Performing Regular Operations with 1-Limited Automata[⋆]

Giovanni Pighizzini[1], Luca Prigioniero[1][0000−0001−7163−4965], and Šimon Sádovský[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano
via Celoria, 18, 20133 Milan, Italy
{pighizzini,prigioniero}@di.unimi.it
[2] Department of Computer Science, Comenius University
Mlynská Dolina, 842 48 Bratislava, Slovakia
sadovsky@dcs.fmph.uniba.sk

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 on deterministic 1-limited automata: while for boolean operations the simulations remain polynomial, for product, star, and reversal they cost exponential in size. These bounds are tight.

1 Introduction

It is well known that regular languages are recognized by finite automata and are closed under several language 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 size of the description of recognizing devices. In this paper we focus on the complexity of union, intersection, complementation, product, star, and reversal. The costs of these operations on deterministic finite automata (1DFAs) have been widely studied in the literature [7, 6, 15, 16], while the case of two-way finite automata (in both deterministic and nondeterministic version) has also been considered [5, 4].

In this paper we study the descriptional complexity of language operations on deterministic 1-limited automata (D1-LAs). *Limited automata* are a kind of single-tape Turing machines with rewriting restrictions, introduced by Hibbard in 1967 [2] and recently reconsidered and deeply investigated (see, e.g., [1, 8–12, 14]). 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

⋆ The research of Šimon Sádovský was supported, in part, by Slovak Scientific Grant Agency VEGA (Grant 1/0601/20) and by Comenius University in Bratislava (Grant UK/258/2021).

constant $d \geq 0$ (we use the name *d-limited automaton* to explicitly mention the constant d). For any fixed $d \geq 2$, d-limited automata have the same power as pushdown automata, namely they accept exactly context-free languages [2], while deterministic 2-limited automata recognize exactly the class of deterministic context-free languages [10]. For $d = 0$ no rewritings are possible, hence the resulting models are two-way finite automata. The computational power does not increase if the rewritings in any cell are restricted *only to the first visit*. In other words, 1-limited automata are no more powerful than finite automata [13]. However, their descriptions can be significantly more succinct. In particular, a double exponential size gap between 1-limited automata and one-way deterministic finite automata has been proved in [9], 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.

In the study of the descriptional complexity of language operations given a family of recognizers (*source* devices), the goal is the investigation of the size of the devices (*target* devices) accepting the languages obtained by applying some operations to (the languages accepted by) the source devices. Up to now, in the literature it has been analyzed the size of *target* devices of the same family as the *source* devices. However, the results on the succinctness of the description of 1-limited automata suggested us to propose a different approach. Here, 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 capabilities of 1-limited automata of rewriting the cells of the tape during the first visit do 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 the equivalent finite automata. In fact, if we consider operations between 1DFAs (as source devices), we are able to create D1-LAs accepting the languages obtained by applying such operations that are smaller than the equivalent 1DFAs obtained by using standard constructions [16]. In particular, while the 1DFAs accepting the languages obtained by applying the operations of reversal, product, and star on the languages accepted by 1DFAs cost exponential, the constructions we provided yield equivalent D1-LAs whose sizes are only polynomial in the sizes of the source 1DFAs.

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 the fact that there is no smaller automaton than the one obtained by converting the simulated D1-LAs into 1DFAs first (obtaining exponentially larger machines), and then applying the corresponding (polynomial-size) language operation construction for obtaining a D1-LA.

2 Preliminaries

We assume the reader familiar with notions from formal languages and automata theory, in particular with *one-way* and *two-way* deterministic finite automata (1DFAs and 2DFAs for short, respectively). For further details see, e.g., [3]. 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^c denotes the *complement* of L, L^* denotes the (*Kleene*) *star* of L, L^R denotes the *reversal of* L, and $L \cdot L'$, $L \cup L'$, and $L \cap L'$ denote the *product* (or *concatenation*), *union*, and *intersection* of L and L', respectively (with the usual meaning).

A *deterministic* 1*-limited automaton* (D1-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 is a finite set of states, Σ is a finite *input alphabet*, Γ is a finite *working alphabet* such that $\Sigma \cup \{\triangleright, \triangleleft\} \subseteq \Gamma$, $\triangleright, \triangleleft \notin \Sigma$ are two special symbols, called the *left* and the *right end-markers*, and $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function. At the beginning of the computation, the input is stored onto the tape surrounded by the two end-markers, the left end-marker being at the position zero. Hence, on input w, the right end-marker is on the cell in position $|w| + 1$. The head of the automaton is on cell 1 and the state of the finite control is the *initial state* q_0. 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. Furthermore, the head cannot pass the end-markers, except at the end of computation, to accept the input, as explained below. However, replacing symbols is allowed to modify the content of each cell only during the first visit (after that, the contents of the cell is said to be *frozen*), with the exception of the cells containing the end-markers, which are never modified. For technical details see [10]. $\mathcal{A}$ accepts an input w if and only if there is a computation path which starts from the initial state q_0 with the input tape containing w surrounded by the two end-markers and the head on the first input cell, and which ends in a *final state* $q \in F$ after passing the right end-marker. It is an easy observation that one can enforce 1-limited automata to always rewrite each cell in the first visit so that they know whether they are scanning the cell for the first time or not.

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-limited automata 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 $\Theta(\#\Sigma \cdot \#Q \cdot \log(\#Q))$.

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. However, we show that, using D1-LAs as simulating machines, the costs reduce to polynomials. Then, we analyze the simulations of these operations when the given machines are D1-LAs. In this case, by studying suitable witness languages, we prove that the costs become exponential.

3.1 Simulations of Operations on 1DFAs

We now describe how to obtain a D1-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 $n' = \#Q'$, $n'' = \#Q''$, $Q' = \{q'_0, q'_1, \ldots, q'_{n'-1}\}$, and $Q' = \{q''_0, q''_1, \ldots, q''_{n''-1}\}$.

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, the 1DFA accepts. Since the simulating 1DFA keeps in its finite control, at the same time, a state of $\mathcal{A}'$ and the set of states reached by all the parallel simulations of $\mathcal{A}''$, its size is $\Theta(n' \cdot 2^{n''})$, which is optimal [16].

In our case the 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.

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*. We 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 $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$. The D1-LA stores in its finite control the position of the frontier in the right block of the window, the relative position of the head within the window, and the state of the automaton $\mathcal{A}'$, which is updated every time the cell at the frontier is read. At the beginning of the computation, the simulated state of the automaton $\mathcal{A}'$ 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.

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. Before visiting the cell in that position, 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. 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'. 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. 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.

After gathering this information, the 1-LA moves the head to the frontier, overwrites the cell, and the frontier is moved to the next cell. 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 $(0, \text{R})$.

When the machine detects the end of the input, indicated by the right endmarker $\lhd$, it has to check whether some simulation of $\mathcal{A}''$ halts in some accepting state. 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 rather than stopping in position $(n'' - 1, \text{L})$. The D1-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.

By computing the size of the resulting D1-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 D1-LA.

Theorem 1. *Let $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ and $\mathcal{A}'' = (Q'', \Sigma, \delta'', q_0'', F'')$ be two 1DFAs. Then there exists a D1-LA accepting $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ with $O(\#Q' \#Q''^4)$ states and $5\#\Sigma + 2$ working symbols.*

Let us now turn our attention to the star operation. Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ be a 1DFA. The D1-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 this case, the only automaton to be simulated is $\mathcal{A}$. The first simulation is started from the leftmost input cell. $\mathcal{N}$ then starts a new simulation every time a (simulated) final state is entered by some simulated computation of $\mathcal{A}$. 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 with ✔), and checking whether some of them reaches a state in F. After that, $\mathcal{N}$ checks whether the state whose index is equal to the index of the frontier (relative to the block) is reached at the end of the previous block by some simulation. Once this information is computed, the automaton moves the head on the cell at the frontier and overwrites it.

When the right endmarker is reached, $\mathcal{N}$ only needs to check whether some simulated device is in a final state and, in that case, accepts.

Theorem 2. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a 1DFA. Then there exists a D1-LA accepting $\mathcal{L}(\mathcal{A})^*$ with $O(\#Q^4)$ states and $5\#\Sigma + 2$ working symbols.*

3.2 Simulations of Operations on D1-LAs

We now focus on the size costs of the operations of product and star on D1-LAs. An immediate approach is to convert the source D1-LAs to 1DFAs, and then to apply the constructions shown in the previous section. Since converting D1-LAs into 1DFAs costs exponential in size [9], this procedure yields exponential-size D1-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.

For each 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: $L_k = \{w\{a,b\}^{kn}w \mid n \geq 0, w \in \{a,b\}^k\}$.

A D1-LA $\mathcal{A}_k$ may recognize L_k as follows. It first scans the leftmost block w of length k of the input, overwriting each symbol with a marked copy. Then, $\mathcal{A}_k$ repeats a subroutine which overwrites any subsequent block of length k, say x, with some fixed symbol $\sharp$, while checking in the meantime whether x equals w or not. This can be achieved as follows. A boolean variable matched is used to keep track of whether or not the prefixes of x and w compared so far match. At the beginning of the inspection of x, the device assigns true to matched, then it iteratively inspects the symbols of x. Suppose that all the symbols to the left of the j-th symbol of x have been inspected and overwritten by $\sharp$. Before inspecting the j-th symbol of x, first, $\mathcal{A}_k$, with the help of a counter modulo k, moves the head leftward to the position j of w and stores the unmarked scanned symbol σ in its finite control; second, it moves the head rightward until reaching the position j of x, namely, the leftmost position that has not been overwritten so far. At this point, $\mathcal{A}_k$ compares the scanned symbol (i.e., the j-th symbol of x) with σ. If the two symbols differ, the machine assigns false to matched. If, after inspecting a block of length k, $\mathcal{A}_k$ detects that the next symbol is the right endmarker, then it stops the computation, accepting in case matched contains true. Otherwise $\mathcal{A}_k$ repeats the subroutine described above in order to inspect the next block.

It is possible to implement $\mathcal{A}_k$ with a number of states linear in k and 7 working symbols (the input symbols and their marked copies, the endmarkers, and the symbol $\sharp$).

Let us now consider the language L_k^2, namely the product of L_k with itself. In this case, the ability of rewriting the tape cell contents of D1-LAs does not come in handy. This is because, ideally, the D1-LA cannot know in advance where to *"split"* the input string into two parts belonging to L_k. This idea is confirmed by the proof of the following result:

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

- *There exist two* D1-LA*s $\mathcal{A}'$ and $\mathcal{A}''$ of size linear in k such that any* D1-LA *accepting $\mathcal{L}(\mathcal{A}') \cdot \mathcal{L}(\mathcal{A}'')$ needs size at least exponential in k.*
- *There exists a* D1-LA *$\mathcal{A}$ of size linear in k such that any* D1-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 D1-LA of size linear in k.

Let us turn our attention to the language $L_k \cdot L_k = L_k^2$. To give a lower bound for the size required by any 1DFA accepting it, we are now 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 each 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. For each subset $S \subseteq \{1, 2, \ldots, N\}$, we define a string w_S as follows. Let $S = \{i_1, i_2, \ldots, i_n\}$, $1 \leq i_1 < i_2 < \ldots < i_n \leq N$. We define $w_S = x_{i_1} x_{i_1} x_{i_1} x_{i_2} x_{i_1} x_{i_3} x_{i_1} \cdots x_{i_n} x_{i_1}$ if $S \neq \emptyset$, otherwise $w_\emptyset = \varepsilon$. In other words, if S is nonempty, then 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. Now, consider two sets $S, T \subseteq \{1, 2, \ldots, N\}$, with $S \neq T$. Hence, 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 $w_S x \in L_k^2$ and $w_T x \notin L_k^2$. Let $x = x_{i_\ell}$. If $\ell > 1$, then $x_{i_1} x_{i_1} x_{i_1} x_{i_2} x_{i_1} \cdots x_{i_{\ell-1}} x_{i_1} \in L_k$ and $x_{i_\ell} x_{i_1} x_{i_{\ell+1}} x_{i_1} \cdots x_{i_n} x_{i_1} x_{i_\ell} \in L_k$. If $\ell = 1$, then $x_{i_1} x_{i_1} \in L_k$ and $x_{i_1} x_{i_2} x_{i_1} \cdots x_{i_n} x_{i_1} x_{i_1} \in L_k$. Hence, in both cases, $w_S x \in L_k^2$. On the other hand, the string $w_T x$ is not in L_k^2 because x does not occur in any other position of w_T. Actually, for the same reason, $w_T x \notin L_k^*$. This observation easily allows to extend our result to the 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. Moreover, since the conversion of D1-LAs into 1DFAs costs exponential [9], each D1-LA accepting $L_k \cdot L_k$ and each D1-LA accepting L_k^* has size at least $2^{O(k)}$. □

In conclusion, starting from two D1-LAs $\mathcal{A}'$ and $\mathcal{A}''$ accepting the languages L' and L'' (resp., from a D1-LA $\mathcal{A}$ accepting a language L), a D1-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

4.1 Simulations of Operations on 1DFAs

It is well known that for union, intersection, and complement, 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. 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 [16].

If we use a 2DFA as target machine, it can perform the simulation of the first 1DFA during a sweep from left to right, then, when the end of the input is reached, the head is brought at the beginning of the tape and the simulation of the second 1DFA is started. In the case of the union, the 2DFA accepts if the simulation of at least one 1DFA accepts, while, in the case of the intersection, the input is accepted if both the simulated 1DFAs accept. The 2DFAs implementing these simulations only need to store, in their 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.

From the resulting 2DFAs we can directly obtain equivalent D1-LAs that, during the first sweep, simply overwrite each tape cell with a copy of the symbol it originally contains.

Theorem 4. *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 D1-LA for the language $\mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}'')$ and*
- *a D1-LA for the language $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}'')$*

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

The D1-LA for the complement can be obtained with a construction analogous to the standard one used for obtaining a 1DFA for complementation, i.e., just by complementing the set of the accepting states.

Theorem 5. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a 1DFA. Then there exists one D1-LA with $\#Q$ states and $\#\Sigma + 3$ working symbols which accepts $\mathcal{L}(\mathcal{A})^c$.*

4.2 Simulations of Operations on D1-LAs

Let us now suppose that source and target machines are D1-LAs. We give constructions based on a result on linear-time simulations of 1-LAs in polynomial size: In [1] it is showed that, given a 1-LA, paying a polynomial growth in size it is possible to obtain an equivalent one that works in linear time. The idea of the construction is similar to the technique used for the simulation of the product of Section 3: the simulating device works on a virtual window of fixed size that is shifted along the tape in a one-way manner. 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 (for further details we address the reader to [1, Theorem 1 and Lemma 6]).

Lemma 1. *For each D1-LA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ there exists an equivalent D1-LA $\mathcal{A}'$ working in linear time with $O(\#Q^4)$ states and $(\#Q + 1) \cdot \#(\Gamma \setminus \Sigma)$ working symbols.*

For the simulation of union and intersection of the languages accepted by two D1-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 D1-LAs (in this phase, only the cells of the two windows are visited: it is used the window of the first simulated device and then, when the information has been gathered, the window of the second simulated device is used). Then the new cell is entered and the information is written (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, and in the case of the intersection it accepts if both the simulated devices accept.

Theorem 6. *Let $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q'_0, F')$ and $\mathcal{A}'' = (Q'', \Sigma, \Gamma', \delta'', q''_0, F'')$ be two* D1-LA*s, $n' = \#Q'$, and $n'' = \#Q''$. Then there exist*
- *a* D1-LA *for the language $\mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}'')$ and*
- *a* D1-LA *for the language $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}'')$*

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

To accept the complement of the language accepted by a D1-LA $\mathcal{A}$, again Lemma 1 can be used to perform a linear-time (and therefore, halting) simulation of $\mathcal{A}$. The simulating D1-LA accepts if $\mathcal{A}$ enters a loop or if it is not in an accepting state at the end of its computation.

Theorem 7. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a* D1-LA*. Then there exists a* D1-LA *with $O(\#Q^4)$ states and $(\#Q+1)\#(\Gamma' \setminus \Sigma)$ working symbols which accepts $\mathcal{L}(\mathcal{A})^c$.*

5 Reversal

The last operation we study is the reversal. Even in this case, the D1-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, it accepts if, simulating the transitions of the 1DFA scanning the input from right to left, enters a final state when the head reaches the left endmarker. 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.

As a consequence, we are able to construct an equivalent D1-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 8. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a* 1DFA*. Then there exists one* D1-LA *with $\#Q + 2$ states and $2\#\Sigma + 2$ working symbols which accepts $\mathcal{L}(\mathcal{A})^R$.*

In the case of D1-LAs, the reversal has an exponential cost in size. The exponential upper bound can be obtained by converting the D1-LA into a 1DFA and then applying Theorem 8. A matching exponential lower bound has been proved in [1].

Theorem 9 ([1, Theorem 4]). *For any integer $k \geq 2$, there exists a* D1-LA $\mathcal{A}$ *of size linear in k such that any* D1-LA *accepting $\mathcal{L}(\mathcal{A})^R$ needs size exponential in k.*

References

1. Guillon, B., Prigioniero, L.: Linear-time limited automata. Theor. Comput. Sci. **798**, 95–108 (2019)
2. Hibbard, T.N.: A generalization of context-free determinism. Information and Control **11**(1/2), 196–238 (1967)
3. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)
4. 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
5. 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
6. 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
7. Maslov, A.N.: Estimates of the number of states of finite automata. In: Doklady Akademii Nauk. vol. 194, pp. 1266–1268. Russian Academy of Sciences (1970)
8. Pighizzini, G.: Limited automata: Properties, complexity and variants. In: DCFS 2019, Proceedings. Lecture Notes in Computer Science, vol. 11612, pp. 57–73. Springer (2019). https://doi.org/10.1007/978-3-030-23247-4_4
9. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. Int. J. Found. Comput. Sci. **25**(7), 897–916 (2014)
10. Pighizzini, G., Pisoni, A.: Limited automata and context-free languages. Fundam. Inform. **136**(1-2), 157–176 (2015)
11. Pighizzini, G., Prigioniero, L.: Limited automata and unary languages. Information and Computation **266**, 60–74 (2019)
12. Pighizzini, G., Prigioniero, L., Sádovský, Š.: 1-Limited Automata: Witness Languages and Techniques. J. Autom. Lang. Comb. (2022), to appear.
13. Wagner, K.W., Wechsung, G.: Computational Complexity. D. Reidel Publishing Company, Dordrecht (1986)
14. 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 (2019)
15. 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
16. 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