Revised submission to the Journal of Automata, Languages and Combinatorics
Revised on January 20, 2022

1-LIMITED AUTOMATA: WITNESS LANGUAGES AND TECHNIQUES

Giovanni Pighizzini[(A)] Luca Prigioniero[(A)] Šimon Sádovský[(B)]

[(A)] *Dipartimento di Informatica, Università degli Studi di Milano*
via Celoria, 18, 20133 Milan, Italy
pighizzini@di.unimi.it prigioniero@di.unimi.it

[(B)] *Department of Computer Science, Comenius University*
Mlynská Dolina, 842 48 Bratislava, Slovakia
sadovsky@dcs.fmph.uniba.sk

ABSTRACT

1-limited automata are single-tape Turing machines with strong rewriting restrictions, that do not allow them to recognize more than regular languages. However, 1-limited automata can be significantly more succinct than equivalent finite automata: in fact, the size gap from 1-limited automata to one-way deterministic finite automata is double exponential. In this paper we present and discuss some languages which can be used as witnesses of the gaps between different kinds of 1-limited automata and finite automata. Among them, refining previous techniques, we show that a language proposed long time ago by Meyer and Fischer as a witness of the optimality of the subset construction for finite state automata, can also be used as witness of all the currently known size gaps between 1-limited automata and different variants of finite automata. We also discuss some open problems and possible further lines of investigation.

1. Introduction and Preliminaries

The investigation of computational models working with restricted resources is a classical topic of theoretical computer science, connecting fundamental areas such as computability and complexity. In this field of research, Thomas Hibbard introduced long time ago *limited automata*, a kind of single-tape Turing machine with rewriting restrictions [2]. This model has been recently reconsidered and deeply investigated in a series of papers (e.g., [8, 9, 4, 3, 1, 7, 10, 16]). However, many problems related to it are still open and deserve further investigations.

Limited automata are nondeterministic single-tape Turing machines which can overwrite 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 the whole 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 [2]. At the bottom level of the hierarchy, deterministic 2-limited automata recognize exactly the class of deterministic context-free languages [9].

For $d = 0$ no rewritings are possible, hence the resulting models are two-way finite automata, which recognize exactly regular languages in both deterministic and nondeterministic cases [11, 13]. 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 [15]. 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 [8], 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. These and other bounds are summarized in Figure 1.

In the investigation of these size relationships there are still some challenging open problems (for a recent survey see [7]). Among them, we point out the cost of the elimination of nondeterminism from 1-limited automata, for which we know a double exponential upper bound and an exponential lower bound. These problems are also related to the main longstanding open question of Sakoda and Sipser concerning the state costs of the transformation of one-way and two-way nondeterministic finite automata into equivalent two-way deterministic finite automata [12], as pointed out in [7].

In order to solve this kind of problems, and in particular in order to try to increase lower bounds and, in general, to better understand how these models work, it is useful to investigate languages which are witnesses for some of the known gaps or seem to be good candidates for better lower bounds. In this paper we will focus on some of these languages. We will recall some known examples, we present some of their variants, and we will discuss new examples. We think that reasoning on these witness languages will be helpful to better understand these models and to stimulate further research and new results and, hopefully, to solve some of the open problems.

The most important gaps in the literature between 1-limited automata and one-way deterministic and nondeterministic finite automata have been witnessed by using languages with a block structure. We will discuss such kind of languages in Section 3. Among the languages of this type, we will present (in Section 6) the "subset parity" language and one variant of it, as possible candidates for increasing the lower bound for the elimination of nondeterminism from 1-limited automata. In Section 4 we recall the results from [10] about the recognition of unary languages by 1-limited automata. For these languages the techniques are different, in particular we cannot consider block structures. This discussion will turn to be useful in Section 5 where we consider the language proposed by Meyer and Fischer to prove the optimality of the subset construction used to eliminate nondeterminism from finite automata. Adapting the techniques for unary languages recalled in the previous section, we show that this

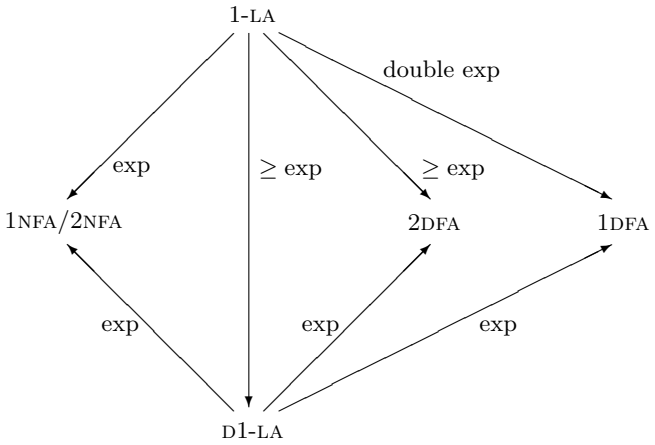

Figure 1: State costs of conversions of 1-limited automata (1-LA) and deterministic 1-limited automata (D1-LA) into equivalent one-way and two-way deterministic and nondeterministic finite automata (1/2 NFA/DFA). The upper bounds in the diagram follow from the simulations of 1-limited automata by finite automata (Theorem 2). For the lower bounds the following witnesses have been used in the literature: the block language L_n presented in Section 3, with the exceptions of lower bounds from D1-LAs to 2DFAs, 1NFAs, and 2NFAs [8]; the unary language U_n presented in Section 4, with the exceptions of lower bounds from 1-LAs to D1-LAs and to 1DFAs [10]. For a further discussion on the bounds on this figure see [7].

language is a witness of all lower bounds for the conversions in Figure 1. We believe that this is interesting, because this language does not have a block structure as the languages in Section 3 which have been used, up to now, to prove the optimality of the conversions from (deterministic and nondeterministic) 1-limited automata to other kinds of finite automata.

In Section 6 we discuss some possible lines for future investigations.

The preliminary notions concerning limited automata are given in Section 2, with a special emphasis on 1-limited automata. In general, we assume the reader familiar with the standard notions concerning formal languages, finite automata, determinism and nondeterminism. Given a machine M, the language accepted by it will be denoted as $\mathcal{L}(M)$. Given a set S let us denote by $\#S$ its cardinality and by 2^S the set of all its subsets. The empty string will be denoted by ε.

2. Limited Automata

Given an integer $d \geq 0$, a *d-limited automaton* (d-LA, for short) is a tuple $A = (Q, \Sigma, \Gamma, \delta, q_I, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite *working alphabet* such that $\Sigma \cup \{\rhd, \lhd\} \subseteq \Gamma$, $\rhd, \lhd \notin \Sigma$ are two special symbols, called the *left* and the *right end-markers*, respectively, $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{-1,+1\}}$ is the transition function. At the beginning of the computation, the input w is stored

on the tape surrounded by the two end-markers, the left end-marker being at the position zero. Hence, the right end-marker is on the cell in position $|w| + 1$. The head of the automaton is on the cell 1 and the state of the finite control is the *initial state* q_I. In one move, according to δ and to the current state, 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, $(q, X, m) \in \delta(p, a)$ means that when the automaton in the state p is scanning a cell containing the symbol a, it can enter the state q, rewrite the cell contents by X, and move the head to the *left*, if $m = -1$, or to the *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 d visits, with the exception of the cells containing the end-markers, which are never modified.
- The end-marker symbols cannot be used to replace the contents of any of the cells which initially contain the input. (With the previous condition, this implies that if $(q, X, m) \in \delta(p, a)$ and either $X \in \{\rhd, \lhd\}$ or $a \in \{\rhd, \lhd\}$, then $X = a$.)
- The head cannot violate the end-markers, i.e, it cannot move to the left of the left end-marker or to the right of the right end-marker, except at the end of computation, to accept the input, as explained below.

Further technical details (which are not necessary in this paper) can be found in [9].

An automaton A is said to be *limited* if it is d-limited for some $d \geq 0$. A accepts an input w if and only if there is a computation path which starts from the initial state q_I 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 violating the right end-marker, i.e., with the head which leaves the tape by moving to the right of the right end-marker. A is said to be *deterministic* (Dd-LA, for short) whenever $\#\delta(q, a) \leq 1$, for any $q \in Q$ and $a \in \Gamma$.

When $d = 0$, namely no rewritings are allowed, we obtain two-way finite automata, shortly denoted as 2NFA or 2DFA depending on whether nondeterministic transitions are allowed or not. So even for two-way automata we assume the same accepting condition as for limited automata. By 1NFAs and 1DFAs we denote one-way nondeterministic and deterministic finite automata, respectively.

In this paper we focus on 1-limited automata in both deterministic and nondeterministic versions. We briefly recall some important results about 1-LAs. First, concerning the computational power of these devices, the following has been proved:

Theorem 1 [15, Thm. 12.1]. *The class of languages accepted by* 1*-limited automata coincides with the class of regular languages.*

Hence, it is quite natural to compare the sizes of 1-LAs with the sizes of equivalent finite automata, i.e., to study these devices from the descriptional complexity point of view. We note that, fixing an input alphabet Σ, the number of symbols which are used to write down the description of a 1-LA is a polynomial in the cardinalities of the state set and of the working alphabet. So, to define the size of 1-LAs we will take

into account those two parameters. Concerning finite state automata we consider the number of states to be a measure of their size, as usual.

Theorem 2 [8]. *Each n-state 1-limited automaton can be transformed into equivalent one-way nondeterministic and deterministic finite automata with $n \cdot 2^{n^2}$ and $2^{n \cdot 2^{n^2}}$ states, respectively. Hence, the size costs of the transformation of 1-limited automata into equivalent one-way nondeterministic and deterministic finite automata are exponential and double exponential, respectively.*

We point out that the costs of the transformations in Theorem 2 do not depend on the size of the working alphabet of the given 1-LA. Furthermore, the double exponential cost of the transformation into 1DFAs is related to a double role of the nondeterminism in 1-LAs: when the head of a 1-LA reaches for the first time a tape cell, it overwrites the contents according to a nondeterministic choice; furthermore, the set of nondeterministic choices that are allowed during the next visits to the same cell depends on the symbol that has been chosen to rewrite it in the first visit and that cannot be further changed, namely it depends on the nondeterministic choice which was made during the first visit.

When the given 1-limited automaton is *deterministic*, the same simulation produces a *one-way deterministic* finite automaton of exponential size. In particular, each n-state D1-LA can be transformed into an equivalent 1DFA with no more than $n \cdot (n+1)^n$ states [8].

The results comparing the sizes of finite automata and 1-limited automata are summarized in Figure 1 (see also [7]). The optimality of the bounds in Figure 1 has been shown considering different witness languages. Some bounds are witnessed by languages with a block structure, while other bounds has been proven using unary languages. In the next two sections we will discuss these two kinds of languages.

3. Block Languages

A *block language* B_n is defined, with respect to an integer parameter $n > 0$, as the concatenation of some strings of length n, called *blocks*, over a fixed alphabet Σ, which satisfy some given constraints.

Some examples are:

- The set of strings in which the last block is equal to one of the previous ones:
$$K_n = \{x_1 \cdots x_k x \mid k > 0,\ x_1, \ldots, x_k, x \in \{a,b\}^n, \exists j \in \{1, \ldots, k\},\ x_j = x\}\,.$$
- The set of strings in which two blocks are equal:
$$E_n = \{x_1 \cdots x_k \mid k > 0,\ x_1, \ldots, x_k \in \{a,b\}^n, \exists i, j \in \{1, \ldots, k\}, i < j,\ x_i = x_j\}\,.$$
- The set L_n of strings in which at least n blocks are equal:
$$\begin{aligned} L_n = \{x_1 \cdots x_k \mid k \geq 0,\ & x_1, \ldots, x_k \in \{a,b\}^n, \\ & \exists i_1, i_2, \ldots, i_n \in \{1, \ldots, k\}, i_1 < i_2 < \ldots < i_n \\ & \text{s.t. } x_{i_1} = x_{i_2} = \cdots = x_{i_n}\}\,. \end{aligned}$$

For any of these languages, we can give a 1-la M_n of size $O(n)$ which works as follows:

- In a preliminary scan of the input, from left to right, using a counter modulo n, M_n verifies whether the input length is a multiple of n. If this is not the case, then it stops and rejects. Notice that, since the contents of the cells are frozen after the first visit, only during this scan the cells can be overwritten. Indeed, during the first scan, M_n marks the first cell of some blocks which are guessed to be "interesting", e.g., in the case of the language E_n, it marks the first cell of two blocks which are guessed to be equal (according to the definition of E_n, in the following discussion these two blocks will be called x_i and x_j). In the case of L_n, the number of cells which are marked in this first scan is arbitrary (anyway, according to the above-described strategy, each marked cell is the first cell of a block); then a further scan is used to verify that such a number is n. In this way, the number of states used to mark the "interesting" positions is $O(n)$.
- In the remaining part of the computation the machine verifies whether the previous guess was correct by comparing symbols in the corresponding positions of the selected blocks. To locate these positions, a counter modulo n, which is kept in the finite control, is used. This part can be implemented with $O(n)$ states. We describe a possible strategy for the language E_n, that can be adapted to the other languages under consideration.
  - M_n moves its head from the right end-marker, on which it is located after the first scan, up to reach a marked cell, namely the cell containing the first symbol of x_j. The machine stores in its finite control the symbol and continues to move to the left up to reach the other marked cell, namely the cell containing the first symbol of x_i.
  - If the symbol in this cell is different from that stored in the control then M_n rejects; otherwise, it moves its head one cell to the right, i.e., on the second symbol of x_i, stores it in the finite control, and moves to the right, to locate the second symbol of x_j. This can be done by counting the cells modulo n, while moving to the right and stopping when, after having visited a marked cell, the counter contains 0. Even in this case, if the symbol in the cell is different from that stored in the control then M_n rejects, otherwise it moves one cell to the right, stores the symbol in this cell in the finite control, and moves to the left to search the corresponding cell in x_i. This is also done by using a counter modulo n, with a small modification of the above described strategy. Continuing in this way, M_n can compare symbols in corresponding positions of x_i and x_j. If one of the two marked cells is reached with the counter containing 0, then all the corresponding symbols in the two blocks have been successfully compared and M_n can stop and accept.
  - It is not difficult to verify that with a strategy of this kind, $O(n)$ states are enough. Furthermore the working alphabet Γ consists of two copies of Σ: one copy for the symbols which are marked in the first scan, the other for the symbols which are not marked.

It is also possible to prove that every 1DFA accepting one of these languages should require at least 2^{2^n} states. This can be done by standard distinguishability arguments.

We give a sketch of the proof in the case of E_n. Let $w_1, w_2, \ldots, w_{2^n}$ be a list of all strings in $\{a,b\}^n$, in some fixed order, and let F be the set of all functions from $\{1,2,\ldots,2^n\}$ to $\{0,1\}$. For each $f \in F$, consider the string $w_f = w_1^{f(1)} w_2^{f(2)} \cdots w_{2^n}^{f(2^n)}$, where $w_i^0 = \varepsilon$ and $w_i^1 = w_i$, $i = 1, \ldots, 2^n$. Given $f, g \in F$, with $f \neq g$, it can be verified that any string w_i with $f(i) \neq g(i)$ distinguishes w_f and w_g. In fact, each string of length n different from w_i occurs at most one time as a block in $w_f w_i$ and in $w_g w_i$, while w_i occurs two times in one of them. Since $\#F = 2^{2^n}$, the lower bound follows. A more detailed argument for the language L_n can be found in [8].

From the previous discussion, it follows that all the block languages presented in this section are witnesses of the double exponential gap from 1-LAs to 1DFAs. Furthermore, since the gap from D1-LAs to 1DFAs is "only" exponential, each D1-LA accepting one of these languages should have size at least exponential in n.

We now describe how to get a D1-LA of size exponential in n for the language E_n. Actually, this can be done using a 2DFA A_n, i.e., without any overwriting. For each string $x \in \{a,b\}^n$, A_n makes a scan of the tape in order to check if there are two blocks of the input which are equal to x. In this case it stops and accepts. A_n can be implemented with $O(n \cdot 2^n)$ states, by enumerating all the strings in $\{a,b\}^n$ in some given order. (The factor 2^n is to keep track of the strings of length n in the enumeration, the factor n is to count the position in the input modulo n.)

A similar strategy can be also used for languages K_n and L_n.

4. Unary Languages

The acceptance of unary languages by 1-LAs was extensively studied in [10]. In this section we give a short outline of some of the results presented in that paper. This will turn out to be useful for the results we will prove in the next section.

We first notice that in the unary case block languages are quite trivial. Hence, the techniques discussed in the previous section are not interesting in this case.

Let us start by recalling some useful notions. Then, we will give an example illustrating them.

Definition 3 [14]. The *binary carry sequence* is the infinite integer sequence $\sigma_1\sigma_2\cdots\sigma_j\cdots$ where σ_j is the exponent of the highest power of 2 which divides j, for each integer $j \geq 1$.

Definition 4. Given a finite sequence $s = k_1 k_2 \cdots k_j$ of integers, its *backward increasing sequence*, denoted as BIS(s), is the longest sequence which is obtained by inspecting s from right to left and by selecting the element of s under consideration when it is strictly greater than the elements selected until then, i.e., $\text{BIS}(k_1 k_2 \cdots k_j) = (i_1, i_2, \ldots, i_r)$, $j, r > 0$, if and only if $i_1 = k_{h_1}, i_2 = k_{h_2}, \ldots, i_r = k_{h_r}$ where $h_1 = j$, $h_t = \max\{h' < h_{t-1} \mid k_{h'} > k_{h_{t-1}}\}$ for $t = 2, \ldots, r$, and $k_{h'} < k_{h_r}$ for $0 < h' < h_r$.

For example, the prefix of length $j = 21$ of the binary carry sequence is 010201030102010401020 and BIS(010201030102010401020) $= (0, 2, 4)$. Notice that $21 = 2^4 + 2^2 + 2^0$, i.e., the elements of BIS(010201030102010401020) coincide with the positions of bits 1 in the binary representation of 21. Furthermore, the $22nd$ element of the binary carry sequence is $\sigma_{22} = 1$ and coincides with the smallest nonnegative integer which does not appear in BIS(010201030102010401020). In a similar way, by taking the prefix of length $j = 11$ of the binary carry sequence, we get BIS(01020103010) $= (0, 1, 3)$, while $11 = 2^3 + 2^1 + 2^0$. Even in this case, the next element of the binary carry sequence is the smallest nonnegative integer which does not appear in BIS(01020103010), namely, $\sigma_{12} = 2$.

The above examples illustrate some general properties of the binary carry sequence, which are fundamental in the definition of the D1-LAs we are going to present. The proof can be found in [10]:

Lemma 5. *Let $\sigma_1\sigma_2\cdots\sigma_j$, $j > 0$, denote the prefix of length j of the binary carry sequence.*

- *If BIS$(\sigma_1\sigma_2\cdots\sigma_j) = (i_1, i_2, \ldots, i_r)$ then $j = \sum_{t=1}^{r} 2^{i_t}$, i.e., the value of BIS, applied to the first j elements of the binary carry sequence, gives the positions of bits equal to 1 in the binary representation of j, starting from the least significant bit.*
- *σ_j is the smallest nonnegative integer that does not occur in BIS$(\sigma_1\sigma_2\cdots\sigma_{j-1})$.*

Now we shortly discuss the construction of a unary D1-LA A_n accepting the singleton language $\{a^{2^n}\}$, for a given integer $n > 0$ [10]. The idea is that of replacing input symbols on the tape with the elements of the prefix of length 2^n of the binary carry sequence. To this end, we take the working alphabet of A_n to contain the symbols $0, 1, \ldots, n$.

At the beginning, the first input symbol is replaced by 0, namely by the first element of the binary carry sequence. Suppose that at some point the first $j-1$ input symbols have been replaced by the prefix $\sigma_1\sigma_2\cdots\sigma_{j-1}$ of the binary carry sequence and the head is on the cell $j-1$, while the cell j is not yet visited. According to Lemma 5, the element σ_j that should be written on the cell j is the smallest nonnegative integer that does not occur in BIS$(\sigma_1\sigma_2\cdots\sigma_{j-1})$. This element can be easily computed by visiting some of the cells to the left of the cell j.

For instance, for $j = 12$, the cells up to 11 contain the sequence 01020103010. By moving the head from cell 11 to the left, up to cell 8 which contains 3, A_n can discover that 2 is the smallest integer missing in the backward increasing sequence. So, A_n can move the head to the right, up to the leftmost "fresh" cell, i.e., cell 12, and overwrite it with the next element of the binary carry sequence, i.e., the number 2.

If the length of the input is exactly 2^n, then at some moment the integer n will be written on the rightmost input cell. At this point, the machine can accept after verifying that the next cell contains the right end-marker. Otherwise, the machine will reject either because after writing the integer n on a cell, the tape still contains cells which are not yet visited (the input is too long), or because the head hits the right end-marker before the integer n is written on the tape (the input is too short).

We point out that the D1-LA A_n can be implemented using $O(n)$ states and $O(n)$ tape symbols (further details are presented in [10]). With a minor modification, we can also obtain a D1-LA of similar size accepting the language $U_n = \{a^{2^n}\}^*$. The same language needs 2^n states to be accepted by finite automata, even if nondeterminism and two-way motions are allowed, i.e., each 2NFA accepting U_n must have at least 2^n states ([5], see also Theorem 9). Hence the language U_n is a witness of all exponential gaps from 1-LAs to finite automata in Figure 1.

We conclude this section by discussing how the above described D1-LA can be modified in order to recognize all the unary strings whose length is a multiple of a given integer $N > 0$, which is not required to be a power of 2, i.e., the language $M_N = \{a^N\}^*$. This will be used in the next section for further constructions. To this aim, adapting the previous argument, we build a D1-LA B_N in such a way that, after reading the leftmost K tape cells, their contents will be overwritten as

$$\underbrace{\sigma_1 \cdots \sigma_{N-1} \sharp \cdots \sharp \sigma_1 \cdots \sigma_{N-1} \sharp}_{\lfloor K/N \rfloor \text{ times}} \sigma_1 \cdots \sigma_{K \bmod N}$$

where $\sigma_1 \cdots \sigma_{N-1}$ is the prefix of length $N-1$ of the binary carry sequence, and $\sharp$ is a special "reset" symbol which is written in each cell whose position is a multiple of N. To do that, B_N computes the binary carry sequence in a way similar to that of A_n, with two main differences:

(I) In B_N, the computation of the next symbol of the binary carry sequence is restricted to the suffix of the tape which starts after the rightmost reset symbol $\sharp$ (if no reset symbol has been written all the tape is used; the left end-marker and the symbol $\sharp$ play a similar role here). In other words, all the symbols to the left of the rightmost $\sharp$ are not considered anymore. In this way, after having written a symbol $\sharp$, the computation of the binary carry sequence on the next tape cells is restarted from the beginning.

(II) To compute the next element of the binary carry sequence, the automaton A_n has to find the smallest integer missing in the backward increasing sequence. The automaton B_N will also verify if the backward increasing sequence represents the integer $N-1$. In this case, the reset symbol $\sharp$ will be written on the first fresh cell of the tape.

For instance, suppose that $N = 14$, then $N - 1 = 13 = 2^3 + 2^2 + 2^0$: each time B_N discovers that the backward increasing sequence starts by 0 followed by 2, and hence the next element of the binary carry sequence is 1, it also verifies if the complete backward increasing sequence is $(0, 2, 3)$. In this case, it will write $\sharp$ instead of 1 on the next tape cell.

The implementation of these changes, which is quite technical, keeps the number of states and the size of the working alphabet linear in the maximum element of the binary carry sequence we need to consider, namely the sizes of both these sets are $O(\log N)$.

5. Beyond the Block Structure

In Section 3 we discussed some block languages witnessing the double exponential size gap from 1-LAs to 1DFAs and the exponential gap from D1-LAs to 1DFAs as well. In this section we present another witness language for these gaps. In our knowledge, this is the first example of witness which does not have a block structure similar to the languages of Section 3.

The language we use was proposed longtime ago by Meyer and Fischer as a witness of the exponential state gap between 1NFAs and 1DFAs [6] and it is the language accepted by the N-state 1NFA S_N depicted in Figure 2.

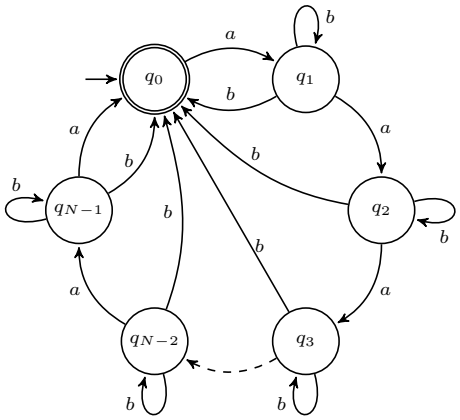

Figure 2: The N-state 1NFA S_N accepting the language of Meyer and Fischer.

Theorem 6. *Given an integer N let S_N be the N-state 1NFA in Figure 2. Then:*

(I) *The minimum 1DFA equivalent to S_N has 2^N states.*

(II) *There exists a 1DFA with $2N$ states accepting the reversal of the language accepted by S_N.*

(III) *There exists a 2DFA with $2N + 2$ states equivalent to S_N.*

Proof. For Item I see [6].

To prove Item II, we observe that a 1NFA S'_N accepting $\mathcal{L}(S_N)^R$ can be obtained just reversing the arrows in the transition diagram in Figure 2. We claim that the only reachable states in the subset automaton obtained from S'_N are the states corresponding to subsets $\{q_i\}$ and $Q \setminus \{q_i\}$, for $i = 0, \ldots, N-1$, where $Q = \{q_0, q_1, \ldots, q_{N-1}\}$ is the set of states of S_N. First, we observe that from the initial state q_0, by reading any sequence of a's, only one state q_i can be reached, $0 \leq i \leq N-1$; furthermore, in this way all the states corresponding to singletons are reachable. Reading b in one of the states q_i does not change the current state, when $i > 0$, but leads to one of the states in the set $\{q_1, \ldots, q_{N-1}\} = Q \setminus \{q_0\}$, when $i = 0$. Notice that this is the only nondeterministic choice that can be taken by the automaton. Hence, the

state corresponding to the subset $Q \setminus \{q_0\}$ is also reachable. Furthermore, from this subset, reading sequences of a's, all the subsets $Q \setminus \{q_i\}$, for $i = 0, ..., N-1$, can be reached. Again, from the subset $Q \setminus \{q_i\}$ reading a letter b the set of reachable states becomes $Q \setminus \{q_0\}$ when $i > 0$, and remains $Q \setminus \{q_0\}$ when $i = 0$. This allows us to conclude that in the subset automaton obtained from S'_N the only reachable states are $\{q_i\}$ and $Q \setminus \{q_i\}$, for $i = 0, \ldots, N-1$ (see Figure 3).

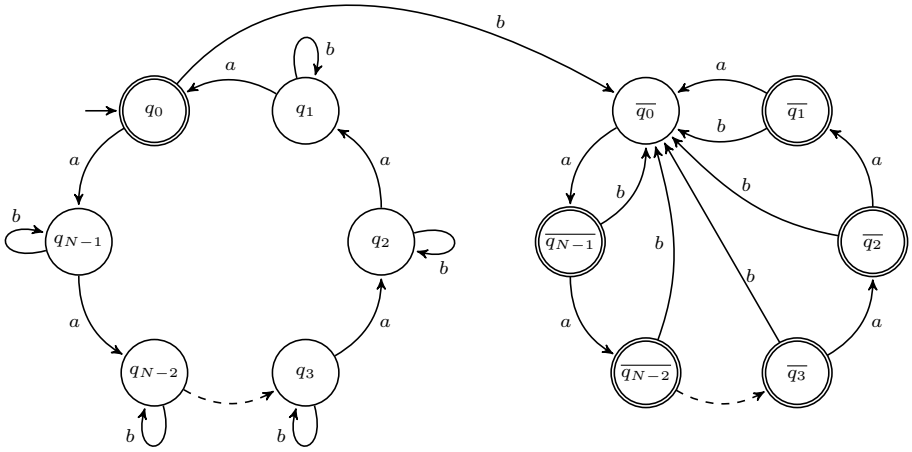

Figure 3: The subset automaton obtained from the reversal of the 1NFA S_N accepting the language of Meyer and Fischer, restricted to reachable states. Here, the state q_i (resp., $\overline{q_i}$) corresponds to the singleton $\{q_i\}$ (the subset $Q \setminus \{q_i\}$), for $i = 0, \ldots, N-1$.

For Item III we can construct a 2DFA that, in the initial state, scans the input from left to right to locate the right end-marker and then simulates the $2N$-state 1DFA accepting $\mathcal{L}(S_N)^R$ obtained at the previous point, while reading the input from right to left. Finally, when it reaches the left end-marker, it moves in the final state in which it performs a final left-to-right scan, violates the right end-marker, and accepts. □

Now we are going to study how to recognize $\mathcal{L}(S_N)$ using 1-LAs and D1-LAs.

Theorem 7. *For each $N > 1$, the language $\mathcal{L}(S_N)$ is accepted by a* 1-LA *with $O(\log N)$ states and $O(\log N)$ working alphabet symbols.*

Proof. We define a 1-LA C_N which, roughly, simulates S_N computations by overwriting a tape cell by the symbol $\sharp$ when the transition on that cell in the simulated computation leads to the initial state q_0 of S_N.

To this aim, while reading a's, C_N behaves as the D1-LA B_N accepting $\{a^N\}^*$ described in Section 4, namely it transforms a sequence a^K into the string

$$\underbrace{\sigma_1 \cdots \sigma_{N-1} \sharp \cdots \sharp \sigma_1 \cdots \sigma_{N-1} \sharp}_{\lfloor K/N \rfloor \text{ times}} \sigma_1 \cdots \sigma_{K \bmod N}.$$

We now discuss the changes we have to make in order to deal with the symbol b. To this aim, it is useful to observe the role of the letter b in the 1NFA S_N:

- In the state q_0 the letter b cannot occur.
- In the other states, either the letter b remains neutral, i.e., it does not produce a state change, or it leads to a "reset" of the automaton.

According to these two points, the automaton C_N behaves as follows, when reading a b from the input:

- If the symbol on the tape in the cell to the left of the one containing b is the left end-marker or the reset symbol $\sharp$, then C_N is simulating S_N being in the state q_0 in which the letter b cannot occur. Hence the move is not defined.
- Otherwise C_N either overwrites the cell containing b by the "neutral" symbol _ (that will be skipped in the next computation steps), thus simulating the transition of S_N that does not change the state, or overwrites it by the reset symbol $\sharp$, so simulating the transition on b which leads to q_0.

When, after writing the symbol $\sharp$ on a cell and moving to the right, the head hits the right end-marker, C_N accepts. It also accepts in the case the head is on the right end-marker at the beginning of the computation, i.e., the input is the empty word.

According to this construction, the string finally written on the tape in any accepting computation on input $x \neq \varepsilon$ has the form

$$w_1\sharp w_2\sharp \cdots w_k\sharp$$

where $k \geq 0$, and for $i = 1, \ldots, k$, $w_i \in \{\_, 0, 1, \ldots, \lfloor \log_2 N \rfloor\}^+$ is a prefix of the binary carry sequence of length less than N padded with some occurrences of the symbol _ (which cannot occur at the beginning of w_i). Furthermore, the original symbol in each position which finally contains the symbol _ was b, while the original symbol in each position which finally contains the symbol $\sharp$ was a if the factor w_i to its left, after removing padding symbols, represents the prefix of length $N-1$ of the binary carry sequence, otherwise it was b. Hence, the positions of symbols $\sharp$ in this string represent the positions in which the simulated computation of S_N entered the state q_0.

We finally observe that C_N uses the same set of states of B_N, whose cardinality is $O(\log N)$. Furthermore, the cardinality of the working alphabet is $O(\log N)$. □

For the recognition of $\mathcal{L}(S_N)$ by *deterministic* 1-LAs we obtain:

Theorem 8. *The language $\mathcal{L}(S_N)$ is accepted by a D1-LA with $O(N)$ states and a working alphabet of size $O(1)$.*

Proof. This is an immediate consequence of Item III of Theorem 6. □

Since by Item I of Theorem 6 the language $\mathcal{L}(S_N)$ requires 2^N states to be accepted by 1DFAs, from Theorems 7 and 8 it turns out that $\mathcal{L}(S_N)$ is a witness of the double exponential gap from 1-LAs to 1DFAs and of the exponential gap from D1-LAs to 1DFAs.

We are now going to show that when N is a power of 2, i.e., $N = 2^n$ for some integer $n \geq 0$, then this language needs also N states to be recognized by any 2NFA.

First we recall that a unary language L is said to be *ultimately λ-cyclic* for some integer $\lambda > 0$, if there exists an integer $\mu \geq 0$ such that for each $n \geq \mu$ it holds that $a^n \in L$ if and only if $a^{n+\lambda} \in L$. If, furthermore, for each $\lambda' < \lambda$ the language L is not ultimately λ'-cyclic then L is said to be *properly ultimately λ-cyclic*. A properly ultimately λ-cyclic language L is said to be properly λ-cyclic when $\mu = 0$, i.e., the minimum 1DFA accepting it is a loop of λ states.

The following result has been proved in [5]:

Theorem 9. *Let L be a unary language which is* properly ultimately λ-cyclic*, where $\lambda = p_1^{k_1} p_2^{k_2} \cdots p_s^{k_s}$, with $p_1, p_2, \ldots, p_s$ prime numbers, and $k_1, k_2, \ldots, k_s$ positive integers. Then the number of states in the cycles of each 2NFA accepting L is at least $p_1^{k_1} + p_2^{k_2} + \cdots + p_s^{k_s}$.*

We give the following simple generalization of Theorem 9.

Theorem 10. *Given an alphabet Σ, a language $L \subseteq \Sigma^*$ and symbol $a \in \Sigma$, let $L \cap \{a\}^*$ be* properly ultimately λ-cyclic*, where $\lambda = p_1^{k_1} p_2^{k_2} \cdots p_s^{k_s}$, with $p_1, p_2, \ldots, p_s$ prime numbers and $k_1, k_2, \ldots, k_s$ positive integers. Then the number of states in the cycles of each 2NFA accepting L is at least $p_1^{k_1} + p_2^{k_2} + \cdots + p_s^{k_s}$.*

Proof. Given a 2NFA accepting L, we can obtain a 2NFA accepting $L \cap \{a\}^*$ by removing all transitions on input symbols others than a. Then the lower bound derives from Theorem 9. □

As a consequence of Theorem 10 we obtain that:

Theorem 11. *Each 2NFA accepting $\mathcal{L}(S_{2^n})$ has at least 2^n states.*

Proof. From the transition diagram in Figure 2, we can observe that $\mathcal{L}(S_N) \cap \{a\}^* = \{a^N\}^*$, for each $N > 0$. Hence $\mathcal{L}(S_N) \cap \{a\}^*$ is properly N-cyclic. When $N = 2^n$ for some $n \geq 0$, from Theorem 10 it follows that each 2NFA accepting $\mathcal{L}(S_N)$ should have at least $N = 2^n$ states. □

6. Conclusion

Merging the techniques for the unary case with the techniques for the general case, we have found a family of languages which is a witness of all the size gaps in Figure 1. This family is the same introduced by Meyer and Fischer to prove the optimality of the subset construction and it was described in Section 5.

Several problems are still open. We mention here two problems from Figure 1 (for a further discussion, in particular for other problems related to possible differences between the unary and the general case, we address the reader to [7]): the size cost of elimination of the nondeterminism from 1-LAs, namely the cost of the conversion of 1-LAs into equivalent D1-LAs, and that of the transformation of 1-LAs into equivalent

2DFAs. In both cases we have a double exponential upper bound, deriving from the simulation of 1-LAs by 1DFAs, and an exponential lower bound, which derives from the fact that the cost of the transformation of D1-LAs into 1DFAs is a simple exponential.

We observe that all the block languages presented in Section 3 are accepted by 1-LAs of size $O(n)$. The languages K_n, E_n, and L_n are also accepted by 2DFAs (and hence by D1-LAs) of size exponential in n.

At the moment, in all known examples of languages having an exponential size gap from 1-LAs to D1-LAs we actually know the same gap to 2DFAs. We do not know any example in which there is an exponential size gap from 1-LAs to D1-LAs, but the gap to 2DFAs seems to be larger.

Trying to prove that the size gap from 1-LAs to D1-LAs (or to 2DFAs) is more than an exponential, we considered the following "subset parity" language P_n and its variant P'_n. P_n is a block language in which the last block of each string is the bitwise XOR of some of the previous ones:

$$\begin{aligned} P_n = \{ x_1 \cdots x_k x \mid k > 0,\ & x_1, \ldots, x_k, x \in \{0,1\}^n, \\ & \exists h > 0, i_1, i_2, \ldots, i_h \in \{1, \ldots, k\}, i_1 < i_2 < \ldots < i_h \\ & \text{s.t. } x = x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus x_{i_h} \}, \end{aligned}$$

where, given two strings of bits $x = a_1 a_2 \cdots a_n$ and $y = b_1 b_2 \cdots b_n$ of the same length n, $x \oplus y$ denotes the string obtained by the bitwise XOR of x and y, namely the string $c_1 c_2 \cdots c_n$, where $c_i = a_i \oplus b_i$, $i = 1, \ldots, n$.

In P'_n we remove the *block structure* (in the sense of Section 3): we require that the suffix of length n of a string is the bitwise XOR of *some non overlapping factors* of length n of the corresponding prefix of the string, namely, unlikely as P_n, the "interesting" blocks can start in any position of the string, provided that they are not overlapping:

$$\begin{aligned} P'_n = \{ wx \mid w \in \{0,1\}^*, x \in \{0,1\}^n, \\ \exists h > 0, x_1, x_2, \ldots, x_h \in \{0,1\}^n, y_0, y_1, \ldots, y_h \in \{0,1\}^* \\ \text{s.t. } w = y_0 x_1 y_1 \cdots y_{n-1} x_h y_h \text{ and } x = x_1 \oplus x_2 \oplus \cdots \oplus x_h \}. \end{aligned}$$

Adapting the techniques of Section 3, we can devise 1-LAs of size polynomial in n accepting P_n and P'_n. However, we did not found a way to obtain D1-LAs of size exponential in n accepting these languages: indeed, while in languages E_n, K_n, and L_n we have to check equality of a fixed number of blocks, here we have to compute a more involved operation on a subset of them. So, at a first glance the languages P_n and P'_n seem to be more complicated to be recognized than the other block languages we presented.

Conjecture 1. *The languages P_n and P'_n are not accepted by any D1-LA of size exponential in n.*

References

[1] B. Guillon, L. Prigioniero, Linear-time limited automata. *Theor. Comput. Sci.* **798** (2019), 95–108.

[2] T. N. Hibbard, A generalization of context-free determinism. *Information and Control* **11** (1967) 1/2, 196–238.

[3] M. Kutrib, G. Pighizzini, M. Wendlandt, Descriptional complexity of limited automata. *Inf. Comput.* **259** (2018) 2, 259–276.

[4] M. Kutrib, M. Wendlandt, Reversible limited automata. *Fundam. Inform.* **155** (2017) 1-2, 31–58.

[5] C. Mereghetti, G. Pighizzini, Two-way automata simulations and unary languages. *Journal of Automata, Languages and Combinatorics* **5** (2000) 3, 287–300.

[6] A. R. Meyer, M. J. Fischer, Economy of description by automata, grammars, and formal systems. In: *FOCS*. IEEE, 1971, 188–191.

[7] G. Pighizzini, Limited automata: Properties, complexity and variants. In: *Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Proceedings*. Lecture Notes in Computer Science 11612, Springer, 2019, 57–73.

[8] G. Pighizzini, A. Pisoni, Limited automata and regular languages. *Int. J. Found. Comput. Sci.* **25** (2014) 7, 897–916.

[9] G. Pighizzini, A. Pisoni, Limited automata and context-free languages. *Fundam. Inform.* **136** (2015) 1-2, 157–176.

[10] G. Pighizzini, L. Prigioniero, Limited automata and unary languages. *Information and Computation* **266** (2019), 60–74.

[11] M. O. Rabin, D. Scott, Finite automata and their decision problems. *IBM J. Res. Dev.* **3** (1959) 2, 114–125.

[12] W. J. Sakoda, M. Sipser, Nondeterminism and the size of two way finite automata. In: R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, A. V. Aho (eds.), *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*. ACM, 1978, 275–286.

[13] J. C. Shepherdson, The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.* **3** (1959) 2, 198 –200.

[14] N. J. A. Sloane, The On-Line Encyclopedia of Integer Sequences. http://oeis.org/A007814.

[15] K. W. Wagner, G. Wechsung, *Computational Complexity*. D. Reidel Publishing Company, Dordrecht, 1986.

[16] T. Yamakami, Behavioral strengths and weaknesses of various models of limited automata. In: *SOFSEM 2019: Theory and Practice of Computer Science - 45th International Conference on Current Trends in Theory and Practice of Computer Science, Proceedings*. Lecture Notes in Computer Science 11376, Springer, 2019, 519–530.