I F I G
Research
Report

Institut für Informatik

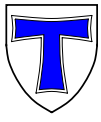
Simplifying Regular Expressions: A Quantitative Perspective

Hermann Gruber Stefan Gulan

IFIG Research Report 0904
August 2009

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

Justus-Liebig-



Universität
Giessen

IFIG Research Report

IFIG Research Report 0904, August 2009

Simplifying Regular Expressions: A Quantitative Perspective

Hermann Gruber[1]

Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany

and

Stefan Gulan[2]

Fachbereich IV—Informatik, Universität Trier
Campus II, D-54296 Trier, Germany

Abstract. In this work, we consider the efficient simplification of regular expressions. We suggest a quantitative comparison of heuristics for simplifying regular expressions. We propose a new normal form for regular expressions, which outperforms previous heuristics while still being computable in linear time. We apply this normal form to determine an exact bound for the relation between the two most common size measures for regular expressions, namely alphabetic width and reverse polish notation length. Then we proceed to show that every regular expression of alphabetic with n can be converted into a nondeterministic finite automaton with ϵ-transitions of size at most $4\frac{2}{5}n + 1$, and that this bound is optimal. This provides an exact resolution of a research problem posed by Ilie and Yu, who had obtained lower and upper bounds of $4n - 1$ and $9n - \frac{1}{2}$, respectively [L. Ilie, S. Yu: Follow automata. *Inform. Comput.* 186, 2003]. For reverse polish notation length as input size measure, an optimal bound was recently determined [S. Gulan, H. Fernau: An optimal construction of finite automata from regular expressions. In: *Proc. FST&TCS*, 2008]. We prove that, under mild restrictions, their construction is also optimal when taking alphabetic width as input size measure.

Categories and Subject Descriptors: F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Relations among complexity measures*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems—*Parsing*;

Additional Key Words and Phrases: regular expressions, finite automata, star normal form, efficient algorithm

[1]E-mail: hermann.gruber@informatik.uni-giessen.de
[2]E-mail: gulan@uni-trier.de

1 Introduction

It is well known that simplifying regular expressions is by no means an easy task, since alone deciding whether a given regular expression is universal, i.e., describes the set of all strings, is **PSPACE**-complete [19]. As witnessed by several recent studies, e.g. [6, 10–13], the descriptional complexity of regular expressions is of great interest, so this problem cannot be simply ignored. At least, a few different heuristics for simplifying regular expressions appear in the literature. Since efficiency matters, these mostly deal with removing only the most obvious redundancies, such as removing Kleene stars over stars or superfluous occurrences of expressions denoting the empty word [5, 9, 16].

In this work, we take a quantitative viewpoint to compare such simplifications: Namely, we compare the total size of a regular expression (disregarding parentheses) to its alphabetic width. The intuition behind this is explained as follows: There are simplifications for regular expressions that are of an ad-hoc nature, e.g. the rule $r + r = r$ cannot simplify $a^* + (a + b)^*$. Also, there are rules that are difficult to apply, e.g. if $L(r) \subseteq L(s)$, then $r + s = s$. But there are also simplifications that do not fall in either category, such as the reduction rules suggested in [17, 5, 9, 16]. In this paper, we suggest a *strong star normal form* of regular expressions, which is a slightly strengthened version of the star normal form defined in [5]. This normal form achieves an optimal ratio when comparing expression size to alphabetic width, and can be computed as efficiently as the original star normal form.

For converting regular expressions into small ε-NFAs, an optimal construction was found recently in [14]. Here, *optimal* means that the algorithm attains the best possible quotient of *expression size* and automaton size. Ilie and Yu [16] proposed the problem of determining the optimal quotient if we replace expression size with *alphabetic width*; they obtained an upper bound of roughly 9. We resolve this open problem by showing that this quotient equals $4\frac{2}{5}$. In fact, we prove that the construction from [14] attains this bound if the input expression is in strong star normal form. We then move on to show that this still holds, under very mild restrictions, also for expressions not in star normal form. Thus our results give the impression that this construction of ε-NFAs from regular expressions is optimal in a very robust sense.

2 Basic Notions

We give a brief account on the syntax and semantics of regular expressions and the languages they denote. Let Σ be an alphabet. The regular expressions over Σ and the languages they denote are defined recursively as follows:

Every symbol a with $a \in \Sigma$ is a regular expression, and if r_1 and r_2 are regular expressions, then $(r_1 + r_2)$, $(r_1 \cdot r_2)$, $(r_1)^?$ and $(r_1)^*$ are also regular expressions. The language denoted by a regular expression r, in symbols $L(r)$, is defined inductively as follows: $L(a) = \{a\}$, $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$, $L(r_1^?) = \{\varepsilon\} \cup L(r_1)$ and $L(r_1^*) = L(r_1)^*$. A language is called regular if it is denoted by some regular expression.

Note that we slightly deviate from the standard that is used in most textbooks: We do not introduce special symbols for the empty set and the empty word, but add special operator for adding the empty word to the language. The disadvantages of our abbreviated syntax are minor—we cannot describe the degenerate languages $\emptyset$ and $\{\varepsilon\}$, while there is a big advantage: The abbreviated syntax disallows *a priori* the construction of many kinds of unnatural and redundant expressions, such as $\varepsilon \cdot r$ or $\emptyset^*$.

There are two commonly used measures for the length of a regular expression: The alphabetic width of a regular expression r, denoted by $\mathrm{alph}(r)$ is defined as the total number of occurrences of alphabetic symbols from Σ. The second measure is the reverse polish notation length. To allow for better comparison with previous works, e.g., [9, 16], we define the (abbreviated) reverse polish notation length of an expression r as $\mathrm{arpn}(r) = |r|_\Sigma + |r|_+ + |r|_\cdot + |r|_* + |r|_?$, and define its unabbreviated rpn-length as $\mathrm{rpn}(r) = \mathrm{arpn}(r) + |r|_?$. This comes from the fact that replacing each subexpression of the form $s^?$ with $s+\varepsilon$ increases the overall length by 1 each time. The alphabetic width of a regular language L is then defined as the minimum alphabetic width among all regular expressions denoting L, and is denoted by $\mathrm{alph}(L)$. The notions $\mathrm{rpn}(L)$ and $\mathrm{arpn}(L)$ are defined in the same vein.

We will also need a few notions from term rewriting; a thorough introduction is given in [2]. Let S be a set, and let $\to$ be a relation on S. Let $\to^*$ denote the transitive closure of $\to$. Two elements b and c from S are called *joinable*, if there exists an element $d \in S$ such that both $b \to^* d$ and $c \to d$. If furthermore d itself has no successors e with $d \to e$, then d is a *(joint) normal form* for b and for c.

The relation $\to$ is *confluent*, if for all $a, b, c \in S$ with $a \to^* b$ and $a \to^* c$, the elements b and c are joinable. It is *locally confluent*, if for all $a, b, c \in S$ with $a \to b$ and $a \to c$, the elements b and c are joinable. The relation is *terminating*, if there is no infinite descending chain $a_1 \to a_2 \to \cdots$.

It is easy to prove that if $\to$ is confluent and terminating, then each element has a unique normal form, see e.g. [2, Thm. 2.1.9]. Indeed for unique normal forms, we only need to establish local confluence instead of confluence: Newman's Lemma states that if a terminating relation is locally confluent, then it is confluent ([18], see also [2, Lem. 2.7.2]).

3 Alphabetic Width Versus Reverse Polish Notation Length

Although the authors originally devised a different simplification algorithm, they noticed their algorithm sharing some features with the *star normal form* proposed by Brueggemann-Klein [5]. Therefore the authors decided to merge the two notions into a single simplification algorithm, and it turned out that the star normal form indeed "almost does it". As in the original definition of star normal form from [5], the definition below is based on the use of two operators on regular expressions.

Definition 1. *The operators $\circ$ and $\bullet$ on regular expressions are inductively defined as follows: For the first operator, let $a^\circ = a$, for $a \in \Sigma$, $(r+s)^\circ = r^\circ + s^\circ$,*

$r^{?\circ} = r^\circ$, $r^{*\circ} = r^\circ$, *and*

$$(rs)^\circ = \begin{cases} rs, & \text{if } \varepsilon \notin L(rs) \\ r^\circ + s^\circ & \text{else} \end{cases}.$$

The second operator is given by: $a^\bullet = a$, *for* $a \in \Sigma$, $(r+s)^\bullet = r^\bullet + s^\bullet$, $(rs)^\bullet = r^\bullet s^\bullet$, $r^{*\bullet} = r^{\bullet\circ *}$, *and*

$$r^{?\bullet} = \begin{cases} r^\bullet & \text{, if } \varepsilon \in L(r) \\ r^{\bullet ?} & \text{otherwise} \end{cases}.$$

The strong star normal form *of a regular expression r is then defined as the expression $r^\bullet$.*

We note that, for instance, the expression $(\emptyset + a)^* + \varepsilon \cdot b + \emptyset \cdot c \cdot (d + \varepsilon + \varepsilon)$ in unabbreviated syntax is in star normal form, so the relative advantage of strong star normal form should be obvious. The difference to star normal form merely consists in using abbreviated syntax and in the addition of a nontrivial rule for computing $r^{?\bullet}$, which in the original definition equals $r^{\bullet ?}$. We note that all the statements [5, Thm. 3.1, Lem. 3.5, 3.6, 3.7] regarding the operators $^\circ$ and $^\bullet$ carry over for the variation defined above.

Observe that the concept of strong star normal form also appears, in less explicit form, in [7]. Those authors had a slightly different but related goal in mind, namely to compare different constructions of ε-free NFAs from regular expressions.

We now compare the reverse polish notation length and alphabetic width of regular expressions in strong star normal form. To this end, for a regular expression r in abbreviated syntax, define $\omega(r) = |r|_? + |r|_*$, that is, the function ω counts the total number of occurrences of unary operators in r. The following statement is immediate from the definition of the operators $\bullet$ and $\circ$, and a similar statement concerning rpn-length is found in [5].

Lemma 2. *Let r be a regular expression. Then $\omega(r^\bullet), \omega(r^\circ) \leq \omega(r)$, and $\omega(r^{*\circ}) \leq \omega(r^*) - 1$.* □

Lemma 3. *Let r be a regular expression, and let $r^\bullet$ denote its star normal form as given by Definition 1. Then $\omega(r^\bullet) \leq \mathrm{alph}(r^\bullet)$, if $\varepsilon \in L(s)$, and $\omega(r^\bullet) \leq \mathrm{alph}(r^\bullet) - 1$ otherwise.*

Proof. We prove the Lemma by lexicographic induction on the pair (n, h), where n is the alphabetic width of $r^\bullet$, and h the height of its parse. The base case of the induction is $(1, 1)$. There we have $r^\bullet \in \Sigma$, and the statement clearly holds.

For the induction step, assume the statement holds for all regular expressions with alphabetic width at most $n-1$ and for all expressions of alphabetic width n and height at most $k-1$. The only interesting cases for the induction step are $r = s^?$ and $r = s^*$. In the first case, we have $r^\bullet = s^\bullet$ if $\varepsilon \in L(s)$. Since the induction hypothesis is applicable for $s^\bullet$, we must have

$$\mathrm{alph}(r^\bullet) = \mathrm{alph}(s^\bullet) \geq \omega(s^\bullet) = \omega(r^\bullet).$$

If $\varepsilon \notin L(s)$, we have $r^\bullet = s^{\bullet?}$, and again the induction hypothesis applies for s. This time, we obtain

$$\operatorname{alph}(r^\bullet) = \operatorname{alph}(s^\bullet) \geq \omega(s^\bullet) + 1 = \omega(r^\bullet).$$

The other interesting case is where the topmost operator in r is the Kleene star. Here we distinguish several cases. The easy cases are $r = s^{?*}$ and $r = s^{**}$: it is easy to see that in these cases $r^\bullet = s^{*\bullet}$ and the claim holds by induction. For the case $r = (s + t)^*$, expansion of the definition gives

$$r^\bullet = (s^{*\bullet\circ} + t^{*\bullet\circ})^*.$$

Since both $s^{*\bullet}$ and $t^{*\bullet}$ must[1] have alphabetic width strictly less than n, and since both describe the empty word, we can apply the induction hypothesis and obtain

$$\operatorname{alph}(r^\bullet) = \operatorname{alph}(s^{*\bullet}) + \operatorname{alph}(t^{*\bullet}) \geq \omega(s^{*\bullet}) + \omega(t^{*\bullet}).$$

Now since $\omega(s^{*\bullet\circ}) \leq \omega(s^{*\bullet}) - 1$, and similar for $t^{*\bullet\circ}$, we can deduce that $\omega(r^\bullet) \leq \operatorname{alph}(r^\bullet) - 2$, and this completes the induction step for this case.

For the case where $r = (st)^*$, we have $r^\bullet = (s^\bullet t^\bullet)^{\circ *}$ and the induction goes through if at least one of s and t does not describe the empty word. If however $\varepsilon \in L(s) \cap L(t)$, then it is easy to prove under this condition that $r^\bullet = (s+t)^{*\bullet}$, a case we already dealt with a few lines above in this proof. □

Theorem 4. *Let L be a regular language. Then* $\operatorname{arpn}(L) \leq 3\operatorname{alph}(L) - 1$ *and* $\operatorname{rpn}(L) \leq 4\operatorname{alph}(L) - 1$.

Proof. Let r be a regular expression, in abbreviated syntax, of minimum alphabetic width denoting L. Then the parse tree of $r^\bullet$ has $\operatorname{alph}(r)$ many leaves. Disregarding unary operators, this is a binary tree with $\operatorname{alph}(r) - 1$ internal vertices that correspond to occurrences of binary operators in r. Since there are furthermore at most $\operatorname{alph}(r)$ many occurrences of unary operators, we have $\operatorname{arpn}(r^\bullet) \leq 3\operatorname{alph}(L) - 1$ and $\operatorname{rpn}(r^\bullet) \leq \operatorname{arpn}(r^\bullet) + \omega(r^\bullet) \leq 4\operatorname{alph}(L) - 1$. □

Thus size and alphabetic width can differ at most by a factor of 4 in unabbreviated syntax. Previous bounds, which were based on other simplification paradigms, by Ilie and Yu [16] and by Ellul et al. [9] only achieved factors of 6 and 7, respectively, in place of 4. For abbreviated syntax, we will later show that the bound of the form $3n - 1$ is best possible. Also note that strong star normal form subsumes all of the previous simplification heuristics from [5, 9, 16].

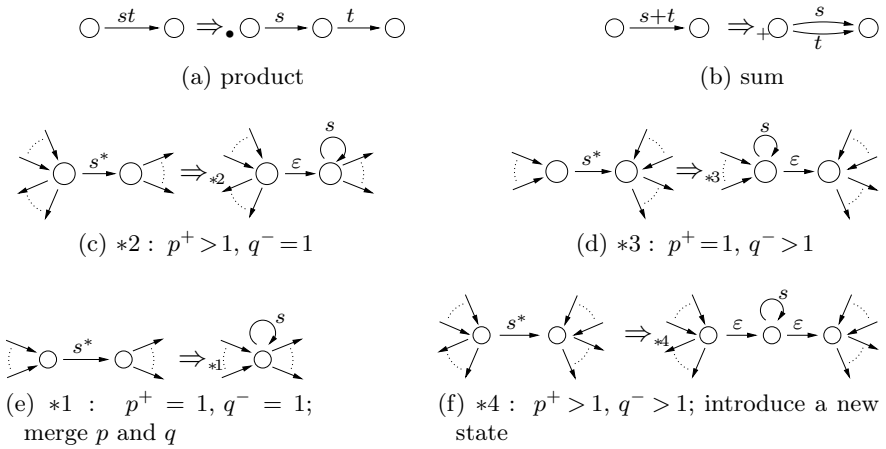
4 Constructing ε-NFAs from Regular Expressions, Revisited

In this section, we show that the construction given by Gulan and Fernau [14] has a desirable feature: Under mild restrictions, it subsumes the conversion of the input expression into strong star normal form. We briefly recapitulate the construction. It is essentially a replacement system on digraphs, arc-labeled

[1] This is just another advantage of using abbreviated syntax.

by regular expressions or occurrences of the symbol ε. Such objects are called *extended finite automata* (EFAs), as they generalize (conventional) finite automata; consult e.g. Wood [22] for a proper introduction. A replacement step within an EFA will also be called *conversion*. Conversions come in two flavors:

- A transition labeled by a regular expression may be replaced wrt. the labels root. These conversions, called *expansions*, are depicted in Fig. 1.
- A substructure defined by ε-transitions may be replaced by a smaller equivalent. These conversions are also called *eliminations*, they are shown in Fig. 2.



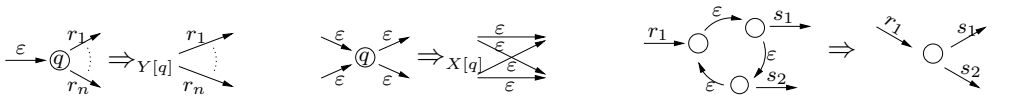
(a) product (b) sum

(c) $*2:\ p^+ > 1,\ q^- = 1$ (d) $*3:\ p^+ = 1,\ q^- > 1$

(e) $*1:\ p^+ = 1,\ q^- = 1$; merge p and q

(f) $*4:\ p^+ > 1,\ q^- > 1$; introduce a new state

Fig. 1: Expanding transitions (p, r, q) for nontrivial r. If $r = s^*$, the out-degree p^+ of p and the in-degree q^- of q need to be considered.



(a) Y-elimination, requires $q^- = 1$

(b) X-elimination, requires $q^- = q^+ = 2$

(c) O-elimination

Fig. 2: Eliminating substructures with ε-labeled transitions. Reverting all transitions in (a), and demanding that $q^+ = 1$ yields a further Y-rule.

Since ε-transitions are allowed in EFAs, we treat $r^?$ implicitly as $r+\varepsilon$. We call the *lhs* of i-expansion or i-elimination an *i-anchor*, and write $E \Rightarrow_i E'$ if E' is derived from replacing an i-anchor in E with its according *rhs*. If the particular type of conversion is irrelevant, we write $E \Rightarrow E'$, an denote a (possibly empty) series of conversions from E to E' with $E \Rightarrow^* E'$. A regular expression r over Σ is identified with the trivial EFA $A_r^0 := (\{q_0, q_f\}, \Sigma, \{(q_0, r, q_f)\}, q_0, q_f)$. On input r, the construction is initialized with A_r^0, which is successively and exhaustively

converted to an ε-NFA, denoted A_r. We slightly restrict the applicability of conversions by two rules:

(R1) As long as any conversion other than $\Rightarrow_X$ is possible, X-elimination must not be applied.
(R2) If two X-anchors share ε-transitions the one from which they are leaving is to be eliminated.

Other than that, conversions may be applied in any order. Note that the restriction (R2) is sound: there can be no cyclic elimination preference amongst X-anchors, since this would imply an ε-cycle, which would be O-eliminated first, due to (R1). The conversion process is split into a sequence of conversions without X-eliminations followed by one with X-eliminations only. This is due to

Proposition 5. *Let $E \Rightarrow_X E'$ be a conversion respecting (R1). Then E and E' contain X-anchors only.*

Proof. Obviously, no complex labels and no cycles, particularly no O-anchors, are introduced upon X-elimination. Assume $E \Rightarrow_{X[q]} E' \Rightarrow_{Y[p]} E''$ is a valid conversion sequence, then p and q are adjacent in E, since the Y-anchor in E' must result from the preceding X-elimination. Let the transition connecting p and q in E be (p, ε, q), then in E', $p^+ = 2$, hence $p^- = 1$. But the in-degree of p is not changed by this X-elimination, so $p^- = 1$ in E, too. Therefore E contains an Y-anchor centered in p, contradicting the assumption that the conversion respects (R1). Since by this assumption, all anchors in E are X-anchors, so are all anchors in E'.

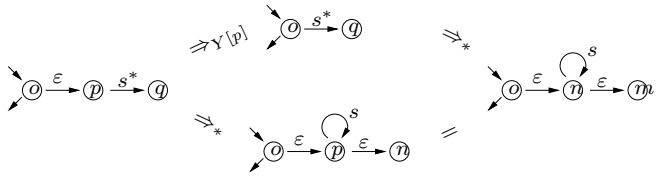
To designate the transition between the two phases, let A_r^k be the first EFA in the sequence $A_r^0 \Rightarrow A_r^1 \Rightarrow \cdots \Rightarrow A_r$, s.t. no conversion other than possibly X-elimination is applicable to A_r^k, we denote this automaton X_r. Of course, if X-elimination does not occur at all upon full conversion, then $X_r = A_r$.

We show that X_r is unique by proving that the replacement system consisting of conversions other than X-elimination is locally confluent on the class of EFAs. To this end, we write $E_1 \cong E_2$, if E_1 and E_2 can be converted to the same EFA, while respecting (R1) and (R2). Since no infinite conversion sequences are possible, uniqueness of X_r follows by applying Newman's Lemma ([18]).

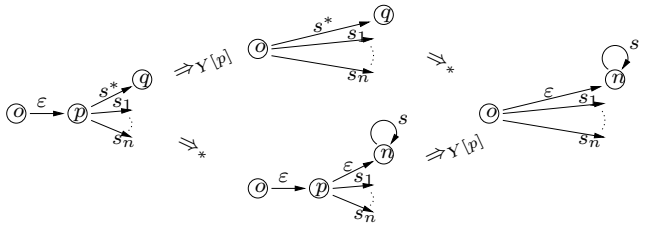
Theorem 6. *The replacement-system consisting of $\Rightarrow_+$, $\Rightarrow_\bullet$, $\Rightarrow_{*i}$, $\Rightarrow_Y$ and $\Rightarrow_O$ is locally confluent on the class of EFAs.*

Proof. We need to show that whenever $E \Rightarrow_i E_1$ and $E \Rightarrow_j E_2$ for $i, j \in \{+, \bullet, *1, *2, *3, *4, Y, O\}$, then $E_1 \cong E_2$. This is trivial if the conversions occur in different regions of E, so assume the i- and j-anchors share at least a state. We assume that at least one of i, j is Y or O, the remaining cases already have been considered in [14, Lem. 6]. We distinguish by the type of i:

– $i = Y[p]$: Let (o, ε, p) be the ε-transition to eliminate and assume $\Rightarrow_j$ is an expansion, i.e., one of the labels r_k as in Fig. 2(a) is a product, a sum or a starred expression. In the case of a sum or product, it is easy to see that the order of $\Rightarrow_i$ and $\Rightarrow_j$ is interchangeable. We sketch the cases involving $*$-expansion in Fig. 3. The cases arising when $\Rightarrow_j$ is Y-elimination, too, are illustrated in Fig. 4.

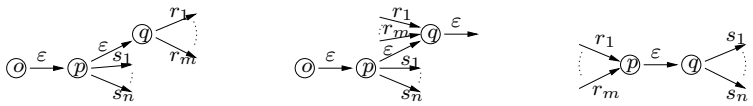


(a) Degenerate case where $p^- = p^+ = 1$; the particular type of $*$-expansion is determined by o^+ and q^-



(b) General case

Fig. 3: Local confluence of cases involving Y-elimination and $*$-expansion. The state denoted n is either q or a newly introduced state, according to q^-. Note that reverting all transitions in the figures yields further valid cases.



(a) (b) (c)

Fig. 4: Elimination-conflicts between overlapping Y-anchors centered in p and q. In (a) and (b), the resulting EFA is invariant under the order of removal. In (c) only one anchor may be eliminated, however, the resulting EFAs are isomorphic.

– $i = O$: O-elimination can be considered as removing the ε-transitions of the cycle, followed by merging all its states into a selected one among them, called the *merge-state*. If $\Rightarrow_j$ is the expansion of $t = (p, s, q)$, assume p lies on the cycle, while q does not. Choose p as the merge-state, then t remains unaffected from O-elimination, hence expansion introduces the same elements before and after O-elimination. If q is part of the cycle but not p, or $p = q$, choose q as the merge-state. If both p and q lie on the cycle and

$p \neq q$, the case of $j = *4$ is detailed in Fig. 5, the remaining cases where j is an expansion are easily dealt with in the same spirit.

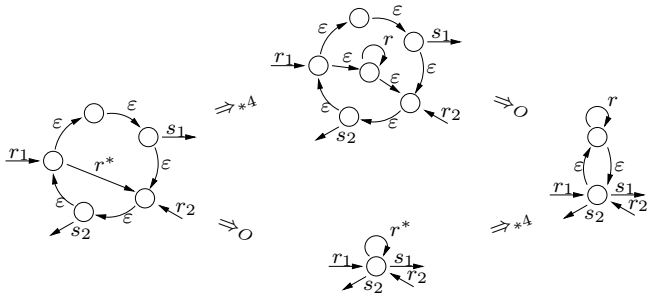


Fig. 5: Conflict between cycle-elimination and expanding a transition connecting two distinct states of the cycle.

Next consider the case that $\Rightarrow_j$ is $Y[q]$-elimination, for some state q, and where q is part of the ε-cycle relevant for O-elimination—the case where q is not on the ε-cycle in question would be again easy. By definition of Y-elimination, we must have $q^- = 1$ (resp. $q^+ = 1$ in the case of reverse Y-elimination), and there must be exactly one ε-transition entering (resp. leaving) the state q. Since $q^- = 1$ (resp. $q^+ = 1$), this transition is necessarily part of the ε-cycle in question. Hence, if O-elimination is applied first, it subsumes Y-elimination, otherwise, Y-elimination may be considererd as the first merging step of O-elimination, followed by merging a smaller cycle. Finally, if $\Rightarrow_j$ also denotes O-elimination, there is at least one common state c to both cycles, which we chose as the merge-state. Regardless of the order, both cycles may be merged into c, thus yielding the same EFA. □

We omit proving that the conversion from X_r to A_r is also locally confluent, which is due to restriction (R2). It follows that A_r is unique, too.

We add an almost trivial preprocessing step on the input expression, called *mild simplification*: Every occurrence of $s^?$ in r, s.t. $\varepsilon \in L(s)$ is replaced with s. The expression such built from r is denoted $\mathrm{simp}(r)$ [2]. Without proof, we mention that computing the strong star normal form subsumes mild simplification:

Lemma 7. *Let r be a regular expression, then* $\mathrm{simp}(r)^\bullet = \mathrm{simp}(r^\bullet) = r^\bullet$

On input r, we mildly simplify it first and then compute $A^0_{\mathrm{simp}(r)}$. Mild simplification is a reasonable first step in order to get smaller ε-NFAs:

Lemma 8. *For any regular expression r, $|A_{\mathrm{simp}(r)}| \leq |A_r|$*

Proof. Let E_1 be an EFA with transition $t = (p, s^?, q)$, and let E_2 be the EFA obtained from E_1 by replacing t with (p, s, q). Expanding t in E_1 yields another EFA E_1'; the sole difference between E_1' and E_2 is the additional transition

[2] $\mathrm{simp}(r)$ can be computed in linear time on the parse of r in a bottom-up manner

(p, ε, q) in E_1'. Now p^+ and q^- are bigger in E_1' than in E_2 — if $s = t^*$, expanding (p, s, q) in E_1' introduces at least as many elements in E_2. On the other hand, removal of p or q in E_1' may result from X- or cycle-elimination, then however, Y- or cycle-elimination would be applicable in E_2. In short, converting E_1' may never lead to an ε-NFA which is smaller than the one reached by converting E_2. Since mildly simplifying a regular expression boils down to replacing some occurrences of $s^?$ with s (in labels), the statement follows. □

The remaining part of this section deals with invariant cases of the construction under $^\circ$ and $^\bullet$. To this end, for a transition $t = (p, r, q)$ let $t^\circ := (p, r^\circ, q)$ and $t^\bullet := (p, r^\bullet, q)$. Note that since the conversions are locally confluent when respecting (R1) and (R2), $\cong$ is an equivalence relation on the class of EFAs.

Lemma 9. *Let E_1 be an EFA with looping transition $l = (q, r, q)$, and let E_2 be the EFA obtained from E_1 by replacing l with l°. Then $E_1 \cong E_2$.*

Proof. Structural induction on r: If $r \in \Sigma$ then $r = r^\circ$, satisfying the claim. Let E_1 and E_2 be as above and assume the claim is true for loops labeled s or t. Let r be

- $s{+}t$: l is replaced by $(q, s, q), (q, t, q)$, while l° is replaced by $(q, s^\circ, q), (q, t^\circ, q)$. By assumption, the pairs are interchangable, hence so are l and l°
- $s^?$: l is replaced by loops $(q, \varepsilon, q), (q, s, q)$, the first of which is an ε-cycle, hence eliminated, while the second may by assumption be replaced with $(q, s^\circ, q) = (q, s^{?\circ}, q) = l^\circ$.
- s^*: $*4$-expansion is applied, introducing an ε-cycle $\{(q, \varepsilon, q'), (q', \varepsilon, q)\}$ and a loop (q', s, q'). Eliminating the cycle identifies q and q', yielding (q, s, q) which may by assumption be replaced with $(q, s^\circ, q) = l^\circ$
- st: If $\varepsilon \notin L(st)$, then $(st)^\circ = st$ and nothing needs to be proven. So assume $\varepsilon \in L(st)$, implying $\varepsilon \in L(s)$ and $\varepsilon \in L(t)$. Let E_1' be the EFA after fully expanding r, without intermediate elimination steps. The first expansion-step replaces t_l with $\{(q, s, q'), (q', t, q)\}$ — both q and q' are still present in E_1', where they lie on an ε-cycle. Consider cycle-elimination in 'slow-motion': in a first step, only q and q' are merged, resulting in a volatile intermediate which happens to be isomorphic to the EFA constructed from fully expanding $l^\circ = (q, s^\circ{+}t^\circ, q)$ in E_2. A second step merges the remaining states, which is equivalent to two cycle-eliminations.

This covers all possible cases, and the proof is complete. □

A more general result can be established for mildly simplified expressions:

Lemma 10. *Let r be mildly simplified and assume $A_r^0 \Rightarrow^* E_1$. Let $t = (p, r, q)$ be any transition in E_1, and let E_2 be as E_1 except that t is replaced with $t^\bullet$. Then $E_1 \cong E_2$.*

Proof. The statement is true for $r \in \Sigma$. Assume it is true for transitions labeled s or t, and let E_1 and E_2 be as above. Let r be

– $s^?$: expansion replaces t with $\{(p,s,q),(p,\varepsilon,q)\}$, the first of which may by assumption be replaced with $(p,s^\bullet,q)$. Since r is mildly simplified, $\varepsilon \notin L(s)$ therefore $r^\bullet = s^{?\bullet} = s^{\bullet?}$; this implies that $(p,r^\bullet,q)$ is expanded into $(p,s^\bullet,q)$ and (p,ε,q) as well.
– s^*: expanding t yields a looping transition $l=(p',s,p')$, which may by assumption be replaced with $l^\bullet$ and by Lemma 9 with $l^{\bullet\circ}$. Clearly, expanding $t^\bullet=(q,s^{\bullet\circ *},q')$ results in $l^{\bullet\circ}$, too.

The remaining cases are straightforward. □

Theorem 11. *Let r be mildly simplified, then the ε-*NFA *constructed from r is isomorphic to the one constructed from its strong star normal form, that is, $A_r \cong A_{r^\bullet}$.*

Proof. Lemma 10 implies $A^0_r \cong A^0_{r^\bullet}$. □

Together with Lemma 7, this shows that the construction is invariant under taking strong star normal form. Put differently, strong star normal form is implicity computed upon conversion of mildly simplified regular expressions.

5 Alphabetic Width and the Size of ε-NFAs

This section is devoted to the resolution of the a research question regarding the size of ε-NFAs posed by Ilie and Yu. In the following, the *size* of an ε-NFA A is defined as the number of states plus the number of transitions in A.

Problem 12. Given a regular expression of alphabetic width n, what is the optimal bound on the size of an equivalent ε-NFA in terms of n?

Ilie and Yu state that their construction gives a bound of $9n - \frac{1}{2}$, and they remark that this does not appear to be close to optimal. In Section 4, we discussed an improved construction due to Gulan and Fernau [14]. In their original paper they gave the following bound in terms of rpn-length on the size of the constructed ε-NFA:

Theorem 13. *Let r be a regular expression of unabbreviated rpn-length n. Then the constructed ε-*NFA *A_r has size at most $22/15(n+1)+1$. Furthermore, there are infinitely many regular languages for which this bound is tight.*

That paper does not consider the abbreviated syntax for regular expressions. Fortunately, subexpressions of the form $r+\varepsilon$ do not contribute to the hardness of the conversion problem. Moreover, converting expressions that reach this bound does not involve X-elimination, so the resultinf NFA is certainly unique. The following bound, in terms of *abbreviated* rpn-length and thus slightly stronger, can be obtained along the same lines:

Theorem 14. *Let r be a regular expression of abbreviated rpn-length n. Then the constructed ε-*NFA *A_r has size at most $22/15(n+1)+1$. Furthermore, there are infinitely many regular languages for which this bound is tight.*

Proof. The analysis is the same as given in [14], except for obvious modifications to the proof of [14, Thm. 10], which is the only place where we take the use of abbreviated syntax into account. The fact that this bound is tight for infinitely many regular languages trivially carries over. □

Together with Theorem 4 and Theorem 11, we obtain the following upper bound in terms of alphabetic width:

Theorem 15. *Let r be a regular expression of alphabetic width n. If r is mildly simplified, then the constructed ε-NFA A_r has size at most $4\frac{2}{5}n+1$. Furthermore, there are infinitely many regular languages for which this bound is tight.*

Proof. Assume r is mildly simplified. Then Theorem 11 implies that the automaton A_r constructed from r is the same as the automaton $A_{r^\bullet}$ constructed from its strong star normal form. By Theorem 4, we know that $\operatorname{arpn}(r^\bullet) \leq 3n-1$. Plugging this into the statement of Theorem 14, it follows that the ε-NFA $A_{r^\bullet}$ constructed from $r^\bullet$ has size at most $22/15(3n-1+1)+1 = 4\frac{2}{5}n+1$.

Gulan and Fernau [14] also give an infinite family of regular expressions r_n showing that the bound $22/15(m-1)+1$ on the size of an ε-NFA equivalent to a regular expression of rpn-length m is optimal: For $k \geq 1$, they define the regular expression

$$r_k = \prod_{i=1}^{k} (a_i^* + b_i^*) \cdot (c_i^* + d_i^* + e_i^*)$$

of rpn-length $m = 15k-1$ and prove that every equivalent ε-NFA has size at least $22k+1 = 22/15(m+1)+1$. Since the alphabetic width of r_k is $\ell = 5k$, this shows that the bound of $22k+1 = 4\frac{2}{5}\ell+1$ stated in the theorem is tight for infinitely many regular languages. □

The examples from the last proof in turn can be used to prove that the bound from Theorem 4 for relating rpn-length and alphabetic width is tight, at least in the abbreviated case:

Theorem 16. *There exists an infinite family L_n of regular languages such that* $\operatorname{alph}(L_n) \leq n$, *whereas* $\operatorname{arpn}(L_n) \geq 3n-1$.

Proof. Consider the language L_n described by the regular expression

$$r_k = \prod_{i=1}^{k} (a_i^* + b_i^*)(c_i^* + d_i^* + e_i^*).$$

For $n = 5k$ and $L_n = L(r_{5k})$, we have $\operatorname{alph}(L_n) = 5k = n$. But the existence of a regular expression of abbreviated rpn-length less than $3n-1 = 15k-1$ would imply with Theorem 14 that there exists an ε-NFA of size less than $22k+1$ accepting L_n, which clearly contradicts the lower bound obtained by Gulan and Fernau. □

6 Conclusion and Further Research

The equivalence of regular expressions being **PSPACE**-complete [19] and not being finitely axiomatizable [1, 8], a proper normal form for regular expressions, in the strong sense that every regular language should have a single normal form, might be difficult to obtain. Also, ideally we would like to have a normal form that realizes minimum alphabetic width and minimum rpn-length, and we would like to have an efficient algorithm for computing such a normal form, two criteria that would apparently contradict the above negative theoretical results.

In this paper, we have suggested a robust notion of reduced regular expressions, the strong star normal form. This notion satisfies at least the latter two criteria, in the sense that for each regular language, there is a (i.e. at least one) regular expression expression in star normal form of minium rpn-length and of minimum alphabetic width, and that it can be computed in linear time. Our notion subsumes various previous attempts at defining such a notion [5, 9, 16]

Furthermore, we showed that the strong star normal form proves useful in various contexts: Apart from a prior application in the context of the construction of ε-free NFAs [7], we gave two further applications.

The first concerns the relation between different complexity measures for regular expressions, namely alphabetic width and (abbreviated) rpn-length. With the aid of strong star normal form, we were able to determine the optimal bound, witnessing superiority of this concept over previous attempts at defining such a notion of irreducibility, which yield only loose bounds [9, 16].

The second application concerns the comparison of descriptional complexity measures across different representations, namely alphabetic width on the one hand, and the minimum size of equivalent ε-NFAs on the other hand. Here we seized the power of a finite automaton construction proposed recently by Gulan and Fernau [14]: Under a mild additional assumption, this construction already incorporates all simplifications offered by strong star normal form. While this alone adds to the impression of robustness of the construction, we also proved an optimal bound on the relation betweeen alphabetic width and the size of finite automata, and we showed that this bound is attained by the mentioned construction.

We believe that there are various further applications in the spirit of the above results, and not only theoretical ones. For instance, the fastest known algorithm [4] for regular expression matching is still based on the classical construction due to Thompson [21]. While better constructions for ε-NFAs may not improve the asymptotic worst-case running time, we hope that these can still lead to noticeably better practical performance of NFA-based regular expression engines.

References

1. L. Aceto, W. Fokkink, and A. Ingólfsdóttir. On a question of A. Salomaa: the equational theory of regular expressions over a singleton alphabet is not finitely axiomatizable. *Theoretical Computer Science*, 209(1–2):163–178, 1998.
2. F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.

3. G. Berry and R. Sethi. From Regular Expressions to Deterministic Automata. *Theoretical Computer Science*, 48(3):117–126, 1986.
4. P. Bille and M. Thorup. Faster Regular Expression Matching. In: *International Colloquium on Automata, Languages, and Programming (Track A)*, pp. 171-182, LNCS 5555, 2009.
5. A. Brüggemann-Klein. Regular Expressions into Finite Automata. *Theoretical Computer Science*, 120(2):197–213, 1993.
6. P. Caron, J.-M. Champarnaud and L. Mignot. Multi-tilde Operators and Their Glushkov Automata. In: *Languages, Automata Theory and Applications*, pp. 290-301, LNCS 5457, 2009.
7. J.-M. Champarnaud, F. Ouardi and D. Ziadi. Normalized Expressions and Finite Automata. *International Journal of Algebra and Computation* 17(1): 141–154, 2007.
8. J. H. Conway. Regular Algebra and Finite Machines. Chapman and Hall, 1971.
9. 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.
10. W. Gelade, W. Martens, F. Neven. Optimizing Schema Languages for XML: Numerical Constraints and Interleaving. *SIAM Journal on Computing* 38(5): 2021–2043, 2009.
11. W. Gelade and F. Neven. Succinctness of the Complement and Intersection of Regular Expressions. In: *Symposium on Theoretical Aspects of Computer Science*, pp. 325–336, Dagstuhl Seminar Proceedings 08001, 2008.
12. H. Gruber and M. Holzer. Finite Automata, Digraph Connectivity, and Regular Expression Size. In: *International Colloquium on Languages, Automata and Programming*, pp.39-50, LNCS 5162, 2008.
13. H. Gruber and J. Johannsen. Optimal Lower Bounds on Regular Expression Size Using Communication Complexity. In: *Foundations of Software Science and Computational Structures*, pp. 273–286, LNCS 4962, 2008.
14. S. Gulan and H. Fernau. An Optimal Construction of Finite Automata from Regular Expressions. In: *Foundations of Software Technology and Theoretical Computer Science*, pp. 211–222, Dagstuhl Seminar Proceedings 08004, 2008.
15. J. Hromkovič, S. Seibert, T. Wilke Translating Regular Expressions into Small ε-Free Nondeterministic Finite Automata. *Journal of Computer and System Sciences* 62(4): 565–588 (2001).
16. L. Ilie and S. Yu. Follow automata. *Information and Computation* 186(1):140–162, 2003.
17. J. Lee and J. Shallit. Enumerating Regular Expressions and Their Languages. *Conference on Implementation and Application of Automata*, LNCS 3317, pp. 2–22, 2004.
18. M. Newman. On Theories with a Combinatorial Definition of "Equivalence". *Annals of Mathematics*, 43(2): 223–243, 1942.
19. A. R. Meyer and L. J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In: *Symposium on Foundations of Computer Science*, pp. 125–129, IEEE Computer Society, 1972.
20. G. Schnitger: Regular Expressions and NFAs Without epsilon-Transitions. In: *Symposium on Theoretical Aspects of Computer Science*, LNCS 3884, pp. 432–443, 2006.
21. K. Thompson. Regular expression search algorithm. *Communications of the ACM* 11(6):419–422, 1968
22. D. Wood. Theory of Computation. John Wiley & Sons, Inc., 1987