On the Descriptional Complexity of Literal Shuffle

Guilherme Duarte[0000−0002−4119−0694]1, Nelma Moreira[0000−0003−0861−0105]1, Luca Prigioniero[0000−0001−7163−4965]2, and Rogério Reis[0000−0001−9668−0917]1 ⋆

[1] CMUP & DCC, Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre, 4169-007 Porto, Portugal.
{guilherme.duarte,nelma.moreira,rogerio.reis}@fc.up.pt
[2] Department of Computer Science, Loughborough University, Epinal Way, Loughborough LE11 3TU, United Kingdom
L.Prigioniero@lboro.ac.uk

Abstract. In this paper, we study the descriptional complexity of several variants of the shuffle operation on regular languages. In the perfect shuffle, words of equal length strictly alternate their symbols. If the words have different lengths, the initial literal shuffle performs the perfect shuffle while both words have symbols and then appends the remaining suffix of the longest word. Lastly, the literal shuffle is an extension of the previous operation, in which the initial shuffle may start at an arbitrary position in one of the two words. We analyze the number of states sufficient and necessary for nondeterministic and deterministic automata to recognize the resulting languages. For nondeterministic finite automata, the cost of the simulation is polynomial. In the worst case, a deterministic finite automaton requires exponentially many states to recognize the literal shuffle of two languages given also by deterministic finite automata, whereas a deterministic 1-limited automaton — namely, a one-tape Turing machine in which each cell may be rewritten only upon its first visit — requires only polynomial many states.

1 Introduction

The *shuffle* of two words u and v is the set of words obtained from interleaving the symbols u and v while preserving the internal order of each word. This operation extends naturally to languages, and the class of regular languages is known to be closed under shuffle. Berard [1] introduced several restricted variations on the shuffle operation. The *perfect shuffle* corresponds to the strict interleaving of symbols from the two words. The *initial literal shuffle* extends this operation to words of different lengths. The *general literal shuffle* further allows the initial literal shuffle to begin after consuming a prefix of one of the words. Unlike the general shuffle, these three restrictions are neither commutative nor

⋆ Research partially supported by CMUP through FCT under the project with DOI 10.54499/UID/00144/2025 and by FCT project AVATAR 2023.12169.PEX. Guilherme Duarte is also supported by the FCT grant 2024.01528.BD.

associative. All of them extend naturally to languages. Moreover, the closure properties of these operations were studied for standard language classes, in particular establishing closure for regular languages. The closure properties of the n-ary versions of the initial literal shuffle and the general literal shuffle were later investigated by Hoffman [8].

In this work, we consider languages represented not only by nondeterministic finite automata (NFAs) and deterministic finite automata (DFAs), but also deterministic 1-limited automata (1-LAs). Limited automata are Turing machines with restrictions on their rewritings and were introduced by Hibbard [7]. These machines operate on a single tape, which initially contains the input. At the left and at the right of the input there are two special end-markers $\rhd$ and $\lhd$, respectively. The machine is restricted to move its head only between them. In particular, it cannot move to the left of the cell containing $\rhd$ and to the right of the cell containing $\lhd$. Also, limited automata may overwrite the content of each cell only during its first d visits, for a fixed $d \geq 0$. We write d-limited automaton to make the parameter d explicit. For $d = 0$, no rewritings are allowed, hence these automata are standard two-way finite automata, thus they have the same expressive power as finite automata. For $d = 1$, the rewritings in each cell are restricted only to the first visit, and no additional expressive power is obtained [10]. For $d \geq 2$, nondeterministic d-limited automata recognize context-free languages.

In descriptional complexity of operations on languages, the goal is to determine the size of the target devices recognizing the languages resulting from an operation, as a function of the sizes of the source devices representing the input languages.

For the general shuffle, given two NFAs with m and n states, respectively, there exists an NFA with mn states that recognizes the shuffle of their languages [3]. In contrast, when the input automata are deterministic, every DFA recognizing the shuffle may require exponentially many states in the worst case. Domaratzki and Salomaa showed that a DFA with $2mn$ states suffices to recognize the perfect shuffle of the languages of two DFAs with m and n states, respectively, and that this bound is tight [4, 5].

In this work, we present constructions for automata recognizing the perfect shuffle, the initial literal shuffle, and the general literal shuffle of two languages. In the nondeterministic case the cost in size of the simulation is polynomial. When DFAs are used both as source and target devices, the cost of the simulation of the general literal shuffle is exponential. On the other hand, when taking 1-limited automata as target devices, the cost becomes polynomial.

We remark that the idea of considering *extensions* of finite automata — that do not increase the recognizing power of the model — to obtain polynomial-size simulations for operations on regular languages is not new. In these cases, the extended devices are used to avoid exponential blow-ups in size. For instance, deterministic 1-LAs have been used to obtain polynomial simulations for the concatenation and the star of languages accepted by DFAs [9], and nondeterministic 1-LAs have been used to simulate the complementation of two-way NFAs [6].

2 Preliminaries

In this section, we establish the notation used. Given two integers i, j with $i < j$, let $[i, j]$ denote the range from i to j, including both i and j, namely $\{i, i+1, \ldots, j\}$. Moreover, we shall omit the left bound if it is equal to 0, thus $[j] = \{0, 1, \ldots, j\}$. Given a set S, we denote its cardinality by $|S|$ and 2^S the set of all its subsets.

An *alphabet* Σ is a finite set of *symbols*. In this paper, we shall consider alphabets Σ as ordered sets, for some fixed total order $<$ on the symbols of Σ. A *word* is a sequence of symbols. A *language* is a set of words over an alphabet, and $\Sigma^\star$ denotes the language containing all words over Σ, including the empty-word ε. Given a word $w = \sigma_1\sigma_2\ldots\sigma_n$, with $\sigma_i \in \Sigma$, let $|w|$ denote its *length*. We use $w_{[i,j]}$ to denote the *infix* of w from σ_i to σ_j, for $j > i$ and define $w_{[i,j]} = \varepsilon$ if $i > j$, $i \notin [1, n]$ or $j \notin [1, n]$. Given two words $u, v \in \Sigma^\star$ we say that u is *lexicographically* less than v, denoted as $u < v$ if either $|u| < |v|$, or $|u| = |v|$ and the words can be written as $u = x\sigma_1 y$, $v = x\sigma_2 z$ with $\sigma_1 < \sigma_2$, for $\sigma_1, \sigma_2 \in \Sigma$ and $x, y, z \in \Sigma^\star$. The set Σ^ℓ corresponds to the set of all words on Σ of length $\ell \geq 0$. The number of occurrences of a symbol $\sigma \in \Sigma$ in a word w is denoted by $|w|_\sigma$. Given two words $u, v \in \Sigma^\star$ and a language $L \subseteq \Sigma^\star$, we say that they are *indistinguishable* w.r.t. L, denoted as $u \equiv_L v$, if $uw \in L \Leftrightarrow vw \in L$, for all $w \in \Sigma^\star$.

A *one-way nondeterministic finite automaton* (NFA) is defined as a tuple $\mathscr{A} = \langle Q, \Sigma, \delta, I, F\rangle$, where Q is a finite set of *states*, $\delta : Q \times \Sigma \to 2^Q$ is the *transition function*, $I \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. When $I = \{q_0\}$, we use $I = q_0$. As customary, we extend δ to words in the natural way. The *right language* of a state $q \in Q$ is $\mathscr{L}_q(\mathscr{A}) = \{\, w \in \Sigma^\star \mid \delta(q, w) \cap F \neq \emptyset \,\}$. The *language accepted* by $\mathscr{A}$ is $\mathscr{L}(\mathscr{A}) = \bigcup_{q\in I} \mathscr{L}_q(\mathscr{A})$. An NFA is *minimal* if no NFA with fewer states recognizes the same language. An NFA operates on a *read-only tape* containing the input word, with an *input head* that moves strictly from left to right and points to the current symbol being processed. An NFA is *deterministic* (DFA) if $|I| = 1$ and $|\delta(q, \sigma)| \leq 1$, for all $q \in Q$ and $\sigma \in \Sigma$. Two states $q_1, q_2 \in Q$ are *equivalent* (or *indistinguishable*) if $\mathscr{L}_{q_1}(\mathscr{A}) = \mathscr{L}_{q_2}(\mathscr{A})$. A *minimal* DFA has no equivalent states and it is unique up to isomorphisms.

To prove lower bounds on the number of states of NFAs, we shall use the *fooling set* technique [2], which we briefly recall. A fooling set for a language L is a set of pairs of words $\mathscr{F} = \{\, \langle x_i, y_i\rangle \mid i \in [1, n] \,\}$ of size $n \geq 1$ where, for each $i, j \in [1, n]$, $x_i y_i \in L$ and if $i \neq j$ then $x_i y_j \notin L$ or $x_j y_i \notin L$. It follows that every NFA for L has at least n states.

A *deterministic 1-limited automaton* (1-LA) is a one-tape Turing machine with restrictions on its rewritings. It has the ability to revisit the input symbols already processed and also to rewrite the contents of each tape cell only upon its first visit, but its head is not allowed to fall out of the input. To that end, we consider two new symbols $\rhd$ and $\lhd$ not belonging to Σ, called the *left* and *right end-marker*, respectively, surrounding each input word on the tape. By $\Sigma_{\rhd,\lhd}$ we denote the set $\Sigma \cup \{\rhd, \lhd\}$. Formally, a deterministic 1-limited automaton

is a tuple $\mathscr{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$, where Q, q_0, F are defined as before, Γ is a finite *work alphabet* such that $\Sigma_{\triangleright,\triangleleft} \subseteq \Gamma$, and $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function. According to δ and the current state, in one move $\mathscr{A}$ reads a symbol from the tape, changes its state, replaces the symbol read from the tape with a new symbol, and moves its head one position either forward or backward. In particular, $\delta(q_1, \sigma) = \langle q_2, X, d \rangle$ means that when the automaton in the state q_1 scans a cell containing the symbol σ, it enters the state q_2, rewrites the cell contents by X, and moves the head to the left or right, according to the direction $d \in \{-1, +1\}$. However, replacing symbols is only allowed during the first visit, with the exception of the cells containing the end-markers, which are *never* modified. Moreover, the end-marker symbols cannot be used to replace the contents of any of the cells. Formally, if $\delta(q_1, \sigma) = \langle q_2, X, d \rangle$ then either $\sigma \in \Sigma$ and $X \in \Gamma \setminus \Sigma_{\triangleright,\triangleleft}$, or $X = \sigma$. After the first visit, the content of a tape cell can no longer be modified. The machine accepts an input if, starting from the initial state with the input head on the left end-marker, it ends the computation in a final state after passing the right end-marker.

The *size* of a machine is the number of symbols required to describe it. For deterministic finite automata, the size is linear in the size of the transition function and in the number of states. Since the transition function is bounded by a polynomial in the number of states and input symbols, the overall size is $\Theta(|Q| \cdot |\Sigma| \cdot \log(|Q|))$. For nondeterministic finite automata, each state and input symbol has a set of possible next states, whose size can be as large as $|Q|$. Hence, the size of the transition function may be quadratic in the number of states, yielding $\Theta(|Q|^2 \cdot |\Sigma|)$. For 1-LAs, the size is bounded by a polynomial in the number of states and work symbols, namely, $\Theta(|Q| \cdot |\Gamma| \cdot \log(|Q| \cdot |\Gamma|))$.

3 Shuffle Operations

Given $u, v \in \Sigma^\star$, the *(general) shuffle* operation between u and v, denoted as $u \sqcup\!\sqcup v$, corresponds to the set

$$u \sqcup\!\sqcup v = \{\, u_1 v_1 u_2 v_2 \cdots u_n v_n \mid u_i, v_i \in \Sigma^\star, i \in [1, n], n \geq 1, \\ u = u_1 u_2 \cdots u_n, v = v_1 v_2 \cdots v_n \,\}.$$

Now, let $u = u_1 u_2 \cdots u_m$ and $v = v_1 v_2 \cdots v_n$, such that $u_i, v_j \in \Sigma$, for $i \in [1, m]$ and $j \in [1, n]$. The *perfect shuffle* of u and v, denoted as $u \sqcup\!\sqcup_\pi v$, results in one word obtained by strictly alternating symbols from u and v, starting with the first symbol from u, and defined only when the two words are of equal length. Formally,

$$u \sqcup\!\sqcup_\pi v = \begin{cases} \emptyset & \text{if } m \neq n; \\ \{\varepsilon\} & \text{if } m = n = 0; \\ \{u_1 v_1 u_2 v_2 \cdots u_n v_n\} & \text{otherwise.} \end{cases}$$

Example 1. Let $u = aba$ and $v = bab$. Then, $u \sqcup\!\sqcup_\pi v = \{abbaab\}$.

When the words u and v have different lengths, the perfect shuffle can be extended to the *initial literal shuffle*, denoted as $u \sqcup\!\sqcup_\iota v$ [1]. In this case, symbols from u and v are strictly interleaved while both words have symbols remaining, and once the shorter word is exhausted the remaining suffix of the longer word is appended to the result. It is formally defined as

$$u \sqcup\!\sqcup_\iota v = \begin{cases} (u \sqcup\!\sqcup_\pi v_{[1,m]}) \cdot v_{[m+1,n]} & \text{if } m \leq n; \\ (u_{[1,n]} \sqcup\!\sqcup_\pi v) \cdot u_{[n+1,m]} & \text{otherwise,} \end{cases}$$

where, given a set S and a word y, $S \cdot y = \{\, xy \mid x \in S \,\}$ and $y \cdot S = \{\, yx \mid x \in S \,\}$.

Example 2. Let $u = baaab$ and $v = abb$. Then, $u \sqcup\!\sqcup_\iota v = \{baababab\}$.

Finally, the operation most relevant to our study is the *general literal shuffle* of two words u and v, denoted as $u \sqcup\!\sqcup_\lambda v$. In this operation, the initial literal shuffle may start at an arbitrary position in one of the two words. Accordingly, the words can be partitioned into three (possibly empty) subwords: a prefix of one of the two words, followed by the perfect shuffle of symbols from both words, and a suffix of the word with symbols remaining. Formally, it is defined as

$$u \sqcup\!\sqcup_\lambda v = \{\, u_{[1,i]} \cdot (u_{[i+1,m]} \sqcup\!\sqcup_\iota v) \mid i \in [m] \,\} \cup \{\, v_{[1,j]} \cdot (u \sqcup\!\sqcup_\iota v_{[j+1,n]}) \mid j \in [n] \,\}$$

Example 3. Let $u = bab$ and $v = ababa$, and let

$$L = u \sqcup\!\sqcup_\lambda v = \{abababab, ababbaab, abbaabba, baabbaba, babababa\}.$$

We have that $ababbaab \in L$ as

$$ababbaab = v_{[1,3]} \cdot (u \sqcup\!\sqcup_\iota v_{[4,5]}) = v_{[1,3]} \cdot (u_{[1,2]} \sqcup\!\sqcup_\pi v_{[4,5]}) \cdot u_{[3,3]},$$

and $babababa \in L$ as

$$babababa = u_{[1,2]} \cdot (u_{[3,3]} \sqcup\!\sqcup_\iota v) = u_{[1,2]} \cdot (u_{[3,3]} \sqcup\!\sqcup_\pi v_{[1,1]}) \cdot v_{[2,5]}.$$

These operations on words are extended to languages in the natural way, that is, by shuffling words from each language in all possible combinations. Formally, let $L', L'' \subseteq \Sigma^\star$ and $\sqcup\!\sqcup_\circ \in \{\sqcup\!\sqcup, \sqcup\!\sqcup_\pi, \sqcup\!\sqcup_\iota, \sqcup\!\sqcup_\lambda\}$. Then,

$$L' \sqcup\!\sqcup_\circ L'' = \bigcup_{u \in L', v \in L''} u \sqcup\!\sqcup_\circ v.$$

Example 4. Let $L' = a^\star$ and $L'' = b^\star$. Then,

$$\begin{aligned} L' \sqcup\!\sqcup L'' &= (a+b)^\star, & L' \sqcup\!\sqcup_\pi L'' &= (ab)^\star, \\ L' \sqcup\!\sqcup_\iota L'' &= (ab)^\star(a^\star + b^\star), & L' \sqcup\!\sqcup_\lambda L'' &= (a^\star + b^\star)(ab)^\star(a^\star + b^\star). \end{aligned}$$

For convenience, we refer to the set of operators $\{\sqcup\!\sqcup_\pi, \sqcup\!\sqcup_\iota, \sqcup\!\sqcup_\lambda\}$ as *literal shuffles*.

4 Finite Automata for Literal Shuffle

It is known that the class of regular languages is closed under the general shuffle operation. We shall now construct an NFA that recognizes the general literal shuffle of the languages accepted by the given operand NFAs. This construction shows that regular languages are also closed under the literal shuffles.

Let $\mathscr{A}' = \langle Q', \Sigma, \delta', I', F' \rangle$ and $\mathscr{A}'' = \langle Q'', \Sigma, \delta'', I'', F'' \rangle$ be the two operand NFAs. We describe $\mathscr{A}_\lambda = \langle Q, \Sigma, \delta, I, F \rangle$, such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}') \sqcup_\lambda \mathscr{L}(\mathscr{A}'')$. First, consider the set of states

$$Q = (Q' \cup Q'' \cup (Q' \times Q'')) \times \{0, 1\}$$

and the initial and the set of final states, respectively,

$$I = (I' \cup I'' \cup (I' \times I'')) \times \{0\} \text{ and } F = ((F' \cup F'') \times \{1\}) \cup (F' \times F'' \times \{0\}).$$

Then, let the transition function δ be defined, for all $q' \in Q'$, $q'' \in Q''$, and $\sigma \in \Sigma$, as:

$$\delta(\langle q', 0\rangle, \sigma) = \{ \langle p', 0\rangle \mid p' \in \delta'(q', \sigma) \} \cup \{ \langle p', q_0'', 0\rangle \mid p' \in \delta'(q', \sigma), q_0'' \in I'' \},$$

where $\langle q', 0\rangle$ represents the state where, *prior* to performing the perfect shuffle, the input is being simulated on $\mathscr{A}'$ at the current state q'. From such a state, the simulation either continues on $\mathscr{A}'$ or initiates the perfect shuffle;

$$\delta(\langle q'', 0\rangle, \sigma) = \{ \langle p'', 0\rangle \mid p'' \in \delta''(q'', \sigma) \} \cup \{ \langle q_0', p'', 0\rangle \mid p'' \in \delta''(q'', \sigma), q_0' \in I' \},$$

analogous to $\delta(\langle q', 0\rangle, \sigma)$ where the input is being simulated on $\mathscr{A}''$;

$$\delta(\langle q', 1\rangle, \sigma) = \{ \langle p', 1\rangle \mid p' \in \delta'(q', \sigma) \},$$

where $\langle q', 1\rangle$ represents the state where, *after* performing the perfect shuffle, the input is being only simulated on $\mathscr{A}'$ at the current state q'. In this case, the simulation continues on $\mathscr{A}'$;

$$\delta(\langle q'', 1\rangle, \sigma) = \{ \langle p'', 1\rangle \mid p'' \in \delta''(q'', \sigma) \},$$

analogous to $\delta(\langle q', 1\rangle, \sigma)$, where the simulation continues on $\mathscr{A}''$;

$$\begin{aligned} \delta(\langle q', q'', 0\rangle, \sigma) = {} & \{ \langle p', q'', 1\rangle \mid p' \in \delta'(q', \sigma) \} \cup \\ & \{ \langle p'', 1\rangle \mid q' \in F', p'' \in \delta''(q'', \sigma) \} \cup \\ & \{ \langle p', 1\rangle \mid q'' \in F'', p' \in \delta'(q', \sigma) \}; \end{aligned}$$

where $\langle q', q'', 0\rangle$ represents the state where the perfect shuffle is being performed, while on the state q' of $\mathscr{A}'$ and q'' of $\mathscr{A}''$. In this state, the simulation may proceed in one of three ways: continue the perfect shuffle, with the next input symbol simulated on $\mathscr{A}'$; if $q' \in F'$, switch to simulating only on $\mathscr{A}''$; if $q'' \in F''$, switch to $\mathscr{A}'$;

$$\delta(\langle q', q'', 1\rangle, \sigma) = \{ \langle q', p'', 0\rangle \mid p'' \in \delta''(q'', \sigma) \},$$

analogous to $\delta(\langle q', q'', 0\rangle, \sigma)$, but now the next input symbol must be simulated on $\mathscr{A}''$.

Consider the following results:

Lemma 1. *Let $\mathscr{A}'$ and $\mathscr{A}''$ be two NFAs that recognize $L', L'' \subseteq \Sigma^\star$, respectively. Then, the automata for $L' \sqcup\!\sqcup_\pi L''$, $L' \sqcup\!\sqcup_\iota L''$, and $L' \sqcup\!\sqcup_\lambda L''$ can be effectively constructed from the construction given above.*

Sketch of Proof. Let $\mathscr{A}' = \langle Q', \Sigma, \delta', I', F' \rangle$ and $\mathscr{A}'' = \langle Q'', \Sigma, \delta'', I'', F'' \rangle$. Moreover, let $\mathscr{A}_\lambda = \langle Q, \Sigma, \delta, I, F \rangle$ be the automaton resulting from the construction above when provided with the automata $\mathscr{A}'$ and $\mathscr{A}''$. Then,

1. $\mathscr{A}_\lambda$ recognizes $L' \sqcup\!\sqcup_\lambda L''$:
   The initial states correspond to the three different possible starting scenarios: perform the perfect shuffle from the beginning ($I' \times I'' \times \{0\}$); start with a simulation on $\mathscr{A}'$ alone ($I' \times \{0\}$); or on $\mathscr{A}''$ alone ($I'' \times \{0\}$). The final states correspond to the three different possible ending scenarios: end with the perfect shuffle ($F' \times F'' \times \{0\}$); end with the rest of input simulated on $\mathscr{A}'$ only ($F' \times \{1\}$); or on $\mathscr{A}''$ ($F'' \times \{1\}$).
2. $\mathscr{A}_\iota = \langle Q_\iota, \Sigma, \delta, I_\iota, F \rangle$, where
   $$Q_\iota = Q \setminus ((Q' \cup Q'') \times \{0\}) \text{ and } I_\iota = I \setminus ((I' \cup I'') \times \{0\})$$
   recognizes $L' \sqcup\!\sqcup_\iota L''$:
   In the initial literal shuffle, the simulation cannot begin solely on $\mathscr{A}'$ or $\mathscr{A}''$ prior to the perfect shuffle phase. Hence, we remove those states from Q and adjust the initial states accordingly, while the final states remain unchanged.
3. $\mathscr{A}_\pi = \langle Q_\pi, \Sigma, \delta, I_\iota, F_\pi \rangle$, where
   $$Q_\pi = Q \setminus ((Q' \cup Q'') \times \{0,1\}) \text{ and } F_\pi = F \setminus ((F' \cup F'') \times \{1\})$$
   recognizes $L' \sqcup\!\sqcup_\pi L''$:
   For the perfect shuffle, only the states simulating the shuffle itself are required. Thus, the initial states are defined as in the case of the initial literal shuffle, but the final states are updated.

Note that $|Q| = 2(mn + m + n)$, $|Q_\iota| = 2mn + m + n$, and $|Q_\pi| = 2mn$. □

4.1 Number of States of NFAs Recognizing the Literal Shuffles

Regarding the perfect shuffle, it is know that given two languages $L', L'' \subseteq \Sigma^\star$ recognized by two DFAs of m and n states, respectively, then $2mn$ states are sufficient for a DFA to recognize $L' \sqcup\!\sqcup_\pi L''$, and the bound is tight [4,5]. For every $\sigma \in \Sigma$ and $n \geq 0$, consider the language $L_{\sigma,n} = \{ w \in \Sigma^\star \mid |w|_\sigma \equiv 0 \bmod n \}$, over a k-symbol alphabet Σ.

Lemma 2. *Let $m, n > 0$ and $\Sigma = \{a, b\}$. Then, $2mn$ states are necessary for an NFA to recognize $L_{a,m} \sqcup\!\sqcup_\pi L_{b,n}$.*

Proof. Consider the set $\mathscr{F} = \{ \langle a^{2i}b^j, a^{2(m-i)}b^{2n-j} \rangle \mid i \in [m-1], j \in [2n-1] \}$. By adapting the proof for the tightness of the number of states required for a DFA [4,5], it follows that $\mathscr{F}$ forms a fooling set for $L_{a,m} \sqcup\!\sqcup_\pi L_{b,n}$. □

Since the number of states of NFAs is bounded by that of their equivalent deterministic counterparts and by Lemma 2, we have the following result.

Theorem 1. *Let $\mathscr{A}'$ and $\mathscr{A}''$ be two NFAs with m and n states, respectively. Then, $2mn$ states are sufficient to recognize $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\pi \mathscr{L}(\mathscr{A}'')$, and the bound is tight.*

We shall now consider the remaining shuffle operations: the initial literal shuffle and the general literal shuffle. Again, from the construction presented above, we have an upper bound on the number of states of the NFAs recognizing the resulting languages.

Lemma 3. *Let $\mathscr{A}'$ and $\mathscr{A}''$ be two NFAs with m and n states, respectively. Then, $2mn + m + n$ and $2(mn + m + n)$ states are sufficient for an NFA to recognize $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\iota \mathscr{L}(\mathscr{A}'')$ and $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\lambda \mathscr{L}(\mathscr{A}'')$, respectively.*

Adapting the fooling set argument used for the perfect shuffle, it is possible to prove that the bound in Lemma 3 is tight. For $m, n > 0$, let $L_{a,m}$ be defined over the alphabet $\{a, b\}$ and $L_{c,n}$ over $\{c, d\}$.

Lemma 4. *Let $m, n > 0$ and $\Sigma = \{a, b, c, d\}$. Then, $2mn+m+n$ and $2(mn+m+n)$ states are necessary for an NFA to recognize $L_{a,m} \sqcup\!\sqcup_\iota L_{c,n}$ and $L_{a,m} \sqcup\!\sqcup_\lambda L_{c,n}$, respectively.*

We obtain the following result from Lemmas 3 and 4.

Theorem 2. *Let $\mathscr{A}'$ and $\mathscr{A}''$ be two NFAs with m and n states, respectively. Then, $2mn + m + n$ and $2(mn + m + n)$ states are sufficient for an NFA to recognize $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\iota \mathscr{L}(\mathscr{A}'')$ and $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\lambda \mathscr{L}(\mathscr{A}'')$, respectively, and the bound is tight for $|\Sigma| \geq 4$.*

4.2 Number of States of DFAs Recognizing the Literal Shuffles

We now turn to the size of DFAs for the various shuffle operators. Let $\mathscr{A}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ and $\mathscr{A}'' = \langle Q'', \Sigma, \delta'', q_0'', F'' \rangle$ be DFAs with m and n states, respectively, for two regular languages L' and L'', respectively. Moreover, let $\mathscr{A}_\iota = \langle Q, \Sigma, \delta, I, F \rangle$ denote the NFA resulting from the construction presented in the previous section recognizing $L' \sqcup\!\sqcup_\iota L''$.

The number of states for a DFA to recognize $L' \sqcup\!\sqcup_\iota L''$ has as upper bound the size of the resulting DFA from the well-known subset construction applied to $\mathscr{A}_\iota$, namely, $\mathscr{D}(\mathscr{A}_\iota)$. That is, $2^{|Q|} = 2^{2mn+m+n}$ states. However, one can notice that some of the subsets can be saved. Let $s \in 2^Q$ be a state from $\mathscr{D}(\mathscr{A}_\iota)$, which is a subset of states of $\mathscr{A}_\iota$. Then, s can be decomposed into two disjoint sets $s = s_1 \cup s_2$ such that $s_1 \subseteq Q' \times Q'' \times \{0, 1\}$ and $s_2 \subseteq (Q' \cup Q'') \times \{1\}$. Observe that if the state s is reachable in $\mathscr{D}(\mathscr{A}_\iota)$, then $|s_1| = 1$. This follows from the fact that δ' and δ'' are deterministic. Thus, we obtain the following upper bound on the number of states of $\mathscr{D}(\mathscr{A}_\iota)$.

Theorem 3. *Given two DFAs $\mathscr{A}'$ and $\mathscr{A}''$ with m and n states, respectively, $2^{m+n+1}mn$ states are sufficient for a DFA to recognize $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\iota \mathscr{L}(\mathscr{A}'')$.*

Now, let us consider the general literal shuffle operation between two regular languages $L', L'' \subseteq \Sigma^\star$. Once again, let $\mathscr{A}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ and $\mathscr{A}'' = \langle Q'', \Sigma, \delta'', q_0'', F'' \rangle$ be DFAs for L' and L'' with m and n states, respectively. Now, let $\mathscr{A}_\lambda = \langle Q, \Sigma, \delta, I, F \rangle$ denote the NFA resulting from the construction presented in the previous section recognizing $L' \sqcup\!\sqcup_\lambda L''$. Of course, $2^{|Q|} = 2^{2(mn+m+n)}$ is an upper bound on the number of states of $\mathscr{D}(\mathscr{A}_\lambda)$. However, again, some of the subsets can be saved. Let $s \in 2^Q$ be a state from $\mathscr{D}(\mathscr{A}_\lambda)$, which is a subset of states of $\mathscr{A}_\lambda$. Then, s can be decomposed into four disjoint sets $s = s_1 \cup s_2 \cup s_3 \cup s_4$ such that $s_1 \subseteq Q' \times \{0\}$, $s_2 \subseteq Q'' \times \{0\}$, $s_3 \subseteq (Q' \cup Q'') \times \{1\}$, and $s_4 \subseteq Q' \times Q'' \times \{0, 1\}$. Observe that if the state s is reachable in $\mathscr{D}(\mathscr{A}_\lambda)$, then $|s_1| = |s_2| = 1$. This follows again from the fact that δ' and δ'' are deterministic. Hence, the image of the states $Q' \times \{0\}$ (resp., $Q'' \times \{0\}$) under δ — that is, the states where the automaton simulates the input only on $\mathscr{A}'$ (resp., $\mathscr{A}''$) *prior* to the perfect shuffle — contains a single state within $Q' \times \{0\}$ (resp., $Q'' \times \{0\}$). Thus, we obtain the following upper bound on the number of states of $\mathscr{D}(\mathscr{A}_\lambda)$.

Theorem 4. *Given two DFAs $\mathscr{A}'$ and $\mathscr{A}''$ with m and n states, respectively, $2^{mn+m+n}mn$ states are sufficient for a DFA to recognize $\mathscr{L}(\mathscr{A}') \sqcup\!\sqcup_\lambda \mathscr{L}(\mathscr{A}'')$.*

We now introduce a family of languages that leads to an exponential blow-up in the size of the minimal DFA recognizing their literal shuffle. This result relies on the fact that the concatenation of two DFAs can result in an exponential increase in size. Since the general literal shuffle of two languages includes, in particular, their concatenation, we shall show how this complexity carries over to the shuffle. For $n \geq 0$, consider the language $L_n = \{\, \#w(\#\Sigma^n)^\star\#w \mid w \in \Sigma^n \,\}$ defined over a k-letter alphabet Σ. Of course, every automaton recognizing L_n must *remember* the prefix of length $n+1$ of the input word, which implies that $O(k^n)$ states are needed. It can be noticed that, given two words $x, y \in L_n$, where $x = \#x'(\#\Sigma^n)^\star\#x'$, words of the form $xy \in L_nL_n \subset L_n \sqcup\!\sqcup_\lambda L_n$ are the *hardest* to recognize. The reason is that, when concatenating these two strings, the automaton has no deterministic way to tell where x ends and y begins. Because of this ambiguity, the DFA is forced to keep track of every length-n factor that follows each occurrence of x', starting from the second one. As a consequence, the number of states necessary to keep this information is $\Omega(2^{k^n})$.

Lemma 5. *For $n \geq 0$ and $k \geq 2$, every DFA recognizing $L_n \sqcup\!\sqcup_\lambda L_n$ over a k-letter alphabet needs $\Omega(2^{k^n})$ states.*

5 Polynomial Simulations with 1-Limited Automata

In this section, we study the cost of simulating the initial and the general literal shuffle of two languages using deterministic 1-LAs, assuming the input languages are given by DFAs. By leveraging the ability of 1-LAs to rewrite the tape and perform two-way scanning, we show that the literal shuffle of two languages can be recognized by deterministic 1-LAs of polynomial size in the number of states of their DFAs.

Let $\mathscr{A}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ and $\mathscr{A}'' = \langle Q'', \Sigma, \delta'', q_0'', F'' \rangle$ be two DFAs with m and n states, respectively, recognizing the languages $L', L'' \subseteq \Sigma^\star$. We shall describe how to obtain a deterministic 1-LA $\mathscr{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ accepting $L = L' \sqcup\!\sqcup_\lambda L''$ in such a way that the size of $\mathscr{A}$ is polynomial in the sizes of $\mathscr{A}'$ and $\mathscr{A}''$.

Our objective is to avoid the exponential blow-up in size by exploiting the rewriting capabilities of 1-LAs. Instead of storing the set of states in the input automata reached by the different literal shuffle scenarios within the states of the machine (as is required in a DFA), $\mathscr{A}$ encodes it on the tape. This information is then accessed using the ability of 1-LAs of scanning the tape in a two-way fashion, to access and resume the simulation corresponding to each scenario.

First, the tape is logically partitioned into blocks of $2mn$ cells, with a possibly shorter final block, where each cell in a block represents a state in the product space $Q' \times Q'' \times \{0,1\}$. Given a state $\langle q', q'', \mathsf{b} \rangle \in Q' \times Q'' \times \{0,1\}$, we define its index in an ordered enumeration of $Q' \times Q'' \times \{0,1\}$ as $\mathsf{ind}(\langle q', q'', \mathsf{b} \rangle) = c$.

To enable the writing and reading of the information necessary to simulate the general literal shuffle of the languages recognized by $\mathscr{A}'$ and $\mathscr{A}''$, while ensuring the simulation cost is polynomial in the size of the input automata, the simulating 1-LA operates within sliding windows spanning two consecutive blocks of cells, i.e., virtual windows of size $4mn$. During the overwriting of a window, the *left block* has been fully overwritten, while the *right block* contains, at some position, the leftmost cell that has not yet been overwritten, referred to as the *relative frontier*. We refer to the positions of the current window as pairs in $[2mn-1] \times \{\mathsf{L}, \mathsf{R}\}$, where the second component indicates whether the position given in the first component belongs to the left (L) or right (R) block of the window. As the 1-LA operates in windows, we consider the infix $w = \sigma_1 \sigma_2 \cdots \sigma_{2mn}$ of the original input as the word read left-to-right of the left block of a window. As a special case, at the beginning of the simulation, the left block of the window has length 1 and only contains the left end-marker and thus $w = \varepsilon$. Moreover, the frontier is initially at position $(0, \mathsf{R})$, and the head of the machine is on the left end-marker.

The written information is organized into *four* tracks. In particular, consider the c-th cell of a given block, with $c \in [2mn - 1]$, and the state $\langle q', q'', \mathsf{b} \rangle$ for which $\mathsf{ind}(\langle q', q'', \mathsf{b} \rangle) = c$. Then,

- The *first track* retains a copy of the input symbol originally contained in the cell prior to any rewriting, ensuring that the symbol remains accessible throughout all possible simulations of the shuffle;
- The *second track* contains a marker indicating whether (✓) or not (✗), *prior* to executing the perfect shuffle, the state q' is reachable in $\mathscr{A}'$ (resp., q'' is reachable in $\mathscr{A}''$) depending on whether $\mathsf{b} = 0$ (resp., $\mathsf{b} = 1$), when simulating the input prefix up to the end of the previous window;
- The *third track* contains a marker indicating whether (✓) or not (✗), after processing symbols from either $\mathscr{A}'$ or $\mathscr{A}''$, the states q' and q'' are reachable on $\mathscr{A}'$ and $\mathscr{A}''$, respectively, while performing the perfect shuffle, when simulating the input prefix as before;

Procedure 1 `literalShuffle`$(\mathscr{A}', \mathscr{A}'')$
Given two DFAs $\mathscr{A}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ and $\mathscr{A}'' = \langle Q'', \Sigma, \delta'', q_0'', F'' \rangle$, where $m = |Q'|$ and $n = |Q''|$, the deterministic 1-LA implementing this procedure recognizes the language $\mathscr{L}(\mathscr{A}') \sqcup_\lambda \mathscr{L}(\mathscr{A}'')$.

```
 1: relativeFrontier ← 0
 2: while TRUE do
 3:     ⟨q′, q″, b⟩ ← ind^{-1}(relativeFrontier)
 4:     sndTrack ← markSndTrack(⟨q′, q″, b⟩, A′, A″)
 5:     trdTrack ← markTrdTrack(⟨q′, q″, b⟩, A′, A″)
 6:     fthTrack ← markFthTrack(⟨q′, q″, b⟩, A′, A″)
 7:     move the head rightward until reaching position (relativeFrontier, R)
 8:     if current symbol ≠ ◁ then
 9:         write(current symbol, sndTrack, trdTrack, fthTrack)
10:         relativeFrontier ← (relativeFrontier + 1) mod 2mn
11:     else
12:         if checkReachability(A′, A″) then ACCEPT else REJECT
```

– The *fourth track* contains a marker indicating whether (✓) or not (✗), *after* executing the perfect shuffle, the state q' is reachable in $\mathscr{A}'$ (resp., q'' is reachable in $\mathscr{A}''$) depending on whether $\mathsf{b} = 0$ (resp., $\mathsf{b} = 1$), when simulating the input prefix as before.

We now present in detail on how $\mathscr{A}$ recognizes $L' \sqcup_\lambda L''$ (see Procedure 1). The deterministic 1-LA stores in its finite control the position of the frontier within the right block of the current window (relativeFrontier), which is initially set to 0. At every step of the simulation, *before* entering the cell at the frontier, the 1-LA gathers the information regarding the reachability of the state encoded in that cell, which is given by $\mathsf{ind}(\mathsf{state})$. To do so, it uses the procedures `markSndTrack`, `markTrdTrack`, and `markFthTrack` to determine the values to be written in the second, third, and fourth tracks of the cell at the frontier, respectively. Then, it moves the input head to the right until reaching the cell at the frontier (this is possible since the position of the frontier is stored in the finite control), and, using the procedure `write`, it overwrites the cell with the information gathered. Finally, the frontier advances to the next cell updating the value of relativeFrontier accordingly, that is, relativeFrontier is incremented by one modulo $2mn$. This process is repeated until the right end-marker is reached, at which point the 1-LA performs a final check to determine whether to accept or reject the input word. The contents written in the *right block* of the final window, which may be shorter than $2mn$ in size, are going to be ignored during this check. Instead, for every state $q = \langle q', q'', \mathsf{b} \rangle \in F' \times F'' \times \{0, 1\}$, the 1-LA checks whether q can be reached by any of the states encoded in the *left block* of the final window, continuing (or starting) the simulation of the shuffles on the suffix of the input word contained in the last window. This is done by using the procedure `checkReachability`, that reuses adapted versions of the functions `markSndTrack`, `markTrdTrack`, and `markFthTrack` and accepts according to the values returned by them.

Theorem 5. *Let $\mathscr{A}'$ and $\mathscr{A}''$ be two DFAs with m and n states, respectively, over a k-letter alphabet Σ. Then, there exists a deterministic 1-LA recognizing $\mathscr{L}(\mathscr{A}') \sqcup_\lambda \mathscr{L}(\mathscr{A}'')$ with $O(m^5n^5)$ states and $9k+2$ work symbols.*

Proof. Let $\mathscr{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ be the deterministic 1-LA implementing Procedure 1 described above. The work alphabet Γ of $\mathscr{A}$ is $\Gamma = \Sigma_{\triangleright,\triangleleft} \cup (\Sigma \times \{\checkmark, \text{✗}\} \times \{\checkmark, \text{✗}\} \times \{\checkmark, \text{✗}\})$ of size $|\Sigma_{\triangleright,\triangleleft}| + 8|\Sigma| = 9k+2$. Moreover, the automaton stores the following components in its states:

- The position of the head within the window of size $4mn$. This is required to know the position of the head with respect to the frontier, and not to pass it before gathering the information to be written at the frontier cell. Moreover, it is also needed to navigate through the left block of the window to retrieve the information stored in the second, third, and fourth tracks;
- The position of the relative frontier (relativeFrontier in Procedure 1) within the right block of the window, whose size is $2mn$;
- Three bits for the values returned by `markSndTrack`, `markTrdTrack`, and `markFthTrack`;
- The procedure `markSndTrack` requires a component of size $2mn$ to loop through the cells of the left block of the window and retrieve the information stored in the second track, and a component of size $2mn$ to keep track of the states reached during the simulation of the prefix of the input word on either $\mathscr{A}'$ or $\mathscr{A}''$. This gives a total of $4m^2n^2$ states for this part of the simulation;
- Similarly, the procedure `markTrdTrack` requires a component of size $2mn$ to loop through the cells of the left block of the window and retrieve the information stored in the second and third tracks, plus a component of size $2mn$ to keep track of the states reached during the simulations of the shuffles. Moreover, the procedure needs an extra component of size $2mn$ to consider all possible splits of the word w into two parts: one to continue the head on either $\mathscr{A}'$ or $\mathscr{A}''$, and another to simulate the shuffle (see Case 2). This gives a total of $8m^3n^3$ states for this part of the simulation;
- The procedure `markFthTrack` requires a component of size $2mn$ to loop through the cells of the left block of the window and retrieve the information stored in the second, third, and fourth tracks. It also needs a component of size $2mn$ to keep track of the reached states. Moreover, the procedure needs two extra components of size $2mn$ to consider all possible splits of the word w into three parts: one to continue the head on either $\mathscr{A}'$ or $\mathscr{A}''$, a second to perform the shuffle, and a third to start the tail on either $\mathscr{A}'$ or $\mathscr{A}''$ (see Case 3). This gives a total of $16m^4n^4$ states for this part of the simulation;
- Some additional state components of constant size used to keep track of the execution flow of the procedure.

Therefore, the number of states of $\mathscr{A}$ is $16mn(4m^2n^2 + 8m^3n^3 + 16m^4n^4) \in O(m^5n^5)$. □

It can be observed that the 1-LA implementing Procedure 1, when executed without the second track, recognizes the initial literal shuffle of the languages accepted by the operand automata. We then have the following result:

Corollary 1. *Let $\mathscr{A}'$ and $\mathscr{A}''$ be two DFAs with m and n states, respectively, over a k-letter alphabet Σ. Then, there exists a deterministic 1-LA recognizing $\mathscr{L}(\mathscr{A}') \sqcup_\iota \mathscr{L}(\mathscr{A}'')$ with $O(m^5n^5)$ states and $5k+2$ work symbols.*

References

1. Berard, B.: Literal shuffle. Theoretical Computer Science **51**(3), 281–299 (1987). https://doi.org/10.1016/0304-3975(87)90037-5
2. Birget, J.C.: Intersection and union of regular languages and state complexity. Information Processing Letters **43**(4), 185–190 (1992). https://doi.org/10.1016/0020-0190(92)90198-5
3. Brzozowski, J.A., Jirásková, G., Liu, B., Rajasekaran, A., Szykula, M.: On the state complexity of the shuffle of regular languages. In: Câmpeanu, C., Manea, F., Shallit, J.O. (eds.) 18th DCFS, 2016. LNCS, vol. 9777, pp. 73–86. Springer (2016). https://doi.org/10.1007/978-3-319-41114-9_6
4. Domaratzki, M.: Trajectory-Based Operations. Ph.D. thesis, Queen's University, Kingston, Ontario, Canada (2004)
5. Domaratzki, M., Salomaa, K.: State complexity of shuffle on trajectories. J. Autom. Lang. Comb. **9**(2/3), 217–232 (2004). https://doi.org/10.25596/JALC-2004-217
6. Guillon, B., Prigioniero, L., Taheri, J.: Polynomial complementation of nondeterministic two-way finite automata by 1-limited automata. In: Mahajan, M., Manea, F., McIver, A., Nguyen, K.T. (eds.) 43rd International Symposium on Theoretical Aspects of Computer Science, STACS 2026, Grenoble, France. LIPIcs, vol. 364, pp. 48:1–48:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2026). https://doi.org/10.4230/LIPICS.STACS.2026.48
7. Hibbard, T.N.: A generalization of context-free determinism. Information and Control **11**(1), 196–238 (1967). https://doi.org/10.1016/S0019-9958(67)90513-X
8. Hoffmann, S.: The n-ary initial literal and literal shuffle. Discrete Applied Mathematics **376**, 112–132 (2025). https://doi.org/10.1016/j.dam.2025.04.023
9. Pighizzini, G., Prigioniero, L., Sádovský, S.: Performing regular operations with 1-limited automata. Theory Comput. Syst. **68**(3), 465–486 (2024). https://doi.org/10.1007/S00224-024-10163-1
10. Wagner, K.W., Wechsung, G.: Computational complexity. D. Reidel Publishing Company, Dordrecht (1986)