A Study on Unique Rational Operations

N. Rampersad, B. Ravikumar, N. Santean* and J. Shallit

Abstract. For each basic language operation we define its "unique" counterpart as being the operation which results in a language whose words can be obtain uniquely through the given operation. As shown in the preliminaries of this paper, these unique operations can arguably be viewed as combined basic operations, placing this work in the popular area of state complexity of combined language operations. Considering unique rational operations, we are questioning about their state complexity. For an answer, we provide upper bounds and empirical results meant to cast light into this matter. Equally important, we hope to have provided a generic methodology for estimating their state complexity. Yet, the core value of this work may lay more in its initiative and approach rather than any particular result.

Keywords: unique concatenation, unique union, unique star, state complexity

1 Introduction

Finite automata (FA) are ubiquitous objects in computer science theory as much as in computer applications. They model finite state systems, from a door lock to the entire Universe – in some views – and check the syntax of regular languages. Computers are deterministic finite automata (DFA), and the English lexicon can be spell-checked by a FA. Recently, automata have found new practical applications, such as in natural language processing [1], communications [2] and software engineering [3] – applications increasingly demanding in terms of computing resources. In this context, the study of state complexity of operations on FA and their languages has become a topic of paramount importance.

From the Formal Languages point of view, FA are yet another tool for defining the family of regular (or rational, as known in certain formalisms) languages, along with regular expressions and right linear grammars. They arise from the perpetual mathematical effort of expressing infinite objects by finite means. In this paper we pursue a new direction in their study, namely, the succinctness of expressing a language obtained by certain `unique` language operations, in terms of the descriptional complexity of the languages involved.

Similar directions have been taken before in Automata Theory, e.g., for `basic` language operations [4–7] and combined operations [8–10] on regular

* corresponding author, nsantean@iusb.edu

languages. In the present paper, we make a leap from the current trends, by addressing the succinctness of some special operations; namely, we address those operations derived from the basic ones, that reach a result in an unique manner: an object obtained in two (or more) ways by applying the given operation is excluded from the result. These unique operations can be expressed, as we show in the beginning of Section 3, as combined basic operations (including intersection, shuffle and homomorphism); nevertheless, those complex formulas help very little in the estimation of their state complexity. In the same section we define the so-called unireg expressions (based on unique operations) and make the connection with unambiguous regular expressions. We prove that unireg expressions express the family of regular languages exactly. Then, we study the closure properties of some other families of languages under these unique operations. In Section 4 we give an upper bound on the state complexity of the language of those words accepted unambiguously by an arbitrary NFA. This construction turned out to be generic enough to provide a common approach for the estimation of state complexity of all unique operations. We then consider the state complexity of unique union, unique concatenation and unique star, and establish upper bounds. Although we are not able to show that some bounds are tight, there is strong empirical evidence for this fact, enforced in several experiments where these upper bounds are consistently reached. In Section 5 we study the complexity of some decision problems related to unireg expressions, namely the membership and non-emptiness problems. Finally, in Section 6 we made a practical connection between unique concatenation and 2-DFA with a pebble.

On a last note, this work is in progress, and there remains a great deal to be carried out. Several directions of further research can be found in the last section of the paper. We feel that much more needs to be done, and it is our belief that, beside the technical aspect of this study, we succeeded to open some new doors in the area of the state complexity of combined operations on regular languages.

2 Definitions and Notations

Let Σ be an alphabet, i.e., a nonempty, finite set of symbols (letters). By Σ^* we denote the set of all finite words (strings of symbols) over Σ, and by ε, the empty word (a word having zero symbols). The operation of concatenation (juxtaposition) of two words u and v is denoted by $u \cdot v$, or simply uv. For $w \in \Sigma^*$, we denote by w^R the word obtain by reversing the order of symbols in w.

A nondeterministic finite automaton over Σ, NFA for short, is a tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is a next-state function, q_0 is an initial state and $F \subseteq Q$ is a set of final states. δ is extended over $Q \times \Sigma^*$ in the usual way. M is deterministic (DFA) if $\delta : Q \times \Sigma \to Q$. We

consider `complete` DFA's, that is, those whose transition function is a total function.

The size of M, *size*(M), is the total number of its states. When we want to emphasize the number of states of M, we say that M is an n-state NFA, and similarly for transitions. The language of M, denoted by $L(M)$, belongs to the family of `regular languages` and consists of those words accepted by M in the usual sense.

A state of M is `accessible` if there exists a path in the associated transition graph, starting from q_0 and ending in that state. By convention, there exists a path from each state to itself labeled with ε. A state is `coaccessible` if there exists a path from that state to some final state. A state which is both accessible and coaccessible is called `useful`, and an automaton which has only useful states is called `trim`.

For a background on finite automata and regular languages we refer the reader to [11].

Definition 1. *Let L, R be languages over Σ. By `unique concatenation` of L and R, denoted as $L \circ R$, we understand the set*

$$L \circ R = \{w \mid w = uv,\ u \in L,\ v \in R,\ \textit{and this factorization is unique}\}\ .$$

Definition 2. *Let L be a language over Σ. By `unique star` of L, denoted as L°, we understand the set*

$$L^\circ = \{\varepsilon\} \cup \{w \mid w = u_1 \dots u_n,\ n \in \mathbb{N},\ u_i \in L \setminus \{\varepsilon\}\ \forall 1 \le i \le n;$$
$$\textit{and this factorization is unique}\}$$

Notice that we could have defined L° such that the factorization in the above definition involves ε as well. However, in this case, if L contained ε, then L° would be empty. Moreover, the connection with unambiguous regular expressions (Lemma 2) could not be made. For these reasons we adopt the above definition.

Notation wise, we denote $L \diamond R = LR \setminus (L \circ R)$ and $L^\diamond = L^* \setminus L^\circ$, and we refer to these operations as `poly concatenation` and `poly star`. Note that $\varepsilon \notin L^\diamond$. We also consider `unique square` and `poly square`, given by $L^{\circ 2} = L \circ L$ and $L^{\diamond 2} = L^2 \setminus L^{\circ 2}$.

Definition 3. *Let L, R be languages over Σ. By `unique union` of L and R, denoted as $L \mathbin{\mathring{\cup}} R$, we understand the set*

$$L \mathbin{\mathring{\cup}} R = (L \setminus R) \cup (R \setminus L)\ ,$$

in other words, the symmetric difference of L and R.

Definition 4. *By a unique regular expression, or* `unireg expression` *for short, we understand the well-formed, parenthesized formula with the following operators: all symbols $a \in \Sigma$ and ε are nullary, $^\circ$ is unary, and $\circ$, $\oplus$ are binary. The operator $\oplus$ is the counterpart of $\overset{\circ}{\cup}$ in expressions.*

Since $\{a\} \circ \{b\} = a \circ b = ab$, for any symbols $a, b \in \Sigma$, we denote the unique concatenation of symbols by juxtaposition, as for the usual concatenation. By convention, the empty unireg expression, enclosed or not in parentheses, denotes the empty set (if $e = ()$ then $\mathscr{L}(e) = \emptyset$). The language $\mathscr{L}(e)$, denoted by unireg expression e, is defined recursively as in the case of regular expressions. However, for reasons which will become apparent later, *we consider only fully-parenthesized expressions.*

3 Properties

3.1 Regular Languages

In the following, we denote by $\sqcup\!\sqcup$ the shuffle operation on words or languages. We also use two new symbols, $1, 2 \notin \Sigma$, and we denote by h_{12} the homomorphism which deletes these symbols in words: $h_{12} : (\Sigma \cup \{1,2\})^* \to \Sigma^*$, $h_{12}(a) = a$, $\forall a \in \Sigma$, and $h_{12}(1) = h_{12}(2) = \varepsilon$.

Lemma 1. *If L and R are regular languages then $L \circ R$, L°, and $L \overset{\circ}{\cup} R$ are regular languages.*

Proof. It is clear that $L \overset{\circ}{\cup} R$ is regular. For the other two, it suffices to observe that the languages

$$L \diamond R = h_{12}\Big((L1R \sqcup\!\sqcup 2) \cap (L2R \sqcup\!\sqcup 1) \cap \big(\Sigma^*(1\Sigma^+2 + 2\Sigma^+1)\Sigma^*\big)\Big) \quad \text{and}$$

$$L^\diamond = h_{12}\Big(\big((L'1)^* \sqcup\!\sqcup 2^*\big) \cap \big((L'2)^* \sqcup\!\sqcup 1^*\big) \cap \big(\Delta^*(1\Sigma^+2 + 2\Sigma^+1)\Delta^*\big)\Big) \ ,$$

where $L' = L \setminus \{\varepsilon\}$ and $\Delta = \Sigma \cup \{1,2\}$, are both regular, their definition involving operations under which regular languages are closed. Then, since $L \circ R = LR \setminus (L \diamond R)$ and $L^\circ = \{\varepsilon\} \cup L^* \setminus L^\diamond$, the conclusion follows. □

Let R be a regular expression over Σ, containing r occurrences of symbols in Σ (multiple occurrences are counted separately). Denote $\Sigma' = \{a_1, \ldots, a_r\}$ to be an alphabet of r new symbols, and consider $h_R : \Sigma'^* \to \Sigma^*$ the homomorphism which maps a_i to the symbol in Σ representing the $i'th$ occurrence of a symbol in R. By R_h we denote the regular expression obtain from R by replacing each

$i'th$ occurrence of a symbol in Σ with the corresponding $a_i \in \Sigma'$. For example, if $R = (a+ab)^*b^*$, then $R_h = (a_1 + a_2a_3)^*a_4^*$, and $h_R(a_1) = h_R(a_2) = a$, $h_R(a_3) = h_R(a_4) = b$.

Definition 5. *With the above notations, a regular expression R is* `unambiguous` *if and only if the restriction of h_R to $\mathscr{L}(R_h)$ is injective.* (This definition is equivalent to that given in [13].)

According to this definition, the above expression $R = (a+ab)^*b^*$ is not unambiguous, since $a_1a_4, a_2a_3 \in L(R_h)$ and $h_R(a_1a_4) = h_R(a_2a_3) = ab$. An unambiguous regular expression "matches" any word in at most one way.

Notice that unambiguity does not ensure unique word parsing. For example, the regular expression $R = (a^* + b)^*$ is unambiguous; however, the word *aa* can be parsed in two ways: by iterating the first Kleene star twice, and by iterating the second Kleene star twice - thus *aa* has two parsing trees. For ensuring unique parsing, there exists another notion, that of `strongly unambiguous` regular expression, studied e.g. in [12], and which will not be considered in this paper.

Let R be a fully parenthesized regular expression over Σ and denote by $\tilde{R}$ the unireg expression obtained from R by replacing its regular operations by their unique counterparts. (We stress that R should be fully parenthesized.)

Lemma 2. *If R is unambiguous then $\mathscr{L}(R) = \mathscr{L}(\tilde{R})$.*

Proof. The proof is by induction on the number of regular operations in R. The case when R has no, or only one, operation can be easily verified. Suppose that the claim holds for $n-1$ operations, $n \geq 2$, and that R has n operations. We have the cases: $R = R_1 \cdot R_2$, $R = R_1 + R_2$ or $R = R_1^*$, where R_1, R_2 are subexpressions of R consisting of at most $n-1$ operations. We treat only the first case: $R = R_1 \cdot R_2$, and observe that $\tilde{R} = \tilde{R}_1 \circ \tilde{R}_2$. First notice that if R is unambiguous then all its subexpressions must necessarily be unambiguous as well. Thus, R_1 and R_2 are unambiguous. This means, by induction hypothesis, that $\mathscr{L}(R_1) = \mathscr{L}(\tilde{R}_1)$ and $\mathscr{L}(R_2) = \mathscr{L}(\tilde{R}_2)$. Then, $\mathscr{L}(R) = \mathscr{L}(R_1 \cdot R_2) = \mathscr{L}(R_1) \cdot \mathscr{L}(R_2) = \mathscr{L}(\tilde{R}_1) \cdot \mathscr{L}(\tilde{R}_2)$. It now suffices to prove that $\mathscr{L}(\tilde{R}_1) \cdot \mathscr{L}(\tilde{R}_2) = \mathscr{L}(\tilde{R}_1) \circ \mathscr{L}(\tilde{R}_2)$, in other words, that $\mathscr{L}(\tilde{R}_1) \diamond \mathscr{L}(\tilde{R}_2) = \emptyset$. Suppose there exist $u_1, u_2 \in \mathscr{L}(R_1)$ and $v_1, v_2 \in \mathscr{L}(R_2)$ such that $u_1v_1 = u_2v_2 = w$. This leads to a contradiction as follows.

Let R_h be the regular expression over Σ', as defined previously. R_h splits into R_h^1 and R_h^2, which are the subexpressions corresponding to R_1 and R_2. Then, if a symbol a_i occurs in R_h^1, it cannot occur in R_h^2, and vice-versa. Since $w \in \mathscr{L}(R)$ and R is unambiguous, there exists a unique $w' \in \mathscr{L}(R_h)$ such that $h(w') = w$. Similarly, there exist uniquely $u_1', u_2' \in \mathscr{L}(R_h^1)$ and $v_1', v_2' \in \mathscr{L}(R_h^2)$ such that $h(u_1') = u_1$, $h(u_2') = u_2$, $h(v_1') = v_1$ and $h(v_2') = v_2$. Since h is a homomorphism, clearly $u_1'v_1' = u_2'v_2' = w'$. Now, assume that $|u_1| < |u_2| = t$ and that

$w' = a_{i_1} \dots a_{i_k}$. Then, a_{i_t} is a symbol occurring in R_h^1 by the fact that $u_2' \in \mathscr{L}(R_h^1)$. At the same time, a_{i_t} is a symbol occurring in R_h^2, by the fact that $v_1' \in \mathscr{L}(R_h^2)$ and v_1' has a proper prefix which is a suffix of u_2'. But we have already made the observation that R_h^1 and R_h^2 cannot share common symbols, and this is a contradiction.

In conclusion, $\mathscr{L}(\tilde{R}_1) \diamond \mathscr{L}(\tilde{R}_2) = \emptyset$, $\mathscr{L}(R) = \mathscr{L}(\tilde{R})$, and the induction is complete. □

Notice carefully that the converse of this lemma does not hold. Indeed, if $R = a + (a + a)$, then $\tilde{R} = a \oplus (a \oplus a)$ and $\mathscr{L}(R) = \mathscr{L}(\tilde{R}) = \{a\}$. However, R is obviously ambiguous.

Corollary 1. *Unireg expressions define the family of regular languages.*

Proof. Let L be a regular language. Without loss of generality, we may assume that $\varepsilon \notin L$ (otherwise, we construct a unireg expression E for $L \setminus \{\varepsilon\}$ and then consider the expression $\varepsilon \oplus E$ for L).

Now, there exists an unambiguous regular expression R for L – such an expression can be obtained from a DFA for L or by an algorithm as in [13]. Let $\tilde{R}$ be the unireg expression obtained by replacing each operation in R by the corresponding unique operation. Then by Lemma 2 we have $\mathscr{L}(R) = \mathscr{L}(\tilde{R}) = L$. □

Remark 1. Unique concatenation is not associative. Indeed, consider the following examples:

1. For $L_1 = \{b, ba^2\}$, $L_2 = \{a^3, a^4\}$ and $L_3 = \{ab, a^2b, a^3b\}$, we have: $(L_1 \circ L_2) \circ L_3 = \{ba^4b, ba^6b, ba^9b\}$, and $L_1 \circ (L_2 \circ L_3) = \{ba^4b, ba^6b, ba^9b, ba^7b\}$.

2. For $L_1 = \{b, ba\}$, $L_2 = \{a^3, a^4\}$, $L_3 = \{ab, a^2b, a^3b\}$ we have: $(L_1 \diamond L_2) \diamond L_3 = \emptyset$ and $L_1 \diamond (L_2 \diamond L_3) = \{ba^6b\}$.

3. Finally, the unique concatenation and unique star are unrelated: if $L = \{a, b, b^2\}$ then $ab^2 \in (L \circ L) \circ L$; however $ab^2 \notin L^\circ$.

Notice that the reverse operation is compatible with the unique operations and with their "poly" counterparts. Indeed, for $L_1, L_2 \subseteq \Sigma^*$ we have:

$$(L_1 \circ L_2)^R = L_2^R \circ L_1^R, \quad (L_1 \mathbin{\mathring{\cup}} L_2)^R = L_1^R \mathbin{\mathring{\cup}} L_2^R, \quad (L_1^\circ)^R = (L_1^R)^\circ,$$
$$(L_1 \diamond L_2)^R = L_2^R \diamond L_1^R, \quad (L_1 \cap L_2)^R = L_1^R \cap L_2^R, \quad (L_1^\diamond)^R = (L_1^R)^\diamond.$$

Reviewing the notation, we have:

- unique concatenation $L_1 \circ L_2$, and poly concatenation: $L_1 \diamond L_2$,
- unique star L°, and poly star $L^\diamond$,
- unique square $L^{\circ 2} = L \circ L$, and poly square $L^{\diamond 2} = L \diamond L$,
- unique union $L_1 \mathring{\cup} L_2$, and poly union $L_1 \mathring{\cup} L_2$,
- *for completeness, we also denote* $\mathtt{unique\ shuffle}\ L_1 \mathring{\sqcup\!\sqcup} L_2$, *and* $\mathtt{poly\ shuffle}\ L_1 \mathring{\sqcup\!\sqcup} L_2$.

Various connections among these operations can be drawn. For example,

Lemma 3. *If L is an arbitrary language then*

$$(L^* \setminus \{\varepsilon\})^{\diamond 2} = L^\diamond, \text{ and } (L^* \setminus \{\varepsilon\})^{\circ 2} = L^\circ \setminus \{\varepsilon\}.$$

Proof. (sketch) If $w \in L^\diamond$ then $w \neq \varepsilon$, $w = u_1 u_2 \dots u_i = v_1 v_2 \dots v_j$, with $u_1, ..., u_i, v_1, ..., v_j \in L$, and there exists k such that $u_1 = v_1$, …, $u_{k-1} = v_{k-1}$ and $u_k \neq v_k$. Then $x = u_1 \dots u_k \in L^*$, $y = u_{k+1} \dots u_i \in L^*$, $x' = v_1 \dots v_k \in L^*$, $y' = v_{k+1} \dots v_j \in L^*$, $x \neq x'$, $y \neq y'$, $xy = x'y' \in L^* \setminus \{\varepsilon\}$. Thus, $w \in (L^* \setminus \{\varepsilon\}) \diamond (L^* \setminus \{\varepsilon\})$. Conversely, if $w \in (L^* \setminus \{\varepsilon\})^{\diamond 2}$ then $w = xyz$, with $x, xy, yz, z \in L^*$ and $y \neq \varepsilon$. Clearly $w \in L^*$, and it has two different factorizations into words in L.

If $w \in L^\circ \setminus \{\varepsilon\}$, suppose by contradiction that $w = xyz$ with $x, xy, yz, z \in L^* \setminus \{\varepsilon\}$. Then denote $u = x, v = yz, u' = xy, v' = z$, and $u, u', v, v' \in L^*$. It is clear that $uv = u'v' \in L^*$, thus $w \in L^\diamond$. This contradicts the fact that $w \in L^\circ$, for $L^\circ \cap L^\diamond = \emptyset$. Conversely, if $w \in (L^* \setminus \{\varepsilon\}) \circ (L^* \setminus \{\varepsilon\})$ then $w \neq \varepsilon$ and suppose by contradiction that $w \notin L^\circ$. Then $w = u_1 u_2 \dots u_i = v_1 v_2 \dots v_j$, with $u_1, ..., u_i, v_1, ..., v_j \in L$, and there exists a k such that $u_1 = v_1, \dots, u_{k-1} = v_{k-1}$ and $u_k \neq v_k$. If we denote $x = u_1 \dots u_k$, $y = u_{k+1} \dots u_i$, $x' = v_1 \dots v_k$, and $y' = v_{k+1} \dots v_j$, we have that $x, y, x', y' \in L^*$, thus $xy \in L^*$, $x'y' \in L^*$, and they denote two distinct factorizations of w into words in L. Thus $w \in L^\diamond$ - a contradiction. □

3.2 Context-Free Languages

Proposition 1. *The following families are not closed under unique union: DCF, CF, and linear CF.*

Proof. It is clear that $\Sigma^* \mathring{\cup} L = \overline{L}$. However, it is well-known that the families CF and linear CF are not closed under complement.

For the case of the family DCF, let $L_1 = \{a^i b^j c^k : i \neq j\}$ and $L_2 = \{a^i b^j c^k : j \neq k\}$. Clearly L_1 and L_2 are deterministic CFL's. Then

$$L_1 \mathring{\cup} L_2 = \{a^i b^j c^k : i = j \neq k \text{ or } i \neq j = k\}.$$

We claim that $L_1 \mathbin{\mathring{\cup}} L_2$ is not context-free. Suppose it is and let n be the constant of Ogden's lemma. Let $z = a^n b^n c^{n+n!}$ and mark the b's. Let $z = uvwxy$ be a decomposition satisfying the conditions of Ogden's lemma. We have the following cases:

- $v = a^p$ and $x = b^q$. If $p = q$, then $uv^{n!/p}wx^{n!/q}y = a^{n+n!}b^{n+n!}c^{n+n!} \notin L_1 \mathbin{\mathring{\cup}} L_2$. If $p \neq q$, then clearly $uv^2wx^2y \notin L_1 \mathbin{\mathring{\cup}} L_2$.
- $v = b^p$ and $x = c^q$. Then clearly $uv^2wx^2y \notin L_1 \mathbin{\mathring{\cup}} L_2$.
- $vwx = b^p$. Then clearly $uv^2wx^2y \notin L_1 \mathbin{\mathring{\cup}} L_2$.

Thus the decomposition $z = uvxwy$ fails to satisfy the conditions of Ogden's lemma, a contradiction; we conclude that $L_1 \mathbin{\mathring{\cup}} L_2$ is not context-free, as required. □

Proposition 2. *The following families are not closed under unique concatenation: DCF, CF and linear CF.*

Proof. Consider the following CFL: $L_1 = \{a^n \mid n \geq 1\} \cup \{a^n b^n \mid n \geq 1\}$ and $L_2 = \{c^n \mid n \geq 1\} \cup \{b^n c^n \mid n \geq 1\}$. It is easy to see that

$$L_1 \circ L_2 = \{a^n c^m \mid m,n \geq 1\} \cup \{a^n b^{n+m} c^m \mid m,n \geq 1\} \cup$$

$$\cup \{a^n b^m c^m \mid m \neq n; m,n \geq 1\} \cup \{a^n b^n c^m \mid m \neq n; m,n \geq 1\},$$

that is, the only words in $L_1 \cdot L_2$ which can be written as a concatenation in more than one way are $a^n b^n c^n$, $n \geq 1$. We can prove that $L_1 \circ L_2$ is not context-free by Ogden's lemma.

Assume by contradiction that it is, and let N be the constant of Ogden's lemma. Take the word $z = a^N b^N c^{N+N!} \in L_1 \circ L_2$. By Ogden's lemma, if we mark the a's, there exists a factorization $z = uvwxy$ such that vx has at least one marked symbol, vwx has at most N marked symbols and $uv^i wx^i y \in L_1 \circ L_2$ for all $i \geq 0$. One can observe that such factorization must necessarily have $v = a^t$ and $x = b^t$, $0 < t \leq N$. But then, for $i = 1 + N!/t$ we have $uv^i wx^i y = a^{N+N!}b^{N+N!}c^{N+N!}$ contradicting that such word must not be in $L_1 \circ L_2$.

Since L_1 and L_2 are both deterministic and linear CF languages, the conclusion follows. □

Finally, we show the following claim about unique Kleene star.

Proposition 3. *The following families are not closed under unique Kleene star: CF and linear CF.*

Proof. Let L_1 and L_2 be as in the previous proposition. Clearly, $L_1 \cup L_2$ is context-free. We can show that $(L_1 \cup L_2)^\circ$ is not context-free. In fact, it is easy to see that

$$(L_1 \cup L_2)^\circ \cap a^*b^*c^* = \{a^n c^m \mid m,n \geq 1\} \cup \{a^n b^{n+m} c^m \mid m,n \geq 1\} \cup$$

$$\cup \{a^n b^m c^m \mid m \neq n; m,n \geq 1\} \cup \{a^n b^n c^m \mid m \neq n; m,n \geq 1\} \cup$$

$$\cup \{a^{n+m} b^m \mid m,n \geq 1\} \cup \{b^n c^{n+m} \mid m,n \geq 1\}$$

It can be shown as before that this is not a CFL language. From this, it follows that CFL is not closed under unique star. Since $L_1 \cup L_2$ is a linear CFL, it follows that linear CFL is not closed under unique star. □

4 State Complexity

Before dealing with the state complexity of unique operations, we first prove the following result. The idea behind the construction given in the proof will be useful in proving upper bounds for the state complexities of unique concatenation and unique star.

Lemma 4. *Let A be an NFA with no ε-transitions, of size m, and let L be the language of those words in Σ^* which are accepted unambiguously by A. Then L is regular and its state complexity is at most $3^m - 2^m + 1$.*

Proof. We construct a DFA whose states are vectors with m components, showing the number of paths from the initial state to each state: 0, 1 or 2 (standing for "two or more paths"). The final states are those vectors which denote exactly one path to one final state of the initial NFA.

Formally, let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$. We construct a DFA $B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ for L of size $3^m - 2^m + 1$ as follows:

1. $Q_B = V_B - V'_B \cup \{s\}$, where V_B is the set of vectors with m elements, each element taking values in $\{0,1,2\}$, and V'_B is the subset of V_B consisting of those vectors that do not have the value 1 in any component. Clearly, $|Q_B| = 3^m - 2^m + 1$. State s has the role of a sink state.
2. $q_B = (1,0,\ldots,0)$, F_B is the set of all vectors v such that the sum of all components of v corresponding to the final states of A is precisely 1.
3. For all $a \in \Sigma$, we denote by M_a the incidence matrix of A with respect to the symbol a. Then, for all $v \in V_B - V'_B$,

$$\delta_B(v,a) = \begin{cases} vM_a & \text{if } vM_a \notin V'_B \\ s & \text{if } vM_a \in V'_B, \end{cases}$$

and $\delta_B(s,a) = s$. Here the matrix multiplication is done as usual, but with $\oplus$ and $\otimes$ as component-wise addition and multiplication, described as follows: for $a,b \in \{0,1,2\}$, we define $a \oplus b = min(a+b,2)$ and $a \otimes b = min(a \cdot b,2)$. (See [14] for an early use of these operations.)

□

Notice that the construction can be modified to recognize the language of those words which are accepted ambiguously by the NFA: just make the appropriate states final. Since this symmetry holds for most constructions proposed throughout the paper, we make, a priori, the following conjecture:

Conjecture. The state complexity results on unique operations (which will be described in the sequel) hold for "poly" operations as well.

While we cannot currently show that the bound given in Lemma 4 is tight, we can give an exponential lower bound, as follows. For $k \geq 0$, define a language

$$L_k = (0+1)^* 0 (0+1)^k 1 (0+1)^*.$$

The languages L_k (or variations thereof) have been used by several authors [15–18] to prove lower bounds for non-deterministic state complexity. The language L_k consists of all words containing *at least one occurrence* of a word in $0(0+1)^k 1$.

Now consider the language

$$UL_k = (0 \oplus 1)^\circ \circ 0 (0 \oplus 1)^k 1 \circ (0 \oplus 1)^\circ$$

obtained from the regular expression for L_k by replacing the ordinary operations with the unique ones. The language UL_k consists of all words containing *exactly one occurrence* of a word in $0(0+1)^k 1$.

Lemma 5. *Any NFA accepting UL_k has at least 2^k states.*

Proof. For every word $x \in \{0,1\}^k$, define a pair $(0x, 1x)$. Note that $0x1x$ is in UL_k, since there is exactly one instance where a 0 is followed by a 1 k positions later. However, for any 2 distinct words x and y, at least one of the words $0x1y$ or $0y1x$ must contain *two* occurrences of a subword in $0(0+1)^k 1$ (since x and y must mismatch in at least one place). Thus, at least one of $0x1y$ or $0y1x$ is not in L. Since there are 2^k pairs, it follows from a result of Birget [19] that any NFA for UL_k has at least 2^k states. □

We easily deduce the following results.

Proposition 4. *There exists an NFA M_k with $O(k)$ states such that any DFA, NFA, or regular expression for the set of words accepted unambiguously by M_k has size at least 2^k.*

Proof. The language L_k is accepted by an NFA M_k with $O(k)$ states. The set of words accepted unambiguously by M_k is exactly UL_k. The result now follows from Lemma 5. □

Proposition 5. *There exists a regular language generated by a unireg expression of size $O(k)$ such that any equivalent DFA, NFA, or regular expression has size at least 2^k.*

Proof. The language UL_k has the desired properties. □

4.1 Unique Union

For the unique union, we observe that given two DFA A and B, of size m and n respectively, we can easily construct a DFA of size mn for $L(A)\mathbin{\mathring{\cup}}L(B)$ by performing the cross-product of A and B and by setting as final states those state pairs which have exactly one component final. We prove that this upper bound of mn is tight.

Theorem 1. *For $m,n \geq 3$, let L_1 and L_2 be accepted by DFA's with m and n states respectively. The state complexity of $L_1 \mathbin{\mathring{\cup}} L_2$ is mn.*

Proof. For $m,n \geq 3$, we use the languages

$$A = \{w \in \{0,1\}^* : |w|_0 \equiv m-1 \bmod m\}$$

and

$$B = \{w \in \{0,1\}^* : |w|_1 \equiv n-1 \bmod n\},$$

which were used by Maslov [20] to prove a similar result for the ordinary union. Clearly, A is accepted by an m-state DFA, B is accepted by an n-state DFA, and $C = A \mathbin{\mathring{\cup}} B$ is accepted by an mn-state DFA. We show that mn states are necessary.

For integers i,i' and j,j', $0 \leq i,i' \leq m-1$ and $0 \leq j,j' \leq n-1$, let $x = 0^i1^j$ and $y = 0^{i'}1^{j'}$ be distinct words. To complete the proof it is enough to show that x and y are inequivalent with respect to the Myhill–Nerode equivalence relation. We have several cases.

Case 1: $i,i' < m-1$ and $j,j' < n-1$. If $i \neq i'$, then $x0^{m-1-i} \in C$ and $y0^{m-1-i} \notin C$, so x and y are inequivalent. If $j \neq j'$, then $x1^{n-1-j} \in C$ and $y1^{n-1-j} \notin C$, so x and y are again inequivalent.

Case 2: $i, i' = m-1$, and $j, j' < n-1$. Then $x1^{n-1-j} \notin C$ and $y1^{n-1-j} \in C$, so x and y are inequivalent.

Case 3: $i, i' < m-1$ and $j, j' = n-1$. Then $x0^{m-1-i} \notin C$ and $y0^{m-1-i} \in C$, so x and y are inequivalent.

Case 4: $i < m-1$, $j < n-1$, $i' = m-1$, and $j' < n-1$. Then $x \notin C$ and $y \in C$, so x and y are inequivalent.

Case 5: $i < m-1$, $j < n-1$, $i' < m-1$, and $j' = n-1$. Then $x \notin C$ and $y \in C$, so x and y are inequivalent.

Case 6: $i < m-1$, $j = n-1$, $i' = m-1$, and $j' < n-1$. If $i < m-2$, then $x0 \in C$ and $y0 \notin C$. If $j' < n-2$, then $x1 \notin C$ and $y1 \in C$. If $i = m-2$ and $j' = n-2$, then $x00 \in C$ and $y00 \notin C$. Thus x and y are inequivalent.

Case 7: $i = m-1$, and $j = n-1$. If either $i' = m-1$ or $j' = n-2$, then $x \notin C$ and $y \in C$, so x and y are inequivalent. Otherwise, apply the argument of one of the preceding cases, as appropriate, to $x0$ and $y0$ to show that x and y are inequivalent.

Thus all mn distinct pairs x,y are inequivalent. It follows that any DFA accepting C has at least mn states. This completes the proof. □

We now approach the more difficult problem of determining the state complexities of unique concatenation and unique star.

4.2 Unique Concatenation

We start with a naive approach, aiming at implementing the poly concatenation first, and then derive the unique operation.

Let $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ accept L_1 and L_2, respectively. Define an NFA $M = (Q_C, \Sigma, \delta_C, s_C, F_C)$ as follows:

$Q_C = Q_A \cup (Q_A \times Q_B) \cup (Q_B \times Q_B)$, $s_C = s_A$ and
$F_C = F_B \times F_B$ if $s_B \notin F_B$
$F_C = F_B \times F_B \cup F_A \times F_B$, else.
δ_C is defined as follows ($a \in \Sigma$) :

$\delta_C(q,a) = \{\delta_A(q,a)\}$ if q is in $Q_A - F_A$,
$\delta_C(q,a) = \{\delta_A(q,a), [\delta_A(q,a), \delta(s_B,a)]\}$ if q is in F_A,
$\delta_C([q,r],a) = \{[\delta_A(q,a), \delta_B(r,a)]\}$ if $[q,r]$ is in $(Q_A - F_A) \times Q_B$,
$\delta_C([q,r],a) = \{[\delta_A(q,a), \delta_B(r,a)], [\delta_B(s_B,a), \delta_B(r,a)]\}$ if $[q,r] \in F_A \times Q_B$,
$\delta_C([q,r],a) = \{[\delta_B(q,a), \delta_B(r,a)]\}$ if $[q,r]$ is in $Q_B \times Q_B$.

The following observations are straightforward:

(i) $L_1 \circ L_2 = \overline{L(C)} \cap L_1 L_2$.

(ii) The number of states in C is $m + mn + n^2$.

The number of states in C can be reduced to $m + n + mn + \binom{n}{2}$ by observing that the ordered pairs $[p,q],[q,p] \in Q_B \times Q_B$ are indistinguishable and can be merged. When converting this NFA into a DFA, note that every subset contains exactly one occurrence of a set from Q_A and hence the number of subsets generated is at most $m2^{mn+n(n+1)/2}$.

A tight upper-bound on the number of states in a DFA accepting $L_1 L_2$ was obtained in [21], as being $m2^n - k2^{n-1}$, where k is the number of final states in A. From these, it follows that an upper-bound on the number of states in a DFA accepting $L(A) \circ L(B)$ is given by $m2^{mn+n(n+1)/2}(m2^n - k2^{n-1})$.

However, we can do better than this, by constructing a DFA for the unique concatenation directly, as shown in the proof of the following result.

Theorem 2. *The state complexity of unique concatenation for regular languages is at most $m3^n - k3^{n-1}$, where m and n are the sizes of the input DFAs, and k is the number of final states of the first DFA.*

Proof. (sketch) Let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ be the input DFAs, of size m and n respectively. For proving the upper bound we construct a DFA $C = (Q_C, \Sigma, \delta_C, q_C, F_C)$ for $L(A) \circ L(B)$, of size $m3^n - k3^{n-1}$:

1. $Q_C = Q_A \times V_B - F_A \times V'_B$, where V_B is the set of vectors with n elements, each element taking values in $\{0,1,2\}$, and V'_B is the subset of V_B consisting of those vectors which have 0 in their first component. Clearly, $|Q_C| = m3^n - k3^{n-1}$.
2. $q_C = \langle q_A, (0,\ldots,0)\rangle$ if $q_A \notin F_A$ and $q_C = \langle q_A, (1,0,\ldots,0)\rangle$ otherwise, F_C is the set of those states $\langle q, v\rangle$ such that the sum of all components of v corresponding to the final states of B is precisely 1.
3. For all $a \in \Sigma$, we denote by M_a the incidence matrix of B with respect to the symbol a. Then $\delta_C(\langle q, v\rangle, a) = \langle \delta_a(q,a), v'\rangle$, where $v' = vM_a$ if $\delta(q,a) \notin F_A$ and $v' = vM_a + (1,0\ldots,0)$ otherwise. The matrix multiplication is done as usual, but with $\oplus$ and $\otimes$ as component-wise addition and multiplication, operations described in the proof of Lemma 4.

The idea of this construction was to compute the "multiplicity" of ambiguous computations of the NFA for $L(A)L(B)$, as inspired by the proof of Lemma 4.

□

Considering the DFA N_n in Figure 1, we have found that this DFA is a state complexity worst-case for unique square, proving that the upper bound is reached for this operation:

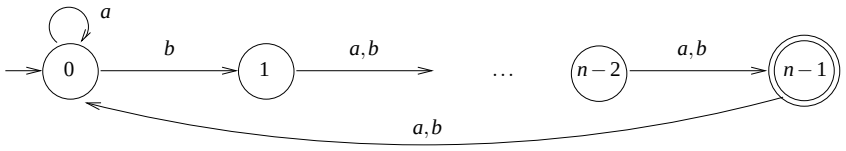

Fig. 1. N_n: a worst case for unique square.

Lemma 6. *For $n \geq 3$, the state complexity of $L(N_n)^{\circ 2}$ is $n3^n - 3^{n-1}$, thus this is a sharp upper bound for unique square (when $k = 1$).*

Proof. We prove that the construction in the proof of Theorem 2 leads to a minimal DFA, by proving its total reachability and non-mergibility. Consider that the states of N_n are numbered (and named) from 0 to $n-1$, and recall that the corresponding DFA (as constructed in Theorem 2) has states of the form $\langle i, (x_1, \ldots x_n)\rangle$, where $x_j \in \{0,1,2\}$ and the component-wise operations of the vector $(x_1, \ldots x_n)$ are $x \oplus y = min(x+y, 2)$ and $x \otimes y = min(xy, 2)$. The adjacency matrices for N_n are:

$$M_a = \begin{pmatrix} 1 & 0 & 0\,0 \ldots & 0 \\ 0 & 0 & 1\,0 \ldots & 0 \\ 0 & 0 & 0\,1 \ldots & 0 \\ & \ldots & & \\ 0 & 0 & 0\,0 \ldots & 1 \\ 1 & 0 & 0\,0 \ldots & 0 \end{pmatrix} , \qquad M_b = \begin{pmatrix} 0 & 1 & 0\,0 \ldots & 0 \\ 0 & 0 & 1\,0 \ldots & 0 \\ 0 & 0 & 0\,1 \ldots & 0 \\ & \ldots & & \\ 0 & 0 & 0\,0 \ldots & 1 \\ 1 & 0 & 0\,0 \ldots & 0 \end{pmatrix} .$$

The following facts about state transitions will be useful later:

$$\langle 0, (x_1, \ldots, x_n)\rangle \xrightarrow{a} \langle 0, (x_1 \oplus x_n, 0, x_2, \ldots, x_{n-1})\rangle \ ,$$
$$\langle i, (x_1, \ldots, x_n)\rangle \xrightarrow{a} \langle i+1 \mod n, (x_1 \oplus x_n, 0, x_2, \ldots, x_{n-1})\rangle, \forall i \neq 0 \ ,$$

$$\langle j, (x_1, \ldots, x_n)\rangle \xrightarrow{b} \langle j+1 \mod n, (x_n, x_1, \ldots, x_{n-1})\rangle, \forall j \neq n-2 \ ,$$
$$\langle n-2, (x_1, \ldots, x_n)\rangle \xrightarrow{b} \langle n-1, (x_n \oplus 1, x_1, \ldots, x_{n-1})\rangle \ ,$$

$$\langle 0, (x_1, \ldots, x_n)\rangle \xrightarrow{b^n} \langle 0, (x_1, x_2 \oplus 1, x_3, \ldots, x_n)\rangle \ ,$$

$$\langle 1,(x_1,\ldots,x_n)\rangle \xrightarrow{b^n} \langle 1,(x_1,x_2,x_3\oplus 1,x_4,\ldots,x_n)\rangle \ ,$$
$$\ldots$$
$$\langle j,(x_1,\ldots,x_n)\rangle \xrightarrow{b^n} \langle j,(x_1,x_2,\ldots,x_{j+2}\oplus 1,\ldots,x_n)\rangle, \forall j\neq n-1 \ ,$$
$$\ldots$$
$$\langle n-2,(x_1,\ldots,x_n)\rangle \xrightarrow{b^n} \langle n-2,(x_1,x_2,\ldots,x_n\oplus 1)\rangle \ ,$$
$$\langle n-1,(x_1,\ldots,x_n)\rangle \xrightarrow{b^n} \langle n-1,(x_1\oplus 1,x_2,\ldots,x_n)\rangle \ .$$

I. Reachability

We show that any state $\langle i,(x_1,\ldots,x_n)\rangle)$ of our DFA is reachable from the initial state $\langle 0,(0,\ldots,0)\rangle$.The following computation proves it:

$$\langle 0,(0,\ldots,0)\rangle \xrightarrow{b^{nx_1}} \langle 0,(0,x_1,0\ldots,0)\rangle \xrightarrow{a}$$
$$\xrightarrow{a} \langle 0,(0,0,x_1,0,\ldots 0)\rangle \xrightarrow{b^{nx_n}} \langle 0,(0,x_n,x_1,0,\ldots 0)\rangle \xrightarrow{a}$$
$$\xrightarrow{a} \langle 0,(0,0,x_n,x_1,0,\ldots 0)\rangle \xrightarrow{b^{nx_{n-1}}} \langle 0,(0,x_{n-1},x_n,x_1,0,\ldots 0)\rangle \xrightarrow{a}$$
$$\ldots$$
$$\xrightarrow{a} \langle 0,(0,0,x_4,\ldots,x_n,x_1)\rangle \xrightarrow{b^{nx_3}} \langle 0,(0,x_3,\ldots,x_n,x_1,)\rangle \xrightarrow{a}$$
$$\xrightarrow{a} \langle 0,(x_1,0,x_3,\ldots,x_n)\rangle \xrightarrow{b^{nx_2}} \langle 0,(x_1,x_2,\ldots,x_n)\rangle$$

Thus, from the initial state we can reach an arbitrary state $\langle 0,(x_1,\ldots,x_n)\rangle$. For reaching $\langle i,(x_1,\ldots,x_n)\rangle$, with $i<n-1$, we first reach $\langle 0,(x_{i+1},\ldots,x_n,x_1\ldots x_i)\rangle$ and then we apply the word b^i: $\langle 0,(x_{i+1},\ldots,x_n,x_1\ldots x_i)\rangle \xrightarrow{b^i} \langle i,(x_1,\ldots x_n)\rangle$. For reaching $\langle n-1,(x_1,\ldots,x_n)\rangle$, with $x_1>0$ (recall that x_1 can not be 0 in this case), we first reach $\langle 0,(x_n,x_1-1,\ldots,x_{n-1})\rangle$, then we apply b^{n-2} reaching $\langle n-2,(x_2,\ldots,x_n,x_1-1)\rangle$, and then we apply b one more time.

II. Non-mergibility

We prove that no two distinct states $\langle i,(x_1,\ldots,x_n)\rangle$ and $\langle j,(y_1,\ldots,y_n)\rangle$ are mergible, by finding a word which maps one of these states into a final state and maps the other into a non-final state. Incidentally, it becomes apparent that our DFA has no sink state.

(case 1: $i\neq j$) We choose the word $a^{n-i}b^na^{n-2}$, and obtain the following computations:

$$\langle i,(x_1,\ldots x_n)\rangle \xrightarrow{a^{n-i-1}} \langle n-1,(1\oplus x_1\oplus\textstyle\sum_{j=i+2}^{n}x_j,0,\ldots 0,x_2,\ldots,x_{i+1})\rangle \xrightarrow{a}$$
$$\xrightarrow{a} \langle 0,(1\oplus x_1\oplus\textstyle\sum_{j=i+1}^{n}x_j,0,\ldots,0,x_2,\ldots x_i)\rangle \xrightarrow{b^n}$$

$$\xrightarrow{b^n} \langle 0, (1 \oplus x_1 \oplus \sum_{j=i+1}^{n} x_j, 1, 0, \ldots, 0, x_2, \ldots x_i)\rangle \xrightarrow{a^{n-2}}$$
$$\xrightarrow{a^{n-2}} \langle n-2, (1 \oplus \sum_{j=1}^{n} x_j, 0, \ldots, 0, 1)\rangle \ ,$$

thus we reach a final state. For the other state we have the following computation:

$$\langle j, (y_1, \ldots y_n)\rangle \xrightarrow{a^{n-i}} \langle j+n-i \mod n, (\textstyle\sum(-), 0, \ldots, 0, y_2, \ldots y_i)\rangle \xrightarrow{b^n}$$
$$\xrightarrow{b^n} \langle j+n-i \mod n, (\textstyle\sum(-), 0, \ldots, 1, \ldots, y_2, \ldots y_i)\rangle \xrightarrow{a^{n-2}} \qquad (*)$$
$$\xrightarrow{a^{n-2}} \langle j+n-i \mod n, (\textstyle\sum(-), 0, \ldots, 1, \ldots, 0)\rangle$$

and a final state is not reached. In step marked $(*)$, 1 appears at position $j+n-i+2 \mod n$, and on the second position we necessarily have 0, since $j \neq i$ (hence $2 \neq j+n-i \mod n$). By the end of the computation, the value 0 on the second position migrates to the end of the vector; thus the computation fails. Recall that $n \geq 3$ thus $n-2 \geq 1$ and the last step has at least one transition.

(case 2: $i = j$) Since the states are distinct, there exists a position k such that $x_k \neq y_k$. Without loss of generality we may assume that $x_k < y_k$ (otherwise we flip the states). We distinguish the following subcases:

1. ($x_k = 1$ or $y_k = 1$) For $x_k = 1$ we use the word b^{n-k} (recall that $x_k \neq y_k$), and for $y_k = 1$ we flip the states.

2. ($x_k = 0, y_k = 2$) We distinguish two situations. If $k = i+2$ we have the following computations:

   $$\langle i, (x_1, \ldots x_n)\rangle \xrightarrow{b^{n-i-1}} \langle n-1, (1 = x_k \oplus 1, x_{k+1}, \ldots, x_{k-1})\rangle \xrightarrow{b^{n-1}}$$
   $$\xrightarrow{b^{n-1}} \langle n-2, (x_{k+1}, \ldots x_n, x_1, \ldots, 1)\rangle \ ,$$

   which ends in a final state, whereas the same word maps the state $\langle i, (y_1, \ldots y_n)\rangle$ into a non-final state, for $y_k \oplus 1 = 2$. Thus the word b^{2n-i-2} solves this case. If $k \neq i+2$ we have the following generic computation:

   $$\langle i, (x_1, \ldots x_n)\rangle \xrightarrow{b^{n-k+2}} \langle *, (x_{k-1}, x_k, \ldots x_n, x_1, \ldots, x_{k-2}))\rangle \xrightarrow{a^{n-2}}$$
   $$\xrightarrow{a^{n-2}} \langle *, (z, 0, \ldots, 0, x_k)\rangle \xrightarrow{b}$$
   $$\xrightarrow{b} \langle *, (x_k, z, 0, \ldots, 0)\rangle \xrightarrow{a}$$
   $$\xrightarrow{a} \langle t, (x_k, 0, z, 0, \ldots, 0)\rangle \ ,$$

with some $t \in \{0,\dots,n-1\}$ and $z \in \{0,1,2\}$. From here, there exists a word $w = a^r$ which continues the computation up to $\langle *, (1 = x_k \oplus 1, 0, \dots 0, z, 0, \dots, 0)\rangle$, and after that, the word b^{n-1} leads to the state $\langle *, (0, \dots 0, z, 0, \dots, 0, 1)\rangle$. Thus, the word $b^{n-k+2}a^{n-2}bawb^{n-1}$ maps $\langle i, (x_1, \dots, x_n)\rangle$ to a final state, however, this is not true for $\langle i, (y_1, \dots, y_n)\rangle$.

We have proven the non-mergibility as well, thus the DFA is minimal, and its size is exactly the upper bound for unique square. □

Remark 2. This combinatorial proof does not work for $n = 2$. We were expecting this, for we verified experimentally that the upper bound is not reached for 2-state DFAs.

We can also prove the following exponential lower bound for the nondeterministic state complexity of unique concatenation.

Proposition 6. *There exists a pair of NFA's M_1 and M_2 with $O(k)$ states combined, such that $L(M_1)L(M_2)$ is accepted by an $O(k)$ state NFA, but any NFA accepting $L(M_1) \circ L(M_2)$ has at least 2^k states.*

Proof. Take $L(M_1) = (0+1)^*0(0+1)^k1$ and $L(M_2) = (0+1)^*$. Then $L(M_1)L(M_2)$ is accepted by an $O(k)$ state NFA, but any NFA accepting $L(M_1) \circ L(M_2) = UL_k$ has at least 2^k states. □

Unique Concatenation: Empirical Results

`Experiment 1.` (description) We generate all minimal DFA with 3 states and perform unique concatenation on all pairs. There are 1028 distinct DFA, leading to 1056784 operations. Figure 2 provides a histogram of our results: the x-axis represents the size of the output DFA, and the y-axis plots the number of cases which resulted in DFAs of that size.

For two DFAs of size m and n, the theoretical upper bound is $m3^n - k3^{n-1}$ (k is the number of final states in the first DFA). The largest DFAs obtained in this experiment are of size 72, and are the result of operations where the first DFA has precisely one final state. Thus the bound is reached for $m = n = 3$ and $k = 1$. Notice that small DFAs have a higher incidence rate, hinting at the fact that *the worst-case scenarios are sparse*.

Quite interestingly, the upper bound is not reached for $m = n = 2$: we gave a possible explanation for this in Remark 2. The histogram of unique

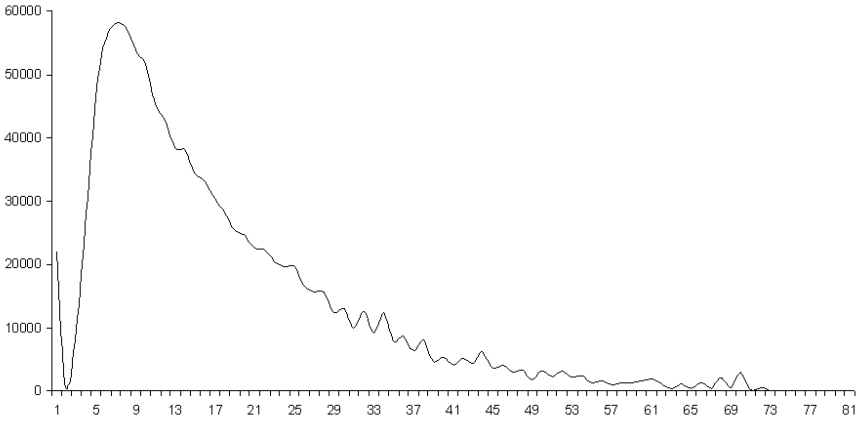

Fig. 2. Histogram for unique concatenation over 3-state minimal DFA.

concatenation in this case is shown in Figure 3. Notice that no operation reaches 15 states (14 is the largest value reached). Moreover, for the unique square in this case, the largest reached size is 12.

`Experiment 2.` (description) Initially we investigated whether the unique square operation has a smaller state complexity. For this we performed this operation for all minimal DFA of size 3 and 4. The results (histograms) are shown in Figure 4 and Figure 5. We found 6 minimal DFA of size 3 (with one final state) whose unique square reaches the upper bound of 72.

`Experiment 3.` (description) We also questioned whether the worst-case scenarios for the standard concatenation are also worst for the unique concatenation. Thus, we took 16 pairs of DFA which have been proven in [22] to reach the upper bound for concatenation (for $m,n \in \{2,3,4,5\}$) and performed their unique concatenation. In the following table, the numbers in parentheses are the upper bounds for unique concatenation, and the other numbers are the results obtained in our experiment.

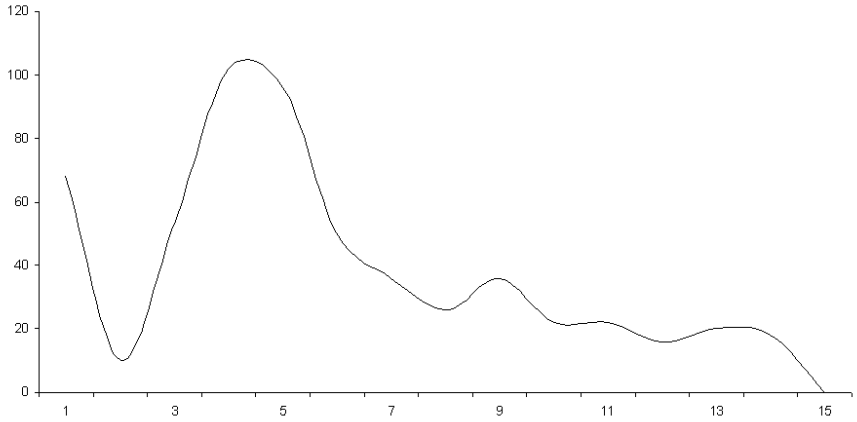

Fig. 3. Histogram for unique concatenation over 2-state minimal DFA.

$m\backslash n$	2	3	4	5
2	9 (15)	29 (45)	88 (135)	267 (405)
3	15 (24)	61 (72)	158 (216)	565 (648)
4	24 (33)	69 (99)	267 (297)	807 (891)
5	27 (42)	113 (126)	283 (378)	1049 (1134)

We observe that none of these examples are reaching upper bounds for unique concatenation, thus it is not necessary that the worst-cases for concatenation are worst for unique concatenation as well. We will see later that the reciprocal may be true: worst-case examples for unique concatenation may be worst for concatenation as well.

`Candidates for a generic example.` Consider the two parameterized minimal DFA: J_i and N_i, with $i \geq 3$, as shown in Figure 6. Our experiments show that the upper bound is reached for any of the following combinations: $L(J_i)^{\circ 2}$, $L(N_i)^{\circ 2}$, $L(J_i) \circ L(J_j)$, $L(N_i) \circ L(N_j)$, $L(J_i) \circ L(N_j)$, with i, j arbitrary integers greater than 2. It is interesting to notice that J_i is given in [23] as example for reaching the upper bound for the normal concatenation, hence it may provide an example where worst-case is achieved for both concatenation and unique concatenation.

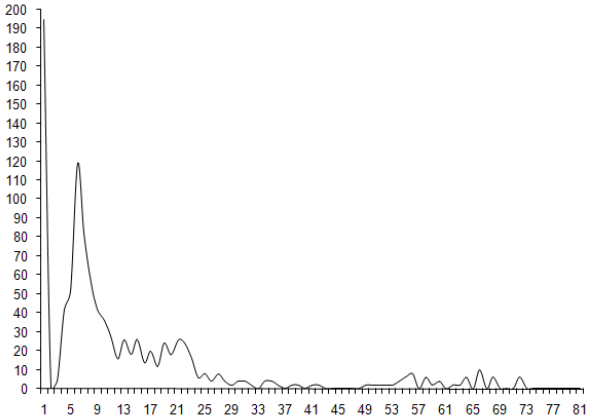

Fig. 4. Histogram for unique square over 3-state minimal DFA.

4.3 Unique Star

As before, we start with a naive approach. Let A be a DFA for a regular language L, of size m, and B be the NFA accepting L^+, obtained by the standard construction without employing ε (B has $m+1$ states). Recall that ε is a by-product of unique star and that poly star cannot produce it. An NFA for $L^\diamond$ works as follows. We start by simulating A. When we nondeterministically hit a final state of B (i.e., we may not stop at the first hit), we start simulating two copies of B in parallel (cross product), one continuing the initial computation, the other starting from the initial state. The input is accepted when both simulations accept. This NFA has $m+1$ state in the first module and m^2 in the second. Thus we have m^2+m+1 states NFA which implements the poly star (it accepts all those words which have more than one factorization into words in L). Since $L^\circ = L^* \setminus L^\diamond$, we obtain a first upper bound for the unique star, of $2^{m^2+m+1}(2^{m-1}+2^{m-k-1})$, where k is the number of final states of A which are not initial.

By using a technique similar to that for unique concatenation we can substantially improve this upper bound:

Theorem 3. *If $L \setminus \{\varepsilon\}$ is accepted by a DFA A of size m and with k final states, then a DFA for L° has at most $3^{m-1}+(k+2)3^{m-k-1}-(2^{m-1}+2^{m-k-1}-2)$.*

(this upper bound has been reached for $k=1$ and $m=2,\ldots,8$ by the generic examples in Figure 11 – thus we conjecture that it is sharp)

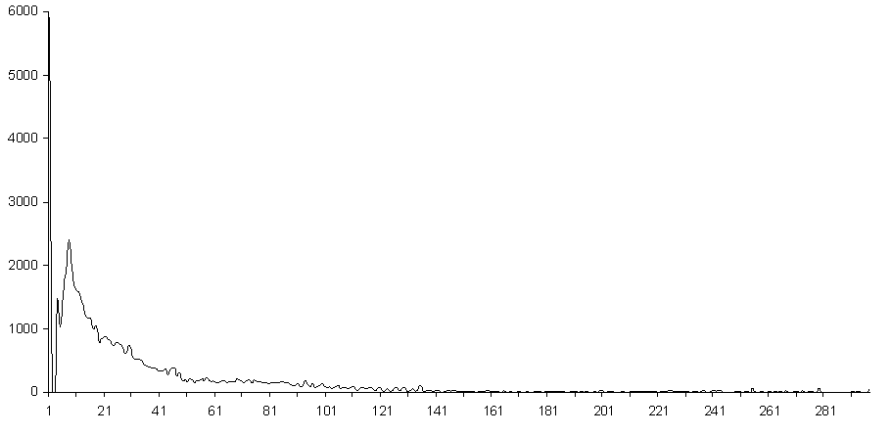

Fig. 5. Histogram for unique square over 4-state minimal DFA.

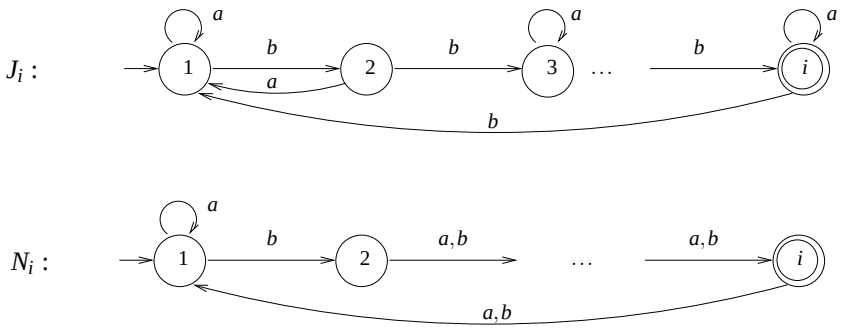

Fig. 6. Parameterized automata J_i and N_i.

Proof. Let $A = (Q_A = \{1,2,\ldots,m\}, \Sigma, \delta_A, 1, F_A)$ be a DFA for L, of size m, and $F_A = \{m-k+1,\ldots,m\}$. By M_a we denote the adjacency matrix of A with respect to the symbol $a \in \Sigma$. Thus $M_a[i,j]$ is 1 if there is a transition from i to j labeled with a, and 0 otherwise.

Denote, as before, $\oplus$ and $\otimes$ to be two operations given by $a \oplus b = min(a+b,2)$ and $a \otimes b = min(a \cdot b,2)$. We define a DFA $B = (Q_B, \Sigma, \delta_B, 0, F_B)$ for L° as follows:

1. $Q_B = V \cup \{0\}$, where 0 is the initial state of B and V is the set of all vectors with m components holding values in $\{0,1,2\}$. The vector entries are indexed from 1 to m.
2. The transition function is defined as follows:

(a) $\delta_B(0,a) = v_a$, where $v_a[\delta_A(1,a)] = 1$, $v_a[1] = 1$ if $\delta_A(1,a) \in F_A$, and $v_a[i] = 0$ for all other indices i.
(b) Denote $S_k[v]$ to be the value $v[m-k+1] \oplus \cdots \oplus v[m]$. For all $j \in \{1,\ldots,m\}$ and $a \in \Sigma$ we set $\delta_B(v,a) = v' + v''$, where $v' = v \otimes M_a$ and $v'' = (S_k(v'),0,0,\ldots,0)$.

3. $F_B = \{v \in V \mid S_k(v) = 1\} \cup \{0\}$.

We use vectors to store the number of computations in A, from the initial state to every state: 0, 1, or 2 (2 = more than one computation). If a vector v is reached during the computation of B, the value $S_k(v)$ gives the number of different computations in A reaching final states. This number has been added to the first component of v, meaning that reaching a final state in A implies reaching its initial state as well, for we aim at accepting words in L^*. If a word w "reaches" a state-vector v in B, then v[i] gives the number (0,1, or 2) of distinct paths in A, labeled with w, from the initial state of A to its state i, when A is modified to accept L^* in the standard way. By setting as final state in B all those vectors which denote exactly one successful such path, we force B accept exactly the words in L°.

It now remains to compute how many states can possibly be reached in B. First, B will have an initial state and, eventually, a sink state. We make two crucial observations: (a) for a reachable state $v \in V$ we must have $v[1] \geq S_k(v)$, and (b) any reachable state $v \in V$ containing only values 0 and 2 is mergible into (or, equivalent to) the sink state. Indeed, the first observation is justified by the fact that a reachable state v accumulates in $v[1]$ the value $S_k(v)$, according to δ_B. For the second observation we notice that if v contains only 0's and 2's, then $\delta_B(v,a)$ will have the same property. Moreover, such state cannot be final. We are now ready to compute the maximum number of reachable states that B can have, *after an eventual minimization*:

1. There is an initial state 0 and eventually a sink state, amounting for 2 states, to begin with.
2. At most $3^{m-k}-1$ vectors v with $S_k(v) = 0$ can be reached. Indeed they are 3^{m-k} such vectors; however the null vector cannot be reached. From these vectors, we subtract those having only 0's and 2's, for they will eventually be merged together within a sink state when B is minimized. There are $2^{m-k}-1$ such vectors, without counting the null vector. Thus, we have altogether $3^{m-k}-2^{m-k}$ states in this case.
3. At most $2k \cdot 3^{m-k-1}$ states v with $S_k(v) = 1$ can be reached. Observe that once $S_k(v) = 1$ we can not have $v[1] = 0$ since $S_k(v)$ has been added to $v[1]$ during an eventual transition. Thus, $v[1]$ can take two values (1 and 2), then the portion of the vector $v[2,\ldots,m-k]$ gives 3^{m-k-1} combinations,

and there are at most k combinations of $v[m-k+1,\ldots,m]$ which ensure $S_k(v)=1$.

4. Finally, at most $3^{m-k-1}(3^k-k-1)$ states v with $S_k(v)=2$ can be reached. Indeed, we have at most 3^k-k-1 combinations in $v[m-k+1,\ldots,m]$ which ensure $S_k(v)=2$. Then $v[1]$ must be 2 (since $S_k(v)$ has been added to it), and there are 3^{m-k-1} combinations for $v[2\ldots m-k]$. However, some of these vectors are surely mergible into the sink state: those with only $0's$ and 2's. There are exactly $2^{m-k-1}(2^k-1)$ such vectors v, since: $v[1]=2$, there are (2^k-1) combinations in $v[m-k+1,\ldots,m]$ (this portion cannot be all 0's), and there are 2^{m-k-1} combinations of 0's and 2's in $v[2,\ldots,m-k]$. Combining all these numbers, we obtain $\underline{3^{m-k-1}(3^k-k-1)-2^{m-k-1}(2^k-1)\text{ states}}$ in this case.

We reach the conclusion by adding up the figures underlined in the above cases 1-4. □

The case when $\varepsilon \in L$ and we are given a DFA for L is proven similarly, and may lead to a slightly different upper bound. As a matter of fact, we can immediately derive an upper bound by noticing that a DFA for $L\setminus\{\varepsilon\}$ has at most one state more than the DFA for L (thus, we just replace m by $m+1$ in the above result). Nevertheless, a proof as in Theorem 3 may improve such upper bound, and it merely involves a different state-indexing scheme. We leave this exercise to the reader.

Unique Star: Empirical Results

Our experiments show that this upper bound is very likely sharp, in both n and k. In Figures 7, 8, 9 and 10 one can find the histograms for $n=2,\ldots,5$. For $n=5$ we only tested all minimal DFAs with one non-initial final state. The upper bound has always been reached, and it has also been reached by the DFAs in Figure 11 for $n=2,\ldots,8$ – thus they are good candidates for the worst-case in general.

5 Decision Problems

In this section, we consider decision problems involving unireg expressions, namely the membership and the non-emptiness problems. We start with the membership problem.

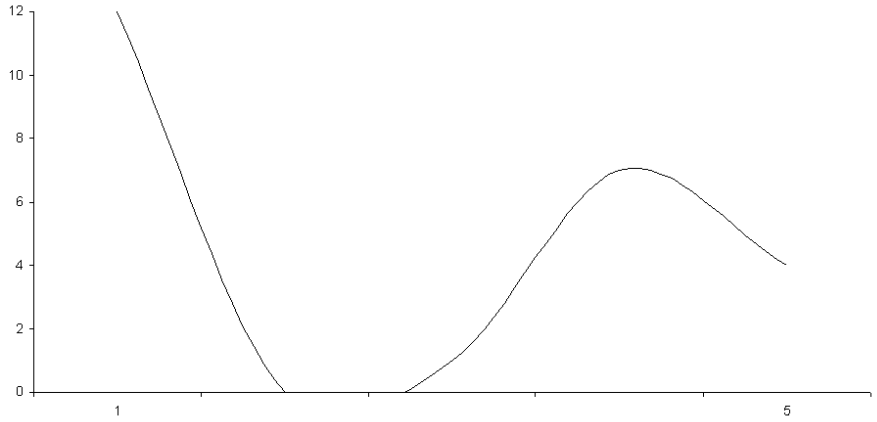

Fig. 7. Histogram for unique star over 2-state minimal DFA.

Theorem 4. *The membership problem for unireg expressions is in P.*

We provide two polynomial algorithms for this problem, one based on the Glushkov automaton for a regular expression, and the other based on the expression-tree for a regular expression and using a dynamic programming technique.

Algorithm 1

Let R be a unireg expression over Σ, and w be a string in Σ^*. We first tackle the special case when $w = \varepsilon$. In this case, we can determine efficiently the membership, by consulting the parse-tree for R. If $w \neq \varepsilon$, we proceed as follows. Let $w = a_1 a_2 \ldots a_k$ and let R' be the regular expression obtained from R by replacing the unique operations with the corresponding regular operations. We use the algorithm of Glushkov to obtain an ε-free NFA M such that the set of strings accepted by M is the same as the set of strings generated by R' (with the possible exception of ε). It is known [13] that Glushkov's algorithm preserves the degree of ambiguity of representation, that is, the number of accepting computations in M for an input word w equals the number of ways in which R' generates w. Then, $L(R)$ consists of those words which are accepted by M in an unique computation or, we say, unambiguously. Thus, it now suffices to detect whether our word w is accepted unambiguously by M. We consider that the states in M are numbered from 0 to $m-1$, and we define a set $\{T_a\}_{a\in\Sigma}$ of square matrices, by setting $T_a[i, j]$ to be the number of transitions

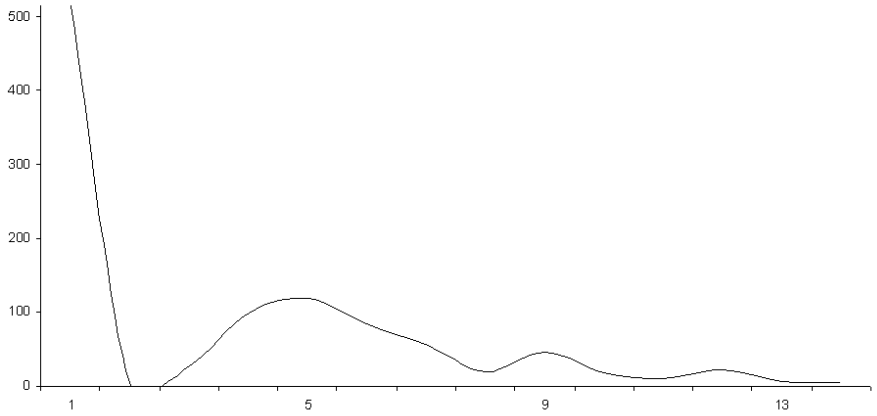

Fig. 8. Histogram for unique star over 3-state minimal DFA.

in M from i to j and labelled with a. Thus, T_a's are $m \times m$ matrices. Let $S = [s(0), s(1), ..., s(m-1)]$ where $s(i) = 1$ if i is the start state, 0 otherwise. Similarly, let $F = [f(0), f(1), ..., f(m-1)]$ where $f(j) = 1$ if j is an accepting state of M, 0 otherwise. Then, it is easy to check that $ST_{a_1}T_{a_2}...T_{a_k}F$ is the number of accepting paths for the string w in M. By computing the above matrix chain product, we can determine the number of accepting paths for w. If this number is 1, then w is accepted; otherwise it is rejected. It is clear that this algorithm runs in time polynomial in $|R|+|w|$, where by $|R|$ we denote the number of symbols in R.

Algorithm 2

Another algorithm for deciding membership for an unireg expression is a slight modification of the well-known dynamic programming algorithm on the expression tree of a regular expression. This algorithm will first construct the expression tree of our unireg expression. Suppose the input word is w of length n. We index the positions of symbols in w from 1 to n. To each node X of the expression tree, we associate an $n \times n$ matrix M_X with values in $\{0,1,2\}$. The cell $M_X[i,j]$ will store information about the subword $w_{i,j}$ of w starting at position i and ending at position j: a value of 0 represents no match of $w_{i,j}$ with the node X, a value of 1 represents exactly one match and a value of 2 represents two or more matches. The matrices are computed in a

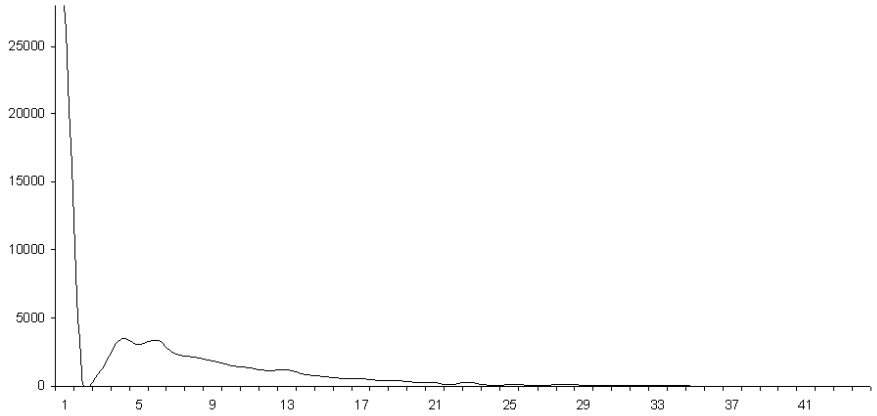

Fig. 9. Histogram for unique star over 4-state minimal DFA.

dynamic-programming style, similar to the classical algorithm. The algorithm answers `YES` if $M_R[1,n] = 1$, where R is the root of the tree, and `NO` otherwise.

Theorem 5. *The non-emptiness problem for unireg expressions is in PSPACE.*

Proof. Let R be a unireg expression over Σ and R' be the regular expression obtained from R by replacing the unique operations with the standard regular operations. Let M be the NFA obtained by applying Glushkov's algorithm to R'. Then $L(R)$ is non-empty if and only if there exists a word w accepted unambiguously by M. We give a polynomial space algorithm to test for the existence of such a w. By Savitch's theorem [24], it suffices to give a non-deterministic algorithm.

For $a \in \Sigma$, let B_a denote the adjacency matrix of M with respect to the input a. By the state complexity result of Lemma 4, if there is a word w accepted unambiguously by M, there is such a w of length at most 3^n, where n is the number of states of M. We thus non-deterministically guess such a word $w = w_1 w_2 \dots w_r$, $r \leq 3^n$, symbol by symbol, and we compute the matrix product $B_{w_1} B_{w_2} \dots B_{w_r} = B$, reusing space after each matrix multiplication. Here the matrix multiplication is again done with $\oplus$ and $\otimes$ as the component-wise operations. We maintain an $O(n)$ bit counter to keep track of the length of the guessed string w. We verify that M accepts w unambiguously by looking at the row of B corresponding to the start state of M and summing the entries in the columns corresponding to the final states of M. This quantity is exactly 1 if and only if M accepts w unambiguously.

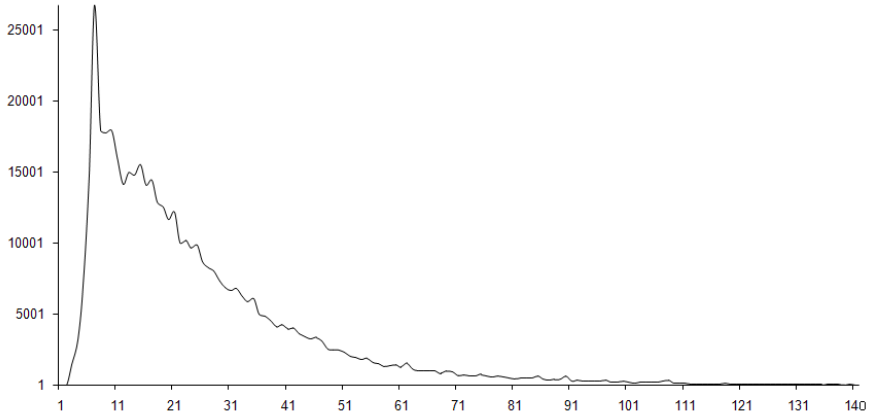

Fig. 10. Histogram for unique star over 5-state minimal DFA with $k = 1$.

The transformation of R to R' to M can be done in polynomial space, and the non-deterministic algorithm described above uses only polynomial space. It follows that the non-emptiness problem can be solved in polynomial space. □

6 Application: 2-DFA with a Pebble

Informally, a pebble 2-DFA is a 2-DFA that has a marker in the finite control. At any step, depending on the current state and the input symbol scanned, the pebble can be placed on a tape square. Also, the next move of the automaton is a function of the presence of pebble on the current square scanned, in addition to the current state and the current symbol read on the tape. Also, if the pebble is in the current square, based on the current state and the current symbol scanned, the finite control has the option to take back the pebble that can then be placed on another square, etc. The acceptance policy is like in a 2-DFA: when it reaches a configuration for which there is no next move, if the state reached is an accepting (rejecting) state, the input is said to be accepted (rejected).

It is well-known that if M is a 2-DFA with a pebble, $L(M)$ is regular. Here we consider the following question: *What is the worst-case blow-up in the number of states when a 2-DFA with a pebble is converted into a 1-DFA?* Let $f(n)$ denote this function. Formally, f is defined by the following two conditions: (1) there is an n-state 2-DFA with a pebble such that the minimum equivalent 1-DFA has $f(n)$ states, and (2) for any n-state 2-DFA with a pebble, there is an equivalent 1-DFA with at most $f(n)$ states. We show that a good lower-bound on $f(n)$ can be obtained from the results of the previous section. The connection

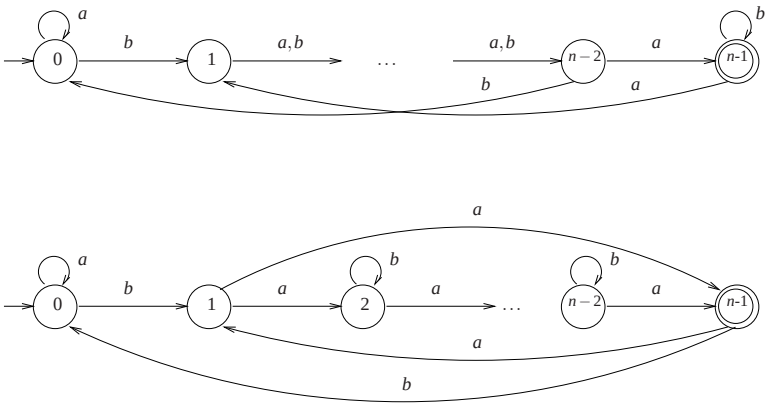

Fig. 11. Worst-case candidates for unique star.

between the state complexity for converting 2-DFA with a pebble to a 1-DFA and the state complexity of unique concatenation is provided by the following lemma.

Proposition 7. *Let A and B be two DFA's with m and n states, respectively. There exists a 2-DFA C with a pebble such that $L(C) = L(A) \circ L(B)$ and C has $2(m+n)+2$ states.*

Proof. We will informally describe the operation of C. The state set of C is given by $Q_C = Q_A \cup Q_B \cup Q'_A \cup Q'_B$, where $Q'_A = \{q' \mid q \in Q_A\}$ and $Q'_B = \{q' \mid q \in Q_B\} \cup \{r, r'\}$. On an input string $\$w\#$, C starts with the reading head on the left end-marker, in its start state s_C — which is, by definition, s_A. The head moves to the right, and C simulates A until an accepting state is reached. At this point, C places a pebble on the current tape square, enters s_B and moves to the right simulating B, until the right end-marker is reached. If at this point an accepting state of B is not reached, then C proceeds as in Step 1, else it proceeds as in Step 2, detailed as follows:

1. Step 1: C enters the state r and makes a right-to-left sweep until it reaches the left end-marker, and enters the state s_A. Then it simulates A as usual, with the difference that when it reaches the pebble, it picks it up and continues the computation to the right till another final state of A is reached. Then it drops the pebble and continues with the simulation of B, as in the initial phase.

2. Step 2: C enters the state r' and makes a right-to-left sweep until it reaches the left end-marker and enters state s'_A. While in a state of the form $q'_a \in Q'_A$,

C simulates A, but uses the primed states and keeps moving to the right. More precisely, if $\delta_A(q_a, b) = q_d$, then C, on input b and in state q'_a, changes its current state to q'_d and moves to the right. It continues this phase until the square with a pebble is detected. At this point, it picks up the pebble, moves to the right continuing the simulation of A using the primed states. At this point there are two cases to consider. (a) As the simulation of A continues, the right end-marker is reached without ever reaching an accepting state of A. In this case, C accepts the input and it halts. (b) An accepting state of A is reached before the right end-marker is reached. When an accepting state of A is reached for the first time, the pebble is dropped on the square that forms the last symbol that caused A to reach the accepting state, C enters the state s'_B and it starts the simulation of B using the primed states. The simulation continues until the right end-marker is reached. At this point, if an accepting state of B is reached, C rejects the input and halts. If an accepting state is not reached, then C enters the state r' and repeats Step 2.

It is clear that C accepts $L(A) \circ L(B)$ and the proof is complete. □

7 Conclusion and Further Work

In this paper we studied unique rational operations and their state complexity. We drew connections between the so-called unireg expressions and unambiguous regular expressions, and we studied the closure of DCF, CF and linear CF languages with respect to unique union and unique concatenation. We obtained a sharp bound of the state complexity for unique union, comparable with that of "plain" union. For unique concatenation we gave a state complexity upper bound which we strongly believe to be sharp, for we provided generic (parameterized) examples that reached the upper bound in all our extensive experiments. For the unique square (unique concatenation of a language with itself), we provided sharp upper bounds and a generic worst-case example, in the laborious proof of Lemma 6. Both bounds are significantly higher than those for the plain concatenation. For the nondeterministic state complexity of unique concatenation we provided an exponential lower bound. In Theorem 3 we provided a curious upper bound for the unique star, that we believe, yet again, to be sharp, for the generic DFAs in Figure 11 have been empirically proven to be worst-case scenarios for this unique operation. Finally, we studied the complexity of the membership and non-emptiness problem for unireg expressions, and we drew a connection between 2-DFA with a pebble and unique concatenation, which we believe that may be extended to unique star as well.

Several problems remain to be dealt with in the future. In the following we give a list which is by no means exhaustive.

- In addition to Lemmas 1 and 3, we conjecture and leave for further work the non-closure of CF languages to the unique star.
- In Lemma 4 we gave an upper bound for the state complexity of the language of all words accepted unambiguously by an NFA. It remains to show that this bound is sharp.
- It is worth investigating the generic conjecture at page 10, which states a connection between the state complexities of unique and poly operations.
- Theorem 2 provides an upper bound for the state complexity of the unique concatenation. It remains to prove that the bound is sharp. We have given generic examples which reached the upper bound in all our experiments and it remains to show that indeed they are a general worst-case.
- At page 19 we gave candidates for worst-case scenarios for unique concatenation and unique square. It remains to prove theoretically that they reach the given upper bound (we have already done it for $L(N_i)^{\circ 2}$).
- The upper bound in Theorem 3 remains to be proven sharp, by proving that the example in Figure 11 is a worst-case example for this operation. There also remains to modify the proof for the case where $\varepsilon \in L$ and obtain a corresponding upper bound.
- The emptiness and equivalence problems for unireg expressions have not been dealt with yet.
- We haven't studied the operation of unique shuffle, $\mathbin{\mathring{\sqcup\!\sqcup}}$, which remains for further work.
- Last but not the least, we propose a study on `bounded unique operations`. For example, the k-unique concatenation of two languages would be denoted by $L_1 \circ_k L_2$, and $w \in L_1 \circ_k L_2$ iff w can be factorized in no more than k different ways as $w = uv$ with $u \in L_1$ and $v \in L_2$. We anticipate that the state complexities of these operations will extend naturally those stated in this paper (possibly, by replacing the constant 3 in the present bounds with k – for this operation).

8 Acknowledgements

This study benefitted greatly from our extensive experiments, and it could have not been possible without the use of Grail++, carefully developed and maintained by Derick Wood, Sheng Yu, and its project members for over 20 years.

References

1. Karttunen, L.: Applications of Finite-State Transducers in Natural Languages Processing. In: Proc. CIAA 2000. Volume 2088 of LNCS. (2000) 34–46

2. Bochmann, G.V.: Submodule Construction and Supervisory Control: a Generalization. In: Proc. CIAA 2001. Volume 2494 of LNCS. (2001) 27–39
3. Harel, D., Politi, M.: Modeling Reactive Systems with Statecharts. McGraw-Hill (1998)
4. Yu, S., Zhuang, Q., Salomaa, K.: The State Complexities of Some Basic Operations on Regular Languages. Theoretical Computer Science **125** (1994) 315–328
5. Campeanu, C., Culik, K., Salomaa, K., Yu, S.: State Complexity of Basic Operations on Finite Languages. In: Proc. WIA 1999. Volume 2214 of LNCS. (1999) 60–70
6. Campeanu, C., Salomaa, K., Yu, S.: Tight Lower Bound for the State Complexity of Shuffle of Regular Langauges. Journal of Automata, Languages and Combinatorics **7** (2002) 303–310
7. Salomaa, A., Wood, D., Yu, S.: On the State Complexity of Reversals of Regular Languages. Theoretical Computer Science **320** (2004) 293–313
8. Y, G., Salomaa, K., Yu, S.: State Complexity of Catenation and Reversal Combined with Star. In: Proc. DCFS 2006. Volume 2494. (2006) 153–164
9. Salomaa, A.S.K., Yu, S.: State Complexity of Combined Operations. Theoretical Computer Science **383** (2007) 140–152
10. Yu, S.: State Complexity: Recent Results and Open Problems. Fundamenta Informaticae **64** (2005) 471–480
11. Yu, S.: Regular Languages. *In [25]*, **Ch.1** (1997) 41–110
12. Brüggemann-Klein, A.: Regular Expressions into Finite Automata. Theoretical Computer Science **120**(2) (1993) 197–213
13. Book, R., Even, S., Greibach, S., Ott, G.: Ambiguity in Graphs and Expressions. IEEE Transactions on Computers **C-20**(2) (1971) 149–153
14. Mandel, A., Simon, I.: On Finite Semigroups of Matrices. Theoret. Comput. Sci. **5** (1977) 101–111
15. Ravikumar, B., Ibarra, O.: Relating the Type of Ambiguity of Finite Automata to the Succinctness of their Representation. SIAM J. Comput. **18** (1989) 1263–1282
16. Holzer, M., Kutrib, M.: State Complexity of Basic Operations of Nondeterministic Finite Automata. In: Proc. CIAA 2002. Volume 2608 of LNCS. (2003) 148–157
17. Hromkovič, J., Karhumäki, J., Klauck, H., Schnitger, G.: Communication Complexity Method for Measuring Nondeterminism in Finite Automata. Inform. Comput. **172** (2002) 202–217
18. Ellul, K., Krawetz, B., Shallit, J., Wang, M.: Regular Expressions: New Results and Open Problems. J. Autom. Lang. Comb. **9** (2004) 233–256
19. Birget, J.C.: Intersection and Union of Regular Languages and State Complexity. Inform. Process. Lett. **43** (1992) 185–190
20. Maslov, A.N.: Estimates of the Number of States of Finite Automata (Russian). Dokl. Akad. Nauk SSSR **194** (1970) 1266–1268 English translation in *Soviet Math. Dokl.* **11** (1970), 1373–1375.
21. Salomaa, K., Zhuang, Q., Yu, S.: The State Complexities of some Basic Operations on Regular Languages. Theoretical Computer Science **125** (1994) 315–328
22. Jirásková, G.: State Complexity of Some Operations on Binary Regular Languages. Theoretical Computer Science **330**(2) (2005) 287–298
23. Rampersad, N.: The State Complexity of L^2 and L^k. Information Processing Letters **98** (2006) 231–234
24. Savitch, W.: Relationship between Nondeterministic and Deterministic Tape Complexities. J. Comput. Syst. Sci. **4** (1970) 177–192
25. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages. Springer-Verlag, Berlin Heidelberg New York (1997)