Then $\circlearrowleft(\ell(L)) = \ell(\circlearrowleft(L))$.*

Proof. Since both the homomorphism and circular shift operation commute with taking finite and infinite unions, it suffices to show the claim for the case L contains a single word $w = a_1a_2\dots a_k$. In the case $k \leq 1$, we have $L = \circlearrowleft(L)$, and the claim is trivially true. So assume $k \geq 2$. Then

$$\begin{aligned}\ell(\circlearrowleft(\{w\})) &= \{\ell(w)\} \cup \ell\left(\{\, a_j \dots a_k a_1 a_2 \cdots a_{j-1} \mid 2 \leq j \leq k \,\}\right) \\ &= \{\ell(w)\} \cup \{\, \ell(a_j)\dots\ell(a_k)\ell(a_1)\ell(a_2)\dots\ell(a_{j-1}) \mid 2 \leq j \leq k \,\}.\end{aligned}$$

Now let $x = b_1b_2\dots b_k$, with $b_i = \ell(a_i)$. Then

$$\begin{aligned}\circlearrowleft(\{\ell(w)\}) &= \{\ell(w)\} \cup \{\, b_j \dots b_k b_1 b_2 \dots b_{j-1} \mid 2 \leq j \leq k \,\} \\ &= \{\ell(w)\} \cup \{\, \ell(a_j)\dots\ell(a_k)\ell(a_1)\ell(a_2)\dots\ell(a_{j-1}) \mid 2 \leq j \leq k \,\},\end{aligned}$$

thus proving the desired equality. □

The following lemma shows how homomorphisms interact with taking left derivatives.

Lemma 18. *Let $L \subseteq \Sigma^*$ be a regular language, let $\ell : \Sigma^* \to \Gamma^*$ be a length-preserving homomorphism, and let $w \in \Gamma^*$. Then $w^{-1}\ell(L) = \bigcup_{x\in\ell^{-1}(w)} \ell(x^{-1}L)$.*

Proof. Let $A = (Q, \Sigma, \delta, Q_0, F)$ be a nondeterministic finite automaton (possibly with multiple start states) accepting L, in the standard notation of [14]. We obtain a nondeterministic finite automaton B accepting $\ell(L)$ by a standard construction: Let $B = (Q, \Gamma, \delta', Q_0, F)$ with $\delta'(q, a) = \bigcup_{b\in\ell^{-1}(a)} \delta(q, b)$, for every $q \in Q$ and every $a \in \Gamma$. For an automaton C accepting $w^{-1}\ell(L)$, we perform the standard quotient construction: Let $C = (Q, \Gamma, \delta', Q'_0, F)$, with $Q'_0 = \bigcup_{q_0\in Q_0} \delta'(q_0, w)$.

For a finite automaton accepting the language $\bigcup_{x\in\ell^{-1}(w)} x^{-1}L$, let $D = (Q, \Sigma, \delta, Q'_0, F)$ with the same start states as above. Note that for the set Q'_0 holds

$$Q'_0 = \bigcup_{q_0\in Q_0} \delta'(q_0, w) = \bigcup_{x\in\ell^{-1}(w)} \bigcup_{q_0\in Q_0} \delta(q_0, x),$$

so this automaton indeed accepts the quotient $\bigcup_{x\in\ell^{-1}(w)} x^{-1}L$. To get an automaton accepting the image under ℓ of this language, we replace in D, similar to above, the transition function δ with δ' and change the input alphabet to Γ. But then we end up with the automaton $(Q, \Gamma, \delta', Q'_0, F)$, which is identical to the automaton C. Thus $w^{-1}\ell(L) = \bigcup_{x\in\ell^{-1}(w)} \ell(x^{-1}L)$ as desired. □

Now we are ready to state the main result of this paper:

Theorem 19. *Let r be a regular expression of size n denoting the language $L \subseteq \Sigma^*$, and let $W \subseteq \Sigma^*$. Then there is a regular expression of size $O(n^2)$ denoting $W^{-1}L$ and a regular expression of size $O(n^3)$ denoting $\circlearrowleft(L)$.*

Proof. Let $\overline{r}$ be the linear expression for r, and $\ell = \ell_r$ be the homomorphism which maps $\overline{r}$ to r. Since ℓ is length preserving, every word $w \in \Sigma^*$ is in $\ell(P_{\overline{r}})^*$, and thus, by Lemma 18, we have

$$w^{-1}\ell(L(\overline{r})) = \bigcup_{x\in\ell^{-1}(w)} \ell(x^{-1}L(\overline{r})).$$

This readily generalizes to sets of words, and we obtain

$$W^{-1}\ell(L(\overline{r})) = \bigcup_{w\in W} \bigcup_{x\in\ell^{-1}(w)} \ell(x^{-1}L(\overline{r})) = \ell(\bigcup_{x\in\ell^{-1}(W)} x^{-1}L(\overline{r})) = \ell\left((\ell^{-1}(W))^{-1}L(\overline{r})\right)$$

The last one of the above expressions is the image under ℓ of a quotient of $\overline{r}$. By Theorem 16, the latter language can be described by a regular expression of size $O(n^2)$, and applying the map ℓ does not increase the alphabetic width. This shows that $\mathrm{alph}(W^{-1}L) = O(n^2)$. For the circular shift, recall from Theorem 16 that $\circlearrowleft(L(\overline{r}))$ has alphabetic width in $O(n^3)$. By Lemma 17, $\circlearrowleft(L(\ell(\overline{r}))) = \ell(\circlearrowleft(L(\overline{r})))$, and, as noted before, applying a length-preserving homomorphism does not increase the alphabetic width. $\square$

Example 20. *Let's come back to the regular expression $r = ((ab)^*a)$ from Example 2. Recall that the length-preserving homomorphism which maps the corresponding linear expression $\overline{r}$ to r reads as $\ell_r = \{a_1 \mapsto a, a_2 \mapsto b, a_3 \mapsto a\}$. In all previous examples the alphabet $\{a,b,c\}$ was used instead of $\{a_1,a_2,a_3\}$ to increase readability—here we stay with this convention, although there may be danger of confusion, since for example an expression for $\ell^{-1}((ab)^*)$ is in fact over the alphabet $\{a_1,a_2,a_3\}$ and not over $\{a,b,c\}$.*

Thus, the left-quotient of $L(r)$ with respect to the set W denoted by the expression $(aba)^*$ can be similarly computed as in Example 13. To this end we determine the linear expressions for r and the expression for $\ell^{-1}((aba)^*)$, which read as $\overline{r} = ((ab)^*c)^*$ and $((a+c)b(a+c))^*$, respectively. Then the main result of this section shows that the language $W^{-1}L(r)$ is given by $\ell((\ell^{-1}(W))^{-1}L(\overline{r}))$. First we determine $A = \{\, d \in \{a,b,c\} \mid \ell^{-1}(W) \cap \Sigma^* d \cap pre(L(\overline{r})) \neq \emptyset \,\}$, where $\Sigma = \{a,b,c\}$. It is easily seen that $A = \{a,c\}$. Then an analogous computation as in Example 13 shows that

$$\begin{aligned}(\ell^{-1}(W))^{-1}L(\overline{r}) &= \overline{r} + d_{\{a,c\}}(\overline{r})\\ &= ((ab)^*c)^* + (b(ab)^*c + \lambda)\cdot((ab)^*c)^*,\end{aligned}$$

where $d_{\{a,c\}}(\overline{r})$ is given in Example 9. Thus applying the length-preserving homomorphism ℓ gives

$$W^{-1}L(r) = ((ab)^*a) + (b(ab)^*a + \lambda)((ab)^*a)^*.$$

What concerns the circular shift of our running example? Finally, we deduce from Example 15 that the circular shift of $L(r)$ is simply

$$\begin{aligned}\circlearrowleft(L(r)) = \lambda + a\cdot b(ab)^*a((ab)^*a)^*\cdot((ab)^*a)^*(ab)^*&\\ + b\cdot(ab)^*a((ab)^*a)^*\cdot((ab)^*a)^*(ab)^*a + a\cdot((ab)^*a)^*\cdot((ab)^*a)^*(ab)^*&,\end{aligned}$$

again by applying the length-preserving homomorphism ℓ to the result presented there. $\square$

Currently, we do not know whether these upper bounds have the right order of magnitude. At least, we found an almost quadratic lower bound on the increase of alphabetic width for the circular shift operation:

Theorem 21. *There exist infinitely many regular languages L_m over a binary alphabet such that L_m admits a regular expression of alphabetic width m, but every regular expression describing $\circlearrowleft(L_m)$ has alphabetic width at least $\Omega\left(\frac{m^2}{\log^2 m}\right)$.*

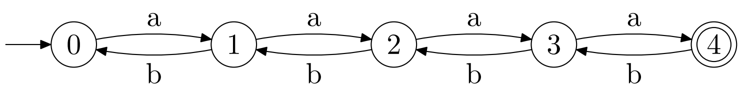

Figure 1: A finite automaton accepting L_4

Proof. Our witness language is the language L_m for which in [18] an $\Omega(m^2)$ lower bound on the number of states needed by a nondeterministic finite automaton in order to accept $\circlearrowleft(L_m)$ was proved. The language L_m contains all words w over the alphabet $\{a, b\}$ such that $|w|_a - |w|_b = m-1$ and moreover for every prefix x of w holds $0 \leq |x|_a - |x|_b \leq m-1$. A finite automaton A_4 accepting L_4 is depicted in Figure 1. By a straightforward generalization of the pattern of the depicted automaton, we obtain an $m+1$-state automaton A_m, whose start state is the state 0, and the only final state is state m. For simplicity, assume for now $m = 2^k$ for some integer $k \geq 0$. It is easily observed that each accepting computation path has to go through the middle state $\frac{m}{2}$ at least once. Each time such a path reaches this state, the path can either continue by reading the letter a, going to the right and, by using only states with numbers higher than $\frac{m}{2}$, eventually returns to the middle state, or accept. The other possibility is that the path continues from the middle state by reading the letter b, the computation continues by using only states with numbers less than or equal to $\frac{m}{2}$, and eventually return to the middle state. This gives rise to the following recursive definition:

$$r_0 = \lambda, \quad r_1 = a(ba)^*, \quad \text{and} \quad r_m = r_{m/2} \cdot (s_{m/2} + t_{m/2})^* \cdot r_{m/2}, \quad \text{for } m \geq 2.$$

Moreover we define

$$s_0 = t_0 = \lambda, \quad s_m = (a \cdot s_{m-1} \cdot b)^*, \quad \text{and} \quad t_m = (b \cdot t_{m-1} \cdot a)^*, \quad \text{for } m \geq 1.$$

Here the regular expression $s_{m/2}$ ($t_{m/2}$, respectively) describe the computations that start and end in the middle state, and using only states numbered higher (lower, respectively) than or equal to $m/2$. Two sample computations are depicted in Figure 2—in the computation drawn as a solid line the word read between the points marked with A and B belongs to $L(r_{m/2})$, that read between B and C to $L(s_{m/2})$, that between C and D to $L(t_{m/2})$, and finally the word read between D and E again to $L(r_{m/2})$. It is easy to see that any computation can be decomposed according to the above given recurrence.

It is not hard to see that this expression has alphabetic width $O(n \log n)$, the above being a typical divide-and conquer recurrence. Since every regular expression of size m can be transformed into a nondeterministic finite automaton having at most $m+1$ states, every regular expression describing $\circlearrowleft(L_n)$ needs to have size at least $\Omega(n^2)$. Letting $m = \text{alph}(r_n) = \Theta(n \log n)$, the lower bound reads as $\Omega(n^2) = \Omega\left(\frac{m^2}{\log^2 m}\right)$. $\square$

For language quotients, in particular for prefix and suffix closure, we can currently provide only a linear lower bound: E.g., for the unary language $\{a^n\}$, we have $(a^*)^{-1}a^n = a^{\leq n}$. Clearly, language $a^{\leq n}$ has alphabetic width at least n, since this is the length of the longest word in the language. At least, observe that the languages $L(r_n)$ and $L(s_n)$ defined in the above proof are mutual derivatives. While s_n is a regular expression of size linear in n, we were unable to find a regular expression for $L(r_n)$ of size $o(n \log n)$. It remains open whether such an expression could possibly exist.

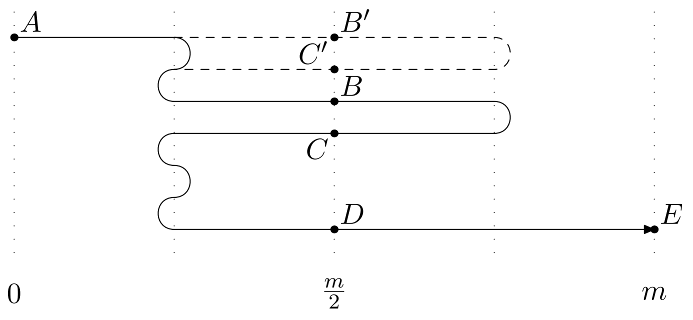

Figure 2: Two sample computations (solid and dashed line) on the m-state finite automaton A_m. Here B (B', respectively) is the point where the computation drawn as a solid (dashed, respectively) line reaches the middle state the first time, and D is that point of the computation, where the middle state is seen the last time during the computation.

5 Conclusion

In this paper, we identified some regularity-preserving language operations whose effect on required regular expression size is not too drastic, i.e., which can incur at most a polynomial blow-up. Among these are all operations which are special cases of language quotients, e.g., the prefix or suffix closure of a set of words, and the circular shift. The naive way to implement such an operation would involve a translation into finite automata and back. However, the conversion into the back direction likely causes an undesirable blow-up in expression size. In contrast, the algorithms presented here are entirely based on rewriting expressions and thus avoid these difficulties altogether. There are two ingredients in such an approach: First, the language operation under consideration needs to admit an efficient solution for linear expressions. Second, it needs to be somehow well-behaved with respect to length preserving homomorphisms.

One task for further research is to find other regularity preserving operations for which this or similar approaches might work. For instance, for the language of scattered substrings (superstrings, respectively) of the language described by a regular expression over Σ, we simply replace every position a with a subexpression $\lambda + a$ (with a subexpression describing $\Sigma^* a \Sigma^*$, respectively) to obtain a regular expression denoting that language. Both operations can be thus performed with only linear increase in expression size provided Σ is fixed. Issues on the state complexity of these operations were studied recently in [11] and [17].

Another, probably difficult, challenge is to try to tighten the bounds given here. Quite a few lower bound techniques for regular expression size, apart from those based on the number of states required by a nondeterministic finite automaton, have been developed recently [9, 10, 12]. Apparently none of them can be used to infer something nontrivial about language quotients.

References

[1] P. R. J. Asveld. Generating all circular shifts by context-free grammars in Chomsky normal form. *Journal of Automata, Languages and Combinatorics*, 11(2):147–159, 2006.

[2] P. R. J. Asveld. Generating all circular shifts by context-free grammars in Greibach normal form. *International Journal of Foundations of Computer Science*, 18(6):1139–1149, 2007.

[3] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.

[4] J. Berstel and J. E. Pin. Local languages and the Berry-Sethi algorithm. *Theoretical Computer Science*, 155(2):439–446, 1996.

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

[6] J.-M. Champarnaud and D. Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *Theoretical Computer Science*, 289(1):137–163, 2002.

[7] R. S. Cohen and J. A. Brzozowski. General properties of star height of regular events. *Journal of Computer and System Sciences*, 4(3):260–280, 1970.

[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. In L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walkuwiewicz, editors, *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, Reykjavik, Iceland, July 2008. Springer. Accepted for publication.

[11] H. Gruber, M. Holzer, and M. Kutrib. More on the size of Higman-Haines sets: Effective c onstructions. In J. O. Durand-Lose and M. Margenstern, editors, *Proceedings of the 5th International Conference Machi nes, Computations, and Universality*, volume 4664 of *LNCS*, pages 193–204, Orléans, France, September 2007. Springer.

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

[13] M. Holzer, K. Salomaa, and S. Yu. On the state complexity of k-entry deterministic finite automata. *Journal of Automata, Languages and Combinatorics*, 6(4):453–466, 2001.

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

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

[16] M. Kappes. Descriptional complexity of deterministic finite automata with multiple initial states. *Journal of Automata, Languages and Combinatorics*, 5(3):269–278, 2000.

[17] A. Okhotin. On the state complexity of scattered substrings and superstrings. Technical Report TUCS Technical Report No.849, University of Turku - Department of Mathematics and Turku Centre for Computer Science and Academy of Finland, October 2007.

[18] A. Okhotin and G. Jirásková. State complexity of cyclic shift. *RAIRO – Theoretical Informatics and Applications*, 42(2):335–360, 2008.