Theory of Computing Systems
https://doi.org/10.1007/s00224-024-10163-1

Performing Regular Operations with 1-Limited Automata

Giovanni Pighizzini[1] · Luca Prigioniero[2] · Šimon Sádovský[3]

Accepted: 18 January 2024

Abstract
The descriptional complexity of basic operations on regular languages using 1-limited automata, a restricted version of one-tape Turing machines, is investigated. When simulating operations on deterministic finite automata with deterministic 1-limited automata, the sizes of the resulting devices are polynomial in the sizes of the simulated machines. The situation is different when the operations are applied to deterministic 1-limited automata: while for boolean operations the simulations remain polynomial, for product, star, and reversal they cost exponential in size. The costs for product and star do not reduce if the given machines are sweeping two-way deterministic finite automata. These bounds are tight.

Keywords Descriptional complexity · Models of computation · Regular languages

1 Introduction

It is well known that the class of regular languages is characterized by finite automata and it is closed under several operations. When a class of languages benefits of such strong closure properties, it is quite natural to ask how much these operations cost in terms of the size of the descriptions of recognizing devices. More precisely, for any fixed operation, the goal is to study the (possibly optimal) size of a machine accepting

This is an extended version of [1].

✉ Luca Prigioniero
l.prigioniero@lboro.ac.uk

Giovanni Pighizzini
pighizzini@di.unimi.it

Šimon Sádovský
sadovsky@dcs.fmph.uniba.sk

1 Dipartimento di Informatica, Università degli Studi di Milano, via Celoria, 18, Milan 20133, Italy

2 Department of Computer Science, Loughborough University, Epinal Way, Loughborough LE11 3TU, UK

3 Department of Computer Science, Comenius University, Mlynská Dolina, Bratislava 842 48, Slovakia

the language resulting by applying the operation as a function of the sizes of the smallest machines accepting the languages on which the operation is applied. In this paper we focus on union, intersection, complementation, product, star, and reversal. These operations have been extensively studied in the literature [2–5]. Their costs in the case of deterministic finite automata are summarized in Table 1.

Providing finite automata with the ability of performing nondeterministic moves, as well as scanning the input in a two-way fashion, does not add computational power to these devices [6, 7]. In other words, also two-way finite automata (in both deterministic and nondeterministic versions) characterize the class of regular languages. The descriptional complexity of operations on these extensions of finite automata has also been considered [8, 9].

In this paper we further extend this research by investigating the descriptional complexity of language operations for another class of machines, which still characterizes regular languages, namely for 1*-limited automata*.

Limited automata are single-tape Turing machines with rewriting restrictions, introduced by Hibbard in 1967 [10] and recently reconsidered and deeply investigated (see, e.g., [11–17]). These devices are two-way finite automata with the extra capability of overwriting the contents of each tape cell only in the first d visits, for a fixed constant $d \geq 0$ (we use the name d*-limited automaton* to explicitly mention the constant d). In his original paper, Hibbard proved that, for any fixed $d \geq 2$, d-limited automata have the same computational power as pushdown automata, namely they accept exactly context-free languages. Moreover, the deterministic version of such devices defines a hierarchy where, for each $d \geq 2$, the class of languages accepted by *deterministic* d-limited automata is properly included in the class of languages accepted by *deterministic* $(d + 1)$-limited automata. This hierarchy does not cover all the class of context-free languages. Indeed, there are some context-free languages which cannot be accepted by any deterministic d-limited automaton, for each arbitrarily large d [10]. At the bottom level of the hierarchy, deterministic 2-limited automata recognize exactly the class of deterministic context-free languages [12].

For $d = 0$ no rewritings are possible, hence the resulting models are two-way finite automata, which, as already mentioned, recognize exactly the class of regular languages in both deterministic and nondeterministic cases [6, 7]. The computational power does not increase if the rewritings in each cell are restricted *only to the first*

Table 1 The costs of operations on deterministic finite automata [5]

Operation	Size
Union	nm
Intersection	nm
Complement	$n+1$
Product	$n2^m - 2^{m-1}$
Kleene star	$\frac{3}{4}2^n$
Reversal	2^n

For each operation, it is shown the size of the automaton obtained by applying the corresponding operation to an automaton of size n (and one of size m, in case of binary operations) *visit*. In other words, 1-limited automata are no more powerful than finite automata [18, p. 209].

However, by considering the sizes of the descriptions, we can see that 1-limited automata can be significantly more succinct than finite automata. In particular, a double exponential size gap between 1-limited automata and one-way deterministic finite automata has been proved, while exponential size gaps have been proved for the conversions from 1-limited automata into one-way nondeterministic finite automata and from deterministic 1-limited automata into one-way deterministic finite automata [11].

In the study of the descriptional complexity of language operations, we consider a class of *source* devices and a class of *target* devices. For any given operation, the goal is to investigate the size of target devices recognizing the languages resulting from the operation, as a function of the sizes of source devices that specify the languages on which the operation is applied. Up to now, results in the literature consider source and target devices from the same family. In this setting, if we apply classical constructions to simulate operations on one-way deterministic finite automata, we obtain exponential size blowup in some cases (i.e., product, star, and reversal) because, intuitively, some nondeterministic steps are introduced, and their elimination costs exponential in finite automata (see Table 1). Hence, we wondered whether, using deterministic models with further capabilities (as 1-limited automata), it is possible to bypass nondeterministic computations introduced in one-way finite automata to simulate these operations, thus avoiding such exponential size growth.

Therefore, for each operation we study, we first take finite automata as source devices, and we simulate the operations on them with 1-limited automata as target devices. We emphasize that we consider *deterministic machines* only. Therefore, we prove that, despite the capability of 1-limited automata of rewriting the cells of the tape during the first visit does not make this model more powerful than finite automata, using these machines as target devices for simulating operations between finite automata yields 1-limited automata more succinct than equivalent finite automata.

On the other hand, when considering 1-limited automata as source and target devices, the simulations cost polynomial only in the case of union, intersection, and complementation. In the case of reversal, product, and star, however, we were able to find exponential lower bounds witnessing that there is no smaller automaton than the one obtained by converting the source deterministic 1-limited automata into one-way finite automata first (obtaining exponentially larger machines), and then applying the corresponding (polynomial-size) language operation construction for obtaining a deterministic 1-limited automaton. In the case of product and star, these lower bounds still hold when restricting the class of source machines to sweeping two-way deterministic finite automata, and keeping deterministic 1-limited automata as target machines.

2 Preliminaries

In this section we recall some basic definitions and notations. We assume the reader familiar with notions from formal languages and automata theory (see, e.g., [19]). Given a set S, $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ, we denote by $|w|$ the length of a string $w \in \Sigma^*$, by w^{R} the reversal of w, and by ε the empty string. Given two languages $L, L' \subseteq \Sigma^*$, L^{R} denotes the *reversal of* L, namely $L^{\mathrm{R}} = \{w^{\mathrm{R}} \mid w \in L\}$, L^* denotes the *Kleene star* of L, L^{c} denotes the *complement* of L, and $L \cdot L'$, $L \cup L'$, and $L \cap L'$ denote the *product*, *union*, and *intersection* of L and L', respectively (with the usual meaning).

Given a machine $\mathcal{M}$, the language accepted by it will be denoted as $\mathcal{L}(\mathcal{M})$. In this work we focus on *deterministic* machines. A *one-way deterministic finite automaton* (1DFA) is defined as usual as a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite *set of states*, Σ is a finite *input alphabet*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is a *set of final states*, and $\delta\colon Q \times \Sigma \to Q$ is a partial *transition function*. At each step, according to its current state p and the symbol σ scanned by the head, $\mathcal{A}$ enters the state $\delta(p, \sigma)$, if defined, and moves the input head rightward to the next symbol. The machine *accepts* the input if, starting from the initial state q_0 with the head on the leftmost input symbol, ends the computation in a final state $q \in F$ after having read the whole input.

Providing 1DFAs with the ability of moving the head back and forth, we obtain *two-way deterministic finite automata* (2DFAs). They are defined by extending the transition function so that a left (-1) or right $(+1)$ head direction is indicated in each instruction. Furthermore, to prevent the head to fall out the input, two special symbols $\rhd$ and $\lhd$ not belonging to Σ, called the *left* and the *right end-marker*, respectively, surround each input word, and enforce the computation to stay between them (except at the end of computation when accepting, as described below). More precisely, on input w the tape contains $\rhd w \lhd$, the left end-marker being at position 0 and the right end-marker being at position $|w| + 1$. By $\Sigma_{\rhd,\lhd}$ we denote the set $\Sigma \cup \{\rhd, \lhd\}$. Formally, the transition function of a two-way automaton is $\delta : Q \times \Sigma_{\rhd,\lhd} \to Q \times \{-1, +1\}$ such that, for each transition $(q, d) \in \delta(p, \sigma)$, if $\sigma = \rhd$ then $d = +1$, and if $\sigma = \lhd$ then $d = -1$ or $q \in F$. In this way, the head cannot pass the end-markers, except at the end of computation to accept. The machine *accepts* the input if, starting from the initial state q_0 with the head on the 1-st tape cell (i.e., scanning the leftmost symbol of the input), it ends its computation in a final state $q \in F$ after passing the right end-marker (i.e., with a move to the right).

We also consider a restriction of 2DFAs in which the direction of the head can change only at the end-markers [20]. These devices are called *sweeping* 2DFA*s*.

We now introduce the main model we are interested in. A *deterministic* 1*-limited automaton* (deterministic 1- LA) is a 2DFA which can rewrite the contents of each tape cell in the first visit only. Formally, it is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, q_0, F are defined as for 2DFAs Γ is a finite *working alphabet* such that $\Sigma_{\rhd,\lhd} \subseteq \Gamma$, and $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function. In one move, according to the transition function and to the current state, $\mathcal{A}$ reads a symbol from the tape, changes its state, replaces the symbol just read from the tape by a new symbol, and moves its head to one position forward or backward. In particular, $\delta(p, a) = (q, X, m)$ means that when the automaton in the state p is scanning a cell containing the symbol a, it enters the state q, rewrites the cell contents by X, and moves the head to *left*, if $m = -1$, or to *right*, if $m = +1$. However, there are the following restrictions:

- Replacing symbols is allowed to modify the contents of each cell only during the first visit, with the exception of the cells containing the end-markers, which are never modified. After the first visit, the contents of the cell are said to be *frozen*.
- The end-marker symbols cannot be used to replace the contents of any of the cells which initially contain the input.

The two previous conditions can be formalized by asking that if $\delta(p, a) = (q, X, m)$ then either $a \in \Sigma$ and $X \in \Gamma \setminus \Sigma_{\{\rhd,\lhd\}}$, or $X = a$.

The *size* of a machine is given by the total number of symbols used to write down its description. Therefore, the size of deterministic 1-LAs is bounded by a polynomial in the number of states and of working symbols, namely, it is $\Theta(\#Q \cdot \#\Gamma \cdot \log(\#Q \cdot \#\Gamma))$. In the case of deterministic finite automata, since no writings are allowed and hence the working alphabet is not provided, the size is linear in the number of instructions and states, which is bounded by a polynomial in the number of states and in the number of input symbols, namely, it is $\Theta(\#Q \cdot \#\Sigma \cdot \log(\#Q))$.

We say that a machine works in *linear time* if there exists a linearly growing function $f : \mathbb{N} \to \mathbb{N}$ such that, for every accepted word w, the number of moves along a computation accepting w is bounded by $f(|w|)$.

3 Product and Kleene Star

We start our investigation by studying the operations of product and star. It is known that the costs for these operations on 1DFAs are exponential due to the need of simulating in a deterministic way the nondeterministic choices used for decomposing the input string into factors from the given languages. However, we show that, using deterministic 1-LAs as target machines, the costs reduce to polynomials. Then, we analyze the simulations of these operations when the source machines are deterministic 1-LAs. In this case, by studying suitable witness languages, we prove that the costs become exponential.

3.1 1DFAs as Source Machines

By exploiting the capabilities of deterministic 1-LAs of rewriting the tape and scanning it in a two-way fashion, here we show that the product and the star of languages accepted by 1DFAs can be recognized by deterministic 1-LAs of polynomial size.

Product We start by describing how to obtain a deterministic 1-LA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepting the concatenation of the languages accepted by two 1DFAs $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ and $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$, in such a way that the size of $\mathcal{A}$ is polynomial in the sizes of $\mathcal{A}'$ and $\mathcal{A}''$. Let $Q' = \{q_0', q_1', \ldots, q_{n'-1}'\}$ and $Q'' = \{q_0'', q_1'', \ldots, q_{n''-1}''\}$. Hence $n' = \#Q'$ and $n'' = \#Q''$.

Let us start by briefly recalling how a 1DFA accepting $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ can work. It simulates $\mathcal{A}'$ on the whole input word and, every time a final state is entered, it starts a parallel simulation of the automaton $\mathcal{A}''$ on the remaining input suffix. When the end of the input is reached, if some computation of $\mathcal{A}''$ is in a final state, then the

1DFA accepts. Since the simulating 1DFA keeps in its finite control, at the same time, a state of $\mathcal{A}'$ and a set of states reached by all the parallel simulations of $\mathcal{A}''$, its size is $\Theta(n' \cdot 2^{n''})$. As shown by Yu, Zhuang, and Salomaa, this size is optimal, namely, in the worst case it cannot be reduced [5].

Here, our goal is to avoid the exponential blowup in size by exploiting the rewriting capability of 1-LAs. To this end, $\mathcal{A}$ still simulates the behavior of $\mathcal{A}'$ by using a state component of size n', and marks the cells from which the simulations of $\mathcal{A}''$ can start, that are the cells next to the ones $\mathcal{A}'$ enters some accepting state. So the simulation can be executed in a sequential rather than parallel way. Moreover, instead of storing the set of states reached by the simulations of $\mathcal{A}''$ in the finite control, $\mathcal{A}$ encodes and writes it along the tape. This information is then accessed, using the ability of 1-LAs of scanning the tape in a two-way fashion, to start and recover the simulations of $\mathcal{A}''$.

In order to encode the set of states reached by the computations of $\mathcal{A}''$, the tape is logically divided into blocks of n'' cells (possibly with a final shorter block). Thus, the i-th cell of each block is marked with ✔ if the state q_i'' is reached by some simulation of $\mathcal{A}''$ ending in the last cell *before* the block, otherwise it is marked with ✗.

The written information is organized into three tracks. In particular, for each frozen cell:

- The first track contains a copy of the input symbol originally contained in the cell before the rewriting, so that it can be still accessed during the simulations of $\mathcal{A}''$;
- The second track contains a marker indicating whether (✔) or not (✗) the automaton $\mathcal{A}'$ has entered an accepting state *right before* reading the cell, i.e., by reading the input prefix which ends in the cell immediately to the left. So that for any cell containing ✔ a simulation of $\mathcal{A}''$ can be started;
- The third track contains a marker indicating whether (✔) or not (✗) the corresponding states are reachable by some simulation of $\mathcal{A}''$, as explained above.

Example 1 In this example we consider the product between the two automata depicted in Fig. 1. In the same figure, the contents of the tape at the end of the computation on input *aacaabaacabbaaabb* of the deterministic 1-LA for the product obtained according to the construction presented above is shown. It is possible to notice that:

- *On the second track*, the cells in positions 4 of the first and of the second block are marked with ✔, in fact the prefixes of length 4 (*aaca*) and 10 (*aacaabaaca*) are accepted by the 1DFA on the left.
- *On the third track*, the cell in position 1 of the second block and the cells in positions 2 and 3 of the third block are marked with ✔. Indeed, starting the simulation of the automaton on the right from the cell in position 4 of the first block, the automaton reaches the state q_1'' at the end of the first block and the state q_3'' at the end of the second block, while, starting the simulation of the automaton on the right from the cell in position 4 of the second block, the automaton reaches the state q_2'' at the end of the second block.

In this case, the simulating deterministic 1-LA accepts the input. In fact, the string *aaca* is accepted by the automaton on the left, and the string *abaacabbaaabb* is accepted by the one on the right. Hence, the accepting simulation of the automaton on the right is the one starting from the cell in position 4 of the first block. At the end of

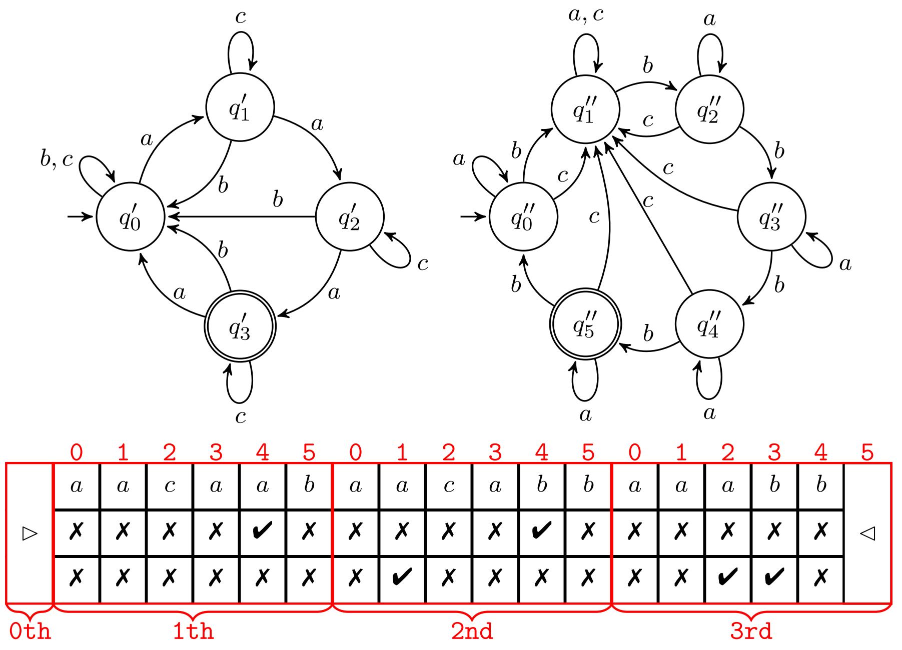

Fig. 1 Two 1DFAs and the contents of the tape at the end of the simulation of the product of their languages on input *aacaabaacabbaaabb*. The red rectangles indicate the logical division in blocks of the tape. Above each cell it is indicated its position relative to the block it is contained in

the first block, thus after reading the word ab from the initial state q''_0, the automaton enters the state q''_1, from which, reading the word $aacabb$, it reaches the state q''_3 at the end of the second block. Finally, from q''_3, reading the suffix $aaabb$ of the input, contained in the third block, the automaton enters the final state q''_5 and accepts.

To make the storing and the recovering of the information about the simulation of the automaton $\mathcal{A}''$ possible while keeping the cost of the simulation polynomial in the size of the simulated devices, the behavior of the simulating 1- LA will be restricted to virtual windows of length $2n''$ that cover two successive blocks of cells. The *right block* covered by a window contains, in some position, the leftmost cell that has not been overwritten so far, to which we refer as relative *frontier*. A typical situation is depicted in Fig. 2, in which $x \in (\Gamma \setminus \Sigma)^{n''} \cup \{\triangleright\}$ indicates the *left block* covered by a window, while the contents of the right block of the window can be decomposed into yu, with $y \in (\Gamma \setminus \Sigma)^*$ and $u \in \Sigma^* \cup \Sigma^*\triangleleft$, such that $|yu| = n''$ unless, possibly, $\triangleleft$ occurs in u, in which case $|yu| \leq n''$. The frontier is on the first position of u. We shall refer to the positions relative to the current window as pairs in $\{0, 1, \ldots, n'' - 1\} \times \{\text{L}, \text{R}\}$, where the pairs whose second element is L (resp., R) denote the left (resp., right) block of the window.

We now present some details on how $\mathcal{A}$ recognizes $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ (see Procedure 1). The details of the cost of this simulation in terms of number of states are discussed in the proof of Theorem 1. The deterministic 1- LA stores in its finite control the position of the frontier in the right block of the window (relativeFrontier), the relative position

Procedure 1 `product`($\mathcal{A}'$, $\mathcal{A}''$)
Given two 1DFAs $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ and $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$, the deterministic 1-LA implementing this procedure accepts the language $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}')$.

```
1  frontierState ← $q_0'$
2  relativeFrontier ← 0
3  relativePosition ← $(0, \text{R})$
4  isFinal ← $q_0' \in F'$
5  stateReachable ← FALSE
6  while current symbol $\neq \lhd$ do
7      write(current symbol, isFinal, stateReachable)
8      relativeFrontier ← (relativeFrontier + 1) mod $\#Q''$
9      if relativeFrontier = 0 then relativePosition ← $(\#Q'' - 1, \text{L})$
10     frontierState ← $\delta'$(frontierState, current symbol)
11     isFinal ← frontierState $\in F'$
12     stateReachable ← checkReachability($\mathcal{A}''$, $\{q''_{\text{relativeFrontier}}\}$, $(\#Q'' - 1, \text{L})$)
13     move the head rightward until reaching position (relativeFrontier, R)
14 if checkReachability($\mathcal{A}''$, $F''$, (relativeFrontier, R) − 1)
     or isFinal and $q_0'' \in F''$ then ACCEPT
15 else REJECT
```

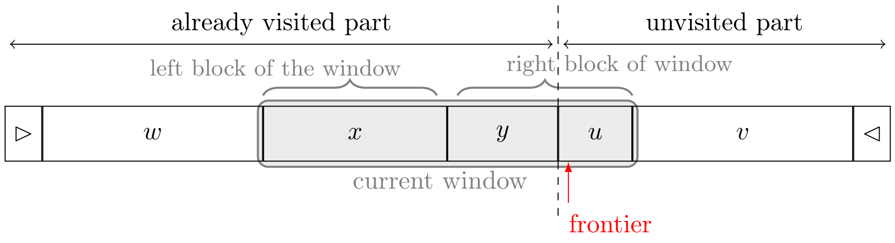

Fig. 2 Typical description of the window during a computation of $\mathcal{A}$: the current frontier occurs in the right block as first position of u, $w \in ((\Gamma \setminus \Sigma)^{n''})^*$, $x \in (\Gamma \setminus \Sigma)^{n''}$, $y \in (\Gamma \setminus \Sigma)^*$, $u \in \Sigma^+$, with $|yu| = n''$, and $v \in \Sigma^*$

of the head within the window (relativePosition),[1] and the state of the automaton $\mathcal{A}'$ (frontierState), which is updated every time the cell at the frontier is read. At the beginning of the computation, frontierState is initialized with q_0', and the relative frontier and the relative position both point at position 0 into the right block of the window (Lines 1, 2 and 3).

Let us now show how the 1-LA can overwrite each block, cell by cell, with an encoding of the set of states reached by all computations of $\mathcal{A}''$ at the end of the previous block and how it can mark the cells in which the simulations of $\mathcal{A}''$ start.

Let (i, R), $i \in \{0, \ldots, n'' - 1\}$, be the position of the frontier. the 1-LA has to gather the information to write in the leftmost cell that has not been rewritten yet. In particular, it has

[1] We assume that relativePosition is automatically updated according to the movements of the head of $\mathcal{A}$: its value is incremented (resp., decremented) when the head is moved to the right (resp., left). Moreover, if the value of relativePosition is $(n'' - 1, \text{L})$ (resp., $(0, \text{R})$) and a move to the right (resp., left) is performed, the new value will be $(0, \text{R})$ (resp., $(n'' - 1, \text{L})$).

1. To check whether the simulated automaton $\mathcal{A}'$ accepts the input scanned so far: This can be easily done by using the state component devoted to the simulation of $\mathcal{A}'$ for simulating a move of $\mathcal{A}'$ on the current input symbol and verify whether it enters a state in F' (Lines 10 and 11). In that case, $\mathcal{A}$ will write ✔ on the second track, ✗ otherwise.
2. To check whether the state q''_i can be reached by some computation of $\mathcal{A}''$ before entering the (first cell of the) right block of the current window: This operation is split into two phases (see Procedure 2). First, the 1-LA starts (from the initial state q''_0) the computations of $\mathcal{A}''$ from each cell of the left block whose second track contains ✔. Then, it recovers, in turn, the computations of $\mathcal{A}''$ from the states indicated in the third track of the cells of the left block, starting from the leftmost position of the window, i.e., relative position $(0, \text{L})$. If, during these two phases, the computation of $\mathcal{A}''$ reaches the state q''_i after simulating the transition on the symbol in the last cell of the left block, i.e., relative position $(n'' - 1, \text{L})$, the simulating automaton has to write ✔ in the third track, ✗ otherwise.

Procedure 2 `checkReachability(`$\mathcal{A}''$, targetStates, lastPosition`)`
Checks whether some state of targetStates can be reached either starting a simulation of the automaton $\mathcal{A}'' = (Q'', \Sigma, \delta'', q''_0, F'')$ (from q''_0) from some marked cell of the second track to lastPosition or recovering the simulation of $\mathcal{A}''$ (from the states encoded in the third track) from the beginning of the window to lastPosition.

```
16 move the head leftward until reaching relative position $(0, \text{L})$ or $\triangleright$
17 for cellPosition $\leftarrow$ relativePosition to lastPosition do
18     move the head until reaching relative position cellPosition
19     if the second track of the tape contains ✔ then
20         if simulate($\mathcal{A}''$, $q''_0$, lastPosition) $\in$ targetStates then
21             return TRUE
22 move the head leftward until reaching relative position $(0, \text{L})$ or $\triangleright$
23 for cellPosition $\leftarrow$ relativePosition to $(\#Q'' - 1, \text{L})$ do
24     move the head until reaching relative position cellPosition
25     if the third track of the tape contains ✔ then
26         move the head rightward until reaching position $(0, \text{L})$
27         if simulate($\mathcal{A}''$, $q''_{\text{cellPosition}}$, lastPosition) $\in$ targetStates then
28             return TRUE
29 return FALSE
```

After gathering this information, the 1-LA moves the head to the frontier (Line 13) and overwrites[2] the cell (Line 7); the frontier is moved to the next cell (Line 8). When the last cell of the window is overwritten, the window shifted forward of one block (i.e., it is shifted $n'' - 1$ cells to the left), so the right block becomes the left one and the frontier points at position $(0, \text{R})$ (Line 9).

When the machine detects the end of the input, indicated by the right end-marker $\triangleleft$ (Line 6), it has to check whether some simulation of $\mathcal{A}''$ halts in some accepting state.

[2] With the function `write`, the 1-LA overwrites the cell currently scanned by the head of the machine with the three arguments of the function, writing one argument per tape track.

This can be done with the same approach described in Item 2, but the two procedures of the two phases continue the simulations until the last cell of the input (i.e., the position to the left of the frontier, that is position (relativeFrontier, R) − 1) rather than stopping in position $(n'' - 1, \text{L})$. The deterministic 1- LA accepts if, during the two phases, some state in F'' is reached at the end of the input or if the simulated state of $\mathcal{A}'$ is final and the initial state of $\mathcal{A}''$ is final as well (Line 14).

Example 2 Let us now consider a detailed portion of the computation of the deterministic 1- LA presented in Example 1. In particular, Fig. 3 shows how the simulating machine recovers the information to write on the third track of the leftmost unvisited cell. This corresponds to one call to `checkReachability`. For ease of presentation, we shall call $\mathcal{A}'$ and $\mathcal{A}''$ the 1DFAs depicted in Fig. 1 on the left and on the right, respectively.

First, the simulating device moves its head to the left to the cell 0 of the left block. Then, moving rightward, it looks for cells that have a marker ✔ on the second track, meaning that $\mathcal{A}'$ reached a final state before scanning that cell, and so a simulation of $\mathcal{A}''$ can start. In this case, the only ✔ in the second track is in the cell number 4. Hence, the deterministic 1- LA simulates $\mathcal{A}''$ from the initial state q_0'' and checks whether the state with index equal to the frontier, which is 3 in this case, is reached reading the symbols of the block from cell 4. Since $\mathcal{A}''$ reaches $q_2'' \neq q_3''$ reading the string bb from q_0'' and there are no other ✔ on the second track of the left block, the deterministic 1- LA moves its head back to the cell 0 of the block. Then, it looks for markers ✔ on the third track, the leftmost being in the cell 1. Therefore, the deterministic 1- LA checks whether $\mathcal{A}''$ reaches q_3'' reading the string $aacabb$ contained in the left block starting from the state q_1''. In this case, the check is successful, so the automaton moves to the cell at the frontier to write ✔ on the third track (cfr. Fig. 1).

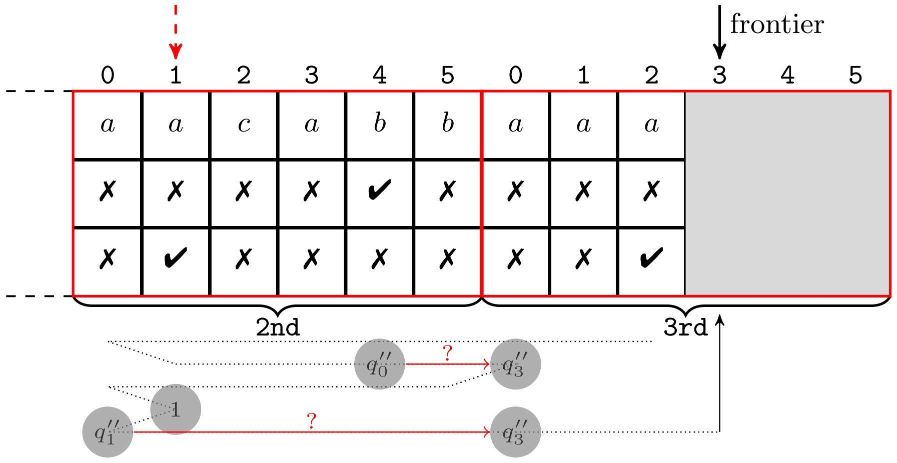

Fig. 3 A detail on the computation of the contents that the deterministic 1- LA presented in Fig. 1 writes on the third track, in the third cell of the third block

By computing the size of the resulting deterministic 1- LA $\mathcal{A}$, we are able to state our result on the acceptance of the product of two regular languages (represented by 1DFAs) by a deterministic 1- LA.

Theorem 1 *Let* $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ *and* $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$ *be two* 1DFA*s. Then there exists a deterministic* 1- LA *accepting* $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ *with* $O(\#Q' \cdot \#Q''^4)$ *states and* $5\#\Sigma + 2$ *working symbols.*

Proof Let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be the deterministic 1- LA implementing Procedure 1 described above. Since, during the computation, the tape cell is split into three tracks, the working alphabet of $\mathcal{A}$ is

$$\Gamma = \Sigma_{\triangleright,\triangleleft} \cup (\Sigma \times \{✔, ✗\} \times \{✔, ✗\}),$$

whose size is $\#\Sigma_{\triangleright,\triangleleft} + 4\#\Sigma$.

Plus, the automaton has to store in its finite control the following components:

- The simulated state of $\mathcal{A}'$ (frontierState), whose size is $\#Q'$;
- The simulated state of $\mathcal{A}''$ (state) in Procedure 3, that is an auxiliary procedure to simulate the automaton passed as argument (in this case $\mathcal{A}''$) on the portion of the tape between the cell where the head of the automaton is when the procedure is called and lastPosition starting from state. This component has size $\#Q''$;
- The position of the head (relativePosition) relative to the current window, of size $2\#Q''$, and the position of the frontier (relativeFrontier) of size $\#Q''$;
- The counter cellPosition of Procedure 2 of size $\#Q''$;
- Some additional state components (of constant size) used to keep track of the execution flow of the procedure.

Therefore, the number of states of $\mathcal{A}$ is $O(\#Q' \cdot \#Q''^4)$. □

Kleene Star Let us now turn our attention to the star operation. Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ be a 1DFA. A deterministic 1- LA $\mathcal{N}$ for $\mathcal{L}(\mathcal{A})^*$ can implement an approach similar to the one used for the product, so we now illustrate the main differences (in the following we shall refer to Procedure 4). The details of the cost of this simulation are given in the proof of Theorem 2.

In this case, the only automaton to be simulated is $\mathcal{A}$. The first simulation is started from the leftmost input cell. Every time a (simulated) final state is entered by some simulated computation of $\mathcal{A}$, $\mathcal{N}$ starts a new simulation. If, at the end of the input, some simulation reaches a final state, then the 1- LA accepts.

To implement this strategy, the tape of $\mathcal{N}$ is still organized as for the simulation of the product, i.e., it is logically split into blocks of size $\#Q$ and three tracks are used to store a copy of the input, indicating whether or not some simulation of $\mathcal{A}$ has entered an accepting state on the previous cell, and a marker indicating whether or not the corresponding states are reachable by some simulation of $\mathcal{A}$.

Before entering a new cell, $\mathcal{N}$ first checks whether the prefix already visited is in $\mathcal{L}(\mathcal{A})^*$. This is done by recovering the simulations of $\mathcal{A}$ (from the states encoded on the third track) and starting the new ones (from the cells of the second track marked

Procedure 3 `simulate(`$\mathcal{A}''$, state, lastPosition`)`
Simulates the automaton $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$ on the portion of the tape from relativePosition to lastPosition starting from state. If, along the computation, the transition function is not defined for the simulated state and the scanned symbol, it returns $\perp$.

```
30 repeat
31     state ← $\delta$(state, current symbol)
32     if relativePosition = lastPosition then return state
33     move the head one cell to the right
34 until state = $\perp$
35 return state
```

Procedure 4 `KleeneStar(`$\mathcal{A}$`)`
Given a 1DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, the deterministic 1- LA implementing this procedure accepts the language $\mathcal{L}(\mathcal{A})^*$.

```
36 relativeFrontier ← 0
37 relativePosition ← (0, R)
38 isFinal ← TRUE
39 stateReachable ← FALSE
40 while current symbol $\neq \lhd$ do
41     write(current symbol, isFinal, stateReachable)
42     relativeFrontier ← (relativeFrontier + 1) mod $\#Q$
43     if relativeFrontier = 0 then relativePosition ← ($\#Q - 1$, L)
44     isFinal ← checkReachability($\mathcal{A}$, $F$, (relativeFrontier, R) − 1)
45     stateReachable ← checkReachability($\mathcal{A}$, {$q_{\text{relativeFrontier}}$}, ($\#Q - 1$, L))
46     move the head rightward until reaching position (relativeFrontier, R)
47 if isFinal then ACCEPT
48 else REJECT
```

with ✔), and checking whether some of them reaches a state in F (Line 44). After that, $\mathcal{N}$ checks whether the state whose index is relativeFrontier is reached at the end of the previous block by some simulation (Line 45). Once this information is computed, the automaton moves the head on the cell at the frontier and overwrites it (Line 41).[2]

When the right end-marker is reached (Line 40), $\mathcal{N}$ only needs to check whether some simulation is in a final state and, in that case, it accepts (Line 47).

Theorem 2 *Let* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *be a* 1DFA. *Then there exists a deterministic* 1-LA *accepting* $\mathcal{L}(\mathcal{A})^*$ *with* $O(\#Q^4)$ *states and* $5\#\Sigma + 2$ *working symbols.*

Proof Let $\mathcal{N} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be the deterministic 1- LA implementing Procedure 1 described above. Also in this case the working alphabet is

$$\Gamma = \Sigma_{\rhd,\lhd} \cup (\Sigma \times \{✔,✗\} \times \{✔,✗\}),$$

whose size is $\#\Sigma_{\rhd,\lhd} + 4\#\Sigma$. Plus, the automaton has to store in its finite control the following information:

- The simulated state of $\mathcal{A}$ (state) in Procedure 3 ($\#Q$ possible values);

- The position of the head (relativePosition) relative to the current window and the position of the frontier (relativeFrontier) ($2\#Q$ and $\#Q$ possible values, respectively);
- The counter cellPosition of Procedure 2 ($\#Q$ possible values);
- Some additional state components (of constant size) used to keep track of the execution flow of the procedure.

Therefore, the number of states of $\mathcal{A}$ is $O(\#Q^4)$. □

3.2 Deterministic 1-LAs and Sweeping 2DFAs as Source Machines

We now focus on the size costs of the operations of product and star on deterministic 1-LAs. An immediate approach is to convert the source deterministic 1- LAs to 1DFAs, and then to apply the constructions shown in the previous section. Since converting deterministic 1- LAs into 1DFAs costs exponential in size [11], this procedure yields exponential-size deterministic 1- LAs for the two operations we are considering. Here, we show that this strategy cannot be improved, in fact we prove exponential lower bounds for these operations that match the blowup in size necessary to convert deterministic 1- LAs into 1DFAs. Actually, we prove even a stronger result by observing that the same holds if the source machines are sweeping 2DFAs.

For any fixed integer $k \geq 2$, let us consider the language of the strings obtained by concatenating at least two blocks of length k, in which the first and the last blocks are equal, i.e., $L_k = \{w\{a, b\}^{jk}w \mid j \geq 0, w \in \{a, b\}^k\}$.

Let us start by describing a sweeping 2DFA $\mathcal{A}_k$ accepting L_k. Notice that, each computation of $\mathcal{A}_k$ is a sequence of traversals (or *sweeps*) of the tape.

During the first traversal of the tape from left to right, using a counter modulo k, $\mathcal{A}_k$ checks that the length of the input word is a multiple of k. If this is not the case, then $\mathcal{A}_k$ rejects. Otherwise, $\mathcal{A}_k$ has to compare the first and the last block of the input. This task can be done using the following strategy. The automaton performs at most k left-to-right traversals of the input $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{k-1}$, interleaved with right-to-left sweeps. The purpose of the traversal $\mathcal{S}_i$ is to check whether the $(i + 1)$-th symbols of the first and of the last block coincide. This can be done using a counter c modulo k stored in the finite control. The counter is set in such a way that, during the sweep $\mathcal{S}_i$, it contains 0 while visiting the cell $i + 1$ of the first block. Furthermore, the machine saves in the finite control the symbol therein contained. While moving the head to the right, the counter is incremented for each visited cell.

The machine inspects each cell that is reached when the value of the counter is 0, by comparing its symbol with the one stored in the finite control. If the two values are equal, the value TRUE is assigned to a Boolean variable matched, FALSE otherwise.

When the right end-marker is reached, if matched contains FALSE, $\mathcal{A}_k$ stops and rejects, because the last block is not equal to the first one. Otherwise, $\mathcal{A}_k$ starts a traversal from right to left, decrementing the counter c at each move. Hence, in a traversal from left to right and in the next one from right to left, a same cell is reached with the same value of c. In particular, in the sweep $\mathcal{S}_i$ and in the next right-to-left traversal, each cell k such that $k \bmod i = 0$ is reached exactly when the counter is 0.

The sweep $\mathcal{S}_0$ starts with the head on the first tape cell and 0 in the counter. At the end of a right-to-left traversal, when the left end-marker is reached, if matched is TRUE, then the head is moved to the first input cell, *without changing the counter*. In this way the counter is properly set for the next sweep.

This procedure is repeated until the right end-marker is reached with the counter containing 1. In this case all the positions have been inspected and, if the variable matched contains TRUE, the automaton accepts.

Since the only information the automaton $\mathcal{A}_k$ has to store in its control is the counter modulo k and the variable matched, it is possible to implement it with a number of states linear in k.

Let us now consider the language L_k^2, namely the product of L_k with itself. In this case, the ability of moving the head in a two-way fashion does not come in handy. This is because, ideally, the machine cannot know in advance where to *"split"* the input string into two parts belonging to L_k, or, in other words, which part of the input belongs to the first occurrence of L_k and which part belongs to the second one. Therefore, it would have to store in its finite control the word of length k immediately following any block matching the first one, to compare it with the last block. This idea is confirmed in the proof of the next theorem, where, using a distinguishability argument, it is proved that each 1DFA for L_k^2 needs a number of states at least double exponential in k. Since it is known that the conversion of deterministic 1- LAs into 1DFAs costs exponential [11], then we can conclude that the size of each deterministic 1- LAs for L_k^2 has to be at least exponential in k.

Theorem 3 *For any integer $k \geq 2$,*

- *There exist two sweeping* 2DFA*s $\mathcal{A}'$ and $\mathcal{A}''$ of size linear in k such that any deterministic* 1- LA *accepting $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ needs size at least exponential in k.*
- *There exists a sweeping* 2DFA *$\mathcal{A}$ of size linear in k such that any deterministic* 1- LA *accepting $\mathcal{L}(\mathcal{A})^*$ needs size exponential in k.*

Proof Let us consider the language L_k. Using the approach described above, it is possible to recognize L_k with a 2DFA of size linear in k.

Let us turn our attention to the language $L_k \cdot L_k = L_k^2$. We prove a lower bound for the size required by any 1DFA accepting it. To this end, we are going to describe a set of pairwise distinguishable strings for this language. We remind the reader that two strings x, y are *distinguishable* with respect to a language L when there is a string z such that exactly one of the two strings xz and yz belongs to L. The cardinality of any set of strings which are pairwise distinguishable with respect to L gives a lower bound for the number of states of each 1DFA accepting L.

Let us consider the list $x_1, x_2, \ldots, x_N$, with $N = 2^k$, of all the strings in $\{a, b\}^k$ in some fixed order.

With each nonempty subset $S = \{i_1, i_2, \ldots, i_n\}$, $1 \leq i_1 < i_2 < \ldots < i_n \leq N$, we associate the string $w_S = x_{i_1}x_{i_1}x_{i_1}x_{i_2}x_{i_1}x_{i_3}x_{i_1} \cdots x_{i_n}x_{i_1}$. In other words, w_S is the ordered sequence of factors corresponding to the elements of S interleaved with occurrences of x_{i_1}. In particular, x_{i_1} occurs at the beginning of the sequence and after every factor. Furthermore we define $w_\emptyset = \varepsilon$.

Now, consider two sets $S, T \subseteq \{1, 2, \ldots, N\}$, with $S \neq T$. Since S and T are different, there is a string $x \in \{a, b\}^k$ contained exactly in one of them. Without loss of generality, assume $x \in S$ and $x \notin T$. We prove that x distinguishes w_S and w_T by showing that $w_S x \in L_k^2$ and $w_T x \notin L_k^2$. In fact, let $S = \{i_1, i_2, \ldots, i_n\}$ and $x = x_{i_\ell}$, with $1 \leq \ell \leq n$. If $\ell > 1$, then we decompose w_S as $w'_S \cdot w''_S$, where $w'_S = x_{i_1}x_{i_1}x_{i_1}x_{i_2}x_{i_1} \cdots x_{i_{\ell-1}}x_{i_1}$, $w''_S = x_{i_\ell}x_{i_1}x_{i_{\ell+1}}x_{i_1} \cdots x_{i_n}x_{i_1}$, and we notice that $w'_S \in L_k$, $w''_S x = w''_S x_{i_\ell} \in L_k$ and hence $w_S x \in L_k^2$. In a similar way, when $\ell = 1$, we take $w'_S = x_{i_1}x_{i_1}$ and $w''_S = x_{i_1}x_{i_2}x_{i_1} \cdots x_{i_n}x_{i_1}$, thus concluding that $w_S x \in L_k^2$. On the other hand, the string $w_T x$ is not in L_k^2 because $x = x_{i_\ell}$ does not occur in the factorization of w_T as concatenation of factors of length k. Actually, for the same reason, $w_T x \notin L_k^*$. Since $w_S \in L_k^2 \subseteq L_k^*$, this observation easily allows to extend our result to the Kleene star operation. Hence x distinguishes w_S and w_T with respect to both the languages L_k^2 and L_k^*.

Since there are 2^N subsets of $\{1, 2, \ldots, N\}$, each 1DFA accepting $L_k \cdot L_k$ and each 1DFA accepting L_k^* needs at least 2^{2^k} states.

To conclude the proof, we point out that each n-state deterministic 1-LA can be simulated by a 1DFA with at most $n(n+1)^n$ states [11]. Hence, if one of the languages $L_k \cdot L_k$ or L_k^* is accepted by a deterministic 1-LA with n states, $n > 2$, then $2^{2^k} \leq n(n+1)^n \leq 2^{\log_2 n + n \log_2(n+1)} \leq 2^{n^2}$, thus implying $2^k \leq n^2$ and, hence, $n \geq 2^{k/2}$. This allows to conclude that, to accept $L_k \cdot L_k$ and L_k^*, deterministic 1-LAs require a number of states at least exponential in k. □

We point out that in [9] exponential lower bounds in the case source and target machines are 2DFAs have been proved for product and star. The language therein used is on a 7-symbol input alphabet, while here we used a binary input alphabet. Furthermore, since 2DFAs are a particular case of deterministic 1-LAs, our result is a generalization of that in [9], because it holds for a larger class of target machines.

By considering as source machines deterministic 1-LAs, we immediately get:

Corollary 4 *For any integer $k \geq 2$,*

- *There exist two deterministic 1-LAs $\mathcal{A}'$ and $\mathcal{A}''$ of size linear in k such that any deterministic 1-LA accepting $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ needs size at least exponential in k.*
- *There exists a deterministic 1-LA $\mathcal{A}$ of size linear in k such that any deterministic 1-LA accepting $\mathcal{L}(\mathcal{A})^*$ needs size exponential in k.*

In conclusion, starting from two deterministic 1-LAs $\mathcal{A}'$ and $\mathcal{A}''$ accepting the languages L' and L'' (resp., from a deterministic 1-LA $\mathcal{A}$ accepting a language L), a deterministic 1-LA for $L' \cdot L''$ (resp., L^*) can be obtained by converting $\mathcal{A}'$ and $\mathcal{A}''$ (resp., $\mathcal{A}$) into 1DFAs, and then applying the transformation of Theorem 1 (resp., Theorem 2). These constructions are optimal, in fact we proved that the exponential blowup in size due to the conversion into 1DFAs cannot be avoided.

4 Union, Intersection, and Complementation

We continue our investigation by analyzing the operations of union, intersection, and complementation using deterministic 1-LAs as target machines. As in the previous section, we start by considering 1DFAs as source devices and then we switch to the case of deterministic 1-LAs.

4.1 1DFAs as Source Machines

It is well known that for union, intersection, and complementation, the simulations are easier than the ones for product and star. Even if the target machines are 1DFAs, it is possible to obtain polynomial-size simulating devices.

Union and Intersection For union and intersection, the resulting 1DFA is obtained by simulating in parallel the 1DFAs accepting the two given languages. Hence, it has a number of states which is the product of the number of states of the two given 1DFAs. This cannot be improved in the worst case [5].

When the target machine is a 2DFA, it can perform the simulation of the first 1DFA during a sweep from left to right, and then, when the end of the input is reached, it can move the head at the beginning of the tape to start the simulation of the second 1DFA. In the case of the union, the resulting 2DFA accepts if the simulation of at least one 1DFA accepts, while, in the case of the intersection, it accepts if both the simulated 1DFAs accept.

The 2DFA implementing this simulation only needs to store, in its state, the copies of the simulated machines, plus one state used to move backward the head at the end of the first simulation. So the total number of states of the simulating devices is 1 plus the sum of the numbers of states of the two simulated 1DFAs. For the sake of completeness we formally present this easy construction in the next theorem.

Theorem 5 *Let $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ and $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$ be two 1DFAs. Then there exist*

- *a 2DFA for the language $\mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}'')$ and*
- *a 2DFA for the language $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}'')$*

with $\#Q' + \#Q'' + 1$ states.

Proof A 2DFA $(Q, \Sigma, \delta, q_I, F' \cup F'')$ accepting the language $\mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}'')$ can be obtained as follows. Without loss of generality, we assume $Q' \cap Q'' = \emptyset$. The set of states is $Q = Q' \cup Q'' \cup \{q_B\}$, and the transition function is defined as follows:

1. $\delta(p, a) = (q, +1)$, for each $a \in \Sigma$ and $p, q \in Q'$ such that $\delta'(p, a) = q$ or $p, q \in Q''$ such that $\delta''(p, a) = q$;
2. $\delta(q, \lhd) = (q_B, -1)$, for each $q \in Q' \setminus F'$;
3. $\delta(q_B, a) = (q, -1)$, for each $a \in \Sigma$;
4. $\delta(q_B, \rhd) = (q_0'', +1)$;
5. $\delta(q, \lhd) = (q, +1)$, for each $q \in F' \cup F''$.

Notice that, when the input is accepted by $\mathcal{A}'$, the automaton stops the computation and accepts after the simulation of $\mathcal{A}'$.

A two-way automaton for the language $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}'')$ can be defined in a very similar way. It only differs in the set of final states which coincides with F'', in transitions of Item 2, due to the fact that the 2DFA has to simulate the second 1DFA only if the first one accepts the input, and in transitions of Item 5 which lead to acceptance only if the simulation of $\mathcal{A}''$ leads to acceptance:

2. $\delta(q, \lhd) = (q_B, -1)$, for each $q \in F'$.
5. $\delta(q, \lhd) = (q, +1)$, for each $q \in F''$.

□

From these constructions we can directly obtain equivalent deterministic 1- LAs that, during the first sweep, simply overwrite each tape cell with a copy of the symbol it originally contains.

Corollary 6 *Let $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ and $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$ be two* 1DFA*s. Then there exist*

- *a deterministic* 1- LA *for the language $\mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}'')$ and*
- *a deterministic* 1- LA *for the language $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}'')$*

with $\#Q' + \#Q'' + 1$ states and $2\#\Sigma + 2$ working symbols.

Observe that, while using 1DFAs as target devices requires to simulate the source machines in parallel (because they are one-way), thus yielding devices with a number of states equal to the product of the states of the simulated 1DFAs [5], here we obtained deterministic 1- LAs whose sizes are equal to the sum of the sizes of the simulated machines.

Complementation Even this case is trivial and it is included for the sake of completeness. A deterministic 1- LA for the complement can be obtained with a construction analogous to the standard one used for obtaining a 1DFA for the complement, i.e., just by complementing the set of the accepting states, provided that the transition function is total.

Theorem 7 *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a* 1DFA. *Then there exists a deterministic* 1- LA *with $\#Q + 1$ states and $\#\Sigma + 3$ working symbols which accepts $\mathcal{L}(\mathcal{A})^c$. If δ is total then the number of states of the resulting deterministic* 1- LA *reduces to $\#Q$.*

Proof A deterministic 1- LA accepting the language $\mathcal{L}(\mathcal{A})^c$ can be defined as $(Q', \Sigma, \Sigma_{\rhd,\lhd} \cup \{\sharp\}, \delta', q_0, Q' \setminus F)$, where $\sharp$ is a new symbol not in Σ, $Q' = Q \cup \{r\}$, with r not in Q, and δ' contains the following transitions:

1. $\delta'(p, a) = (q, \sharp, +1)$, for each $a \in \Sigma$ and $p, q \in Q$ such that $\delta(p, a) = q$;
2. $\delta'(q, \lhd) = (q, \lhd, +1)$, for each $q \in Q' \setminus F$,
3. $\delta'(q, a) = (r, \sharp, +1)$, for each $a \in \Sigma$ and $q \in Q'$ such that $\delta(q, a)$ is not defined.
4. $\delta'(r, a) = (r, \sharp, +1)$, for each $a \in \Sigma$.

If δ is total then $Q' = Q$ and Items 3 and 4 are dropped.

4.2 Deterministic 1-LAs as Source Machines: Polynomial-Size Simulations

More interesting constructions and results can be obtained if both source and target machines are deterministic 1- LAs. By making use of a polynomial-size linear-time simulation of 1- LAs presented in [13]: Given a 1- LA, paying a polynomial growth in size, it is possible to obtain an equivalent one that works in a time which is linear in the input length. The idea of the construction is similar to the technique used for the simulation of the product in Section 3: the simulating device works on a virtual window of fixed size that is shifted along the tape in a one-way manner. More precisely, there exists a constant K not depending on the input length such that, in the computations of the simulating 1- LAs, for any two tape cells at distance K, the leftmost one cannot be visited after having visited the rightmost one. Along each window it is stored the information useful to simulate the behavior of the 1- LA on the cells to the left of the window without accessing such portion of the tape anymore. In this way, it is possible to bound the number of visits to each cell.

Lemma 8 ([13, Theorem 1 and Lemma 6]) *For each deterministic* 1- LA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *there exists an equivalent deterministic* 1- LA $\mathcal{A}'$ *working in linear time with* $O(\#Q^4)$ *states and* $(\#Q + 1) \cdot \#(\Gamma \setminus \Sigma)$ *working symbols.*

Using this result, we are able to obtain deterministic 1- LAs for union, intersection and complement whose sizes are polynomial in the description of the simulated deterministic 1- LAs.

Union and Intersection Let us briefly recall how the linear-time simulation of Lemma 8 works (for further details we address the reader to [13]). The construction is inspired by the conversion of two-way automata into one-way 1DFAs [7], that was also used to convert 1- LAs into one-way finite automata [11]. To simulate a two-way automaton in a one-way fashion, at any point during the computation, the simulating 1DFA stores into its internal state the so called *Shepherdson tables*, that contain the information about the possible two-way computations on the portion of the tape to the left of the head. In this way, the 1DFA does not need to move its head backward, but has to keep updated the tables in its state. Since the number of possible tables is exponential, this technique produces an exponential blowup in size.

In the case of 1- LAs, to perform a linear-time simulation, the simulating 1- LA fills each window with an encoding of the table describing the behavior of the simulated 1- LA on the portion of the tape *preceding* the window. This allows, on the one hand, to perform the simulation visiting the cells of each window a constant number of times and, on the other hand, to avoid the exponential blowup in size of the simulating device.

The idea of the simulation is that, in order to fill the cells of the virtual windows with useful information, only the cells of the window itself can be visited. In particular, every time the simulating device has to scan a new cell, i.e., the leftmost cell that has not been visited yet, it first recovers the information to write in the new cell only visiting the frozen cells in the current window, then it moves the head to the new cell and writes the computed information (on a separate track) along with the working symbol that the simulated device would write on that cell.

For the simulation of union and intersection of the languages accepted by two deterministic 1- LAs, the machines are simulated in parallel. In particular, two (possibly different) virtual windows are used and shifted independently. Before entering a new cell, the simulating device computes the information about the windows of the simulated deterministic 1- LAs (in this phase, only the cells of the two windows are visited: initially, the window of the first simulated device is visited and then, when the information has been gathered and stored in the finite control, the window of the second one is visited to compute and store the information about the second device). Then the new cell is entered and the stored information is saved (on two tracks of the tape), together with the symbols written by the simulated devices (on two extra tracks).

When the end of the input is reached, in the case of the union the simulating device accepts if at least one simulation accepts, in the case of the intersection it accepts if both the simulated devices accept.

Theorem 9 *Let $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q_0', F')$ and $\mathcal{A}'' = (Q'', \Sigma, \Gamma', \delta'', q_0'', F'')$ be two deterministic* 1- LA*s, $n' = \#Q'$, and $n'' = \#Q''$. Then there exist*

- *a deterministic* 1- LA *for the language $\mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}'')$ and*
- *a deterministic* 1- LA *for the language $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}'')$*

both with $O(n'^4 n''^4)$ states and $(n'+1) \cdot (n''+1) \cdot \#(\Gamma' \setminus \Sigma) \cdot \#(\Gamma'' \setminus \Sigma)$ working symbols.

Proof We can use Lemma 8 to perform a parallel linear-time simulation of $\mathcal{A}'$ and $\mathcal{A}''$. So, the simulating deterministic 1- LA has a state component of size $O(n'^4)$ to simulate $\mathcal{A}'$ and one of size $O(n''^4)$ for the simulation of $\mathcal{A}''$. Moreover, it uses 4 tracks: two for the tables of $\mathcal{A}'$ and $\mathcal{A}''$, for which, respectively, $(n'+1)$ and $(n''+1)$ symbols are used, and two tracks for the symbols $\Gamma' \setminus \Sigma$ and $\Gamma'' \setminus \Sigma$ written by $\mathcal{A}'$ and $\mathcal{A}''$, respectively. □

Complementation To accept the complement of the language accepted by a deterministic 1- LA $\mathcal{A}$, again Lemma 8 can be used to perform a linear-time (and therefore, halting) simulation of $\mathcal{A}$. The simulating deterministic 1- LA accepts if $\mathcal{A}$ enters a loop or if it is not in an accepting state at the end of its computation.

Theorem 10 *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic* 1- LA*. Then there exists a deterministic* 1- LA *with $O(\#Q^4)$ states and $(\#Q+1) \cdot \#(\Gamma' \setminus \Sigma)$ working symbols which accepts $\mathcal{L}(\mathcal{A})^c$.*

5 Reversal

The last operation we study is the reversal. Even in this case, the deterministic 1- LA for the reversal of the language accepted by a 1DFA $\mathcal{A}$ can be obtained by exploiting just the capability of the simulating machine of scanning the input in a two-way fashion, so, again, we first give our result for 2DFAs. Roughly, starting from the initial state of $\mathcal{A}$ with the head positioned on the last symbol of the input word, the machine accepts if, simulating the transitions of $\mathcal{A}$ scanning the input from right to left, a final state is entered when the head reaches the left end-marker. This approach yields a 2DFA with a number of states equal to the one of the simulated machine, plus two states for adjusting the position of the head along the tape at the beginning and at the end of the computation.

Theorem 11 *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a* 1DFA. *Then there exists one* 2DFA *with $\#Q + 2$ states which accepts $\mathcal{L}(\mathcal{A})^R$.*

Proof A 2DFA $\mathcal{A}^{\mathrm{R}}$ for the reversal of the language accepted by $\mathcal{A}$ works as follows. At the beginning of the computation it performs a sweep from left to right on the tape. When the head reaches the right end-marker, it performs a *backward simulation* of the automaton $\mathcal{A}$: It uses a copy of the state set of $\mathcal{A}$, and, every time it simulates a transition connecting two states, it performs a move to the left. If the backward simulation, starting from the (simulated) initial state of $\mathcal{A}$, leads to an accepting (simulated) state while reaching the left end-marker, then $\mathcal{A}^{\mathrm{R}}$ accepts (after traversing the whole input to reach the right end of the tape), otherwise it rejects.

Formally, $\mathcal{A}^{\mathrm{R}} = (Q', \Sigma, \delta', q_I, \{q_F\})$, where $Q' = Q \cup \{q_I, q_F\}$ and the transition function is defined as follows:

1. $\delta'(q_I, a) = (q_I, +1)$, for each $a \in \Sigma$;
2. $\delta'(q_I, \lhd) = (q_0, -1)$;
3. $\delta'(q, a) = (\delta(q, a), -1)$, for each $q \in Q$ and each $a \in \Sigma$;
4. $\delta'(q, \rhd) = (q_F, +1)$, for each $q \in F$;
5. $\delta'(q_F, a) = (q_F, +1)$, for each $a \in \Sigma \cup \{\lhd\}$. □

As a consequence, we are able to construct an equivalent deterministic 1- LA that uses the same strategy of the obtained 2DFA, with the only difference that, during the first sweep from left to right, it rewrites on each cell a copy of the symbol it scans.

Theorem 12 *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a* 1DFA. *Then there exists a deterministic* 1- LA *with $\#Q + 2$ states and $2\#\Sigma + 2$ working symbols which accepts $\mathcal{L}(\mathcal{A})^R$.*

In the case of deterministic 1- LAs, the reversal has an exponential cost in size. The exponential upper bound can be obtained by converting the deterministic 1- LA into a 1DFA and then applying Theorem 12. A matching exponential lower bound has been proved in [13, Theorem 4]. More precisely, it is shown that, for each integer $k \geq 2$, the deterministic 1- LA for the language

$$\{x_0 x_1 \cdots x_n \mid n \geq 1, x_i \in \{a, b\}^k \text{ for } i = 0, \ldots, n, \exists j > 0 \text{ s.t. } x_j = x_0\}$$

has size $O(k)$ while the deterministic 1- LA for its reversal has size $2^{O(k)}$.

Theorem 13 ([13, Theorem 4]) *For any integer $k \geq 2$, there exists a deterministic* 1- LA *$\mathcal{A}$ of size linear in k such that any deterministic* 1- LA *accepting $\mathcal{L}(\mathcal{A})^R$ needs size exponential in k.*

6 Conclusion

In this paper we studied the descriptional complexity of classical operations on regular languages using, as target devices, deterministic 1- LAs.

Table 2 The costs of operations of deterministic finite automata and deterministic 1-limited automata. or each operation (first column), the second to fifth columns show the sizes of the devices obtained by simulating the corresponding operation pplied to a source machine of size n accepting a language L (and to one of size m accepting L', in case of binary operations)

Op.	Construction					
	1DFA →1DFA	1DFA →D1-LA	D1-LA→D1-LA		sw. 2DFA →D1-LA	
	states	*states*	*states*	*symbols*	*states*	*symbols*
$L \cup L'$	nm	$n+m+1$	$O(n^4m^4)$	$O(nm\gamma\eta)$	$O(n^4m^4)$	$O(nm)$
$L \cap L'$	nm	$n+m+1$	$O(n^4m^4)$	$O(nm\gamma\eta)$	$O(n^4m^4)$	$O(nm)$
L^{C}	$n+1$	$n+1$	$O(n^4)$	$O(n\gamma)$	$O(n^4)$	$O(n)$
L^{R}	2^n	$n+2$	$2^{O(n)}$	$O(1)$	n	$O(1)$
$L \cdot L'$	$n2^m - 2^{m-1}$	$O(nm^4)$	$2^{O(n+m)}$	$O(1)$	$2^{O(n+m)}$	$O(1)$
L^*	$\frac{3}{4}2^n$	$O(n^4)$	$2^{O(n)}$	$O(1)$	$2^{O(n)}$	$O(1)$

The simulations considered in the second to fifth columns are, respectively, from 1DFAs (as source devices) into 1DFAs (as target devices), from 1DFAs into deterministic 1- LAs, from 1DFAs into 1DFAs, and from sweeping 2DFAs into deterministic 1- LAs. For the constructions of the third column, given a fixed input alphabet of the simulated 1DFAs, the number of working symbols of the resulting deterministic 1- LAs is constant. For the constructions of the fourth colum, the source devices have working alphabets of size γ (and η for binary operations)

It is interesting to notice that, if we consider operations between 1DFAs (as source devices) then we are able to create deterministic 1- LAs accepting the languages obtained by applying such operations that are smaller than the equivalent 1DFAs obtained by using standard constructions [5]. In particular, while the 1DFAs accepting the languages obtained by applying the operations of reversal, product, and Kleene star on the languages accepted by 1DFAs cost exponential, the constructions we provided yield equivalent deterministic 1- LAs whose sizes are only polynomial in the sizes of the source 1DFAs (cfr. second and third columns of Table 2).

On the other hand, the simulation of operations between deterministic 1- LAs by deterministic 1- LAs costs polynomial only in the case of union, intersection, and complementation. In the case of reversal, product, and star, however, we were able to find exponential lower bounds witnessing the fact that there is no smaller automaton than the one obtained by converting the simulated deterministic 1- LAs into 1DFAs first (obtaining exponentially larger machines), and then applying the corresponding (polynomial-size) language operation construction for obtaining a deterministic 1- LA (cfr. fourth column of Table 2). In the special case in which the source machines are sweeping 2DFAs, it is obvious that we can apply the same constructions. sFurthermore, exponential lower bounds still hold for product and star, while it is a trivial observation that the simulation for the reversal costs linear in size (cfr. fifth column of Table 2).

References

1. Pighizzini, G., Prigioniero, L., Sádovský, S.: Performing regular operations with 1-limited automata. In: DLT 2022, Proceedings. Lecture Notes in Computer Science, vol. 13257, pp. 239–250. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05578-2_19
2. Maslov, A.N.: Estimates of the number of states of finite automata. In: Doklady Akademii Nauk, vol. 194, pp. 1266–1268 (1970). Russian Academy of Sciences
3. Leiss, E.L.: Succint representation of regular languages by boolean automata. Theor. Comput. Sci. **13**, 323–330 (1981). https://doi.org/10.1016/S0304-3975(81)80005-9
4. Yu, S., Zhuang, Q.: On the state complexity of intersection of regular languages. SIGACT News **22**(3), 52–54 (1991). https://doi.org/10.1145/126537.126543
5. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. **125**(2), 315–328 (1994). https://doi.org/10.1016/0304-3975(92)00011-F
6. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev **3**(2), 114–125 (1959). https://doi.org/10.1147/rd.32.0114
7. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM J. Res. Dev. **3**(2), 198–200 (1959). https://doi.org/10.1147/rd.32.0198
8. Kunc, M., Okhotin, A.: State complexity of union and intersection for two-way nondeterministic finite automata. Fundam. Informaticae **110**(1–4), 231–239 (2011). https://doi.org/10.3233/FI-2011-540
9. Jirásková, G., Okhotin, A.: On the state complexity of operations on two-way finite automata. Inf. Comput. **253**, 36–63 (2017). https://doi.org/10.1016/j.ic.2016.12.007
10. Hibbard, T.N.: A generalization of context-free determinism. Inf. Control **11**(1/2), 196–238 (1967)
11. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. Int. J. Found. Comput. Sci. **25**(7), 897–916 (2014). https://doi.org/10.1142/S0129054114400140
12. Pighizzini, G., Pisoni, A.: Limited automata and context-free languages. Fundam. Inform. **136**(1–2), 157–176 (2015). https://doi.org/10.3233/FI-2015-1148
13. Guillon, B., Prigioniero, L.: Linear-time limited automata. Theor. Comput. Sci. **798**, 95–108 (2019). https://doi.org/10.1016/j.tcs.2019.03.037
14. Pighizzini, G.: Limited automata: properties, complexity and variants. In: DCFS 2019, Proceedings. Lecture Notes in Computer Science, vol. 11612, pp. 57–73. Springer, Cham(2019). https://doi.org/10.1007/978-3-030-23247-4_4
15. Pighizzini, G., Prigioniero, L.: Limited automata and unary languages. Inf. Comput. **266**, 60–74 (2019). https://doi.org/10.1016/j.ic.2019.01.002
16. Yamakami, T.: Behavioral strengths and weaknesses of various models of limited automata. In: SOFSEM 2019, Proceedings. Lecture Notes in Computer Science, vol. 11376, pp. 519–530. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-10801-4_40
17. Pighizzini, G., Prigioniero, L., Sádovský, S.: 1-Limited automata: Witness languages and techniques. J. Autom. Lang. Comb. **27**(1–3), 229–244 (2022). https://doi.org/10.25596/jalc-2022-229
18. Wagner, K.W., Wechsung, G.: Computational Complexity. D. Reidel Publishing Company, Dordrecht (1986)
19. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)
20. Sipser, M.: Lower bounds on the size of sweeping automata. J. Comput. Syst. Sci. **21**(2), 195–202 (1980). https://doi.org/10.1016/0022-0000(80)90034-3