Limited Automata and Unary Languages

Giovanni Pighizzini and Luca Prigioniero

Dipartimento di Informatica, Università degli Studi di Milano, Italy
{pighizzini,prigioniero}@di.unimi.it

Abstract. Limited automata are one-tape Turing machines that are allowed to rewrite the content of any tape cell only in the first d visits, for a fixed constant d. When $d = 1$ these models characterize regular languages. An exponential gap between the size of limited automata accepting unary languages and the size of equivalent finite automata is proved. Since a similar gap was already known from unary context-free grammars to finite automata, also the conversion of such grammars into limited automata is investigated. It is proved that from each unary context-free grammar it is possible to obtain an equivalent 1-limited automaton whose description has a size which is polynomial in the size of the grammar. Furthermore, despite the exponential gap between the sizes of limited automata and of equivalent unary finite automata, there are unary regular languages for which d-limited automata cannot be significantly smaller than equivalent finite automata, for any arbitrarily large d.

1 Introduction

The investigation of computational models and of their computational power is a classical topic in computer science. For instance, characterizations of the classes in the Chomsky hierarchy by different types of computational devices are well-known. In particular, the class of context-free languages is characterized in terms of pushwdown automata. A less known characterization of this class has been obtained in 1967 by Hibbard, in terms of Turing machines with rewriting restrictions, called *limited automata* [2]. For each integer $d \geq 0$, a d-limited automaton is a one-tape Turing machine which can rewrite the content of each tape cell *only in the first d visits*. For each $d \geq 2$, the class of languages accepted by these devices coincides with the class of context-free languages, while for $d = 1$ only regular languages are accepted [2, 14].

More recently, limited automata have been investigated from the descriptional complexity point of view, by studying the relationships between the sizes of their descriptions and those of other equivalent formal systems. In [11], it has been proved that each 1-limited automaton M with n states can be simulated by a one-way deterministic automaton with a number of states double exponential in a polynomial in n. The upper bound reduces to a single exponential when M is deterministic. Furthermore, these bounds are optimal, namely, they cannot be reduced in the worst case. In [12], it has been shown how to transform each given 2-limited automaton M into an equivalent pushdown automaton, having a description of exponential size with respect to the description of M. Even for this simulation, the cost cannot be reduced in the worst case. On the other hand, the converse simulation is polynomial in size. In [4], it is proved that the cost of the simulation of d-limited automata by pushdown automata remains exponential even when $d > 2$. A subclass of 2-limited automata, which still characterizes context-free languages but whose members are polynomially related in size with pushdown automata, has been investigated in [10].

In all above mentioned results, lower bounds have been obtained by providing witness languages defined over a binary alphabet. In the unary case, namely in the case of languages defined over a one-letter alphabet, it is an open question if these bounds remain valid. It is suitable to point out that in the unary case the classes of regular and context-free languages collapse [1] and, hence, d-limited automata are equivalent to finite automata for each $d > 0$. The existence of unary 1-limited automata which require a quadratic number of states to be simulated by two-way nondeterministic finite automata has been proved in [11], while in [12] it has been shown that the set of unary strings of length multiple of 2^n, can be recognized by a 2-limited automaton of size $O(n)$, for any fixed $n > 0$. On the other hand, each (even two-way nondeterministic) finite automaton requires a number of states exponential in n to accept the same language. The investigation of the size of unary limited automata has been deepened in [5], where the authors stated several bounds for the costs of the simulations of different variants of unary limited automata by different variants of finite automata. Among these results, they proved the existence of languages accepted by $4n$-states 1-limited automata which require $n \cdot e^{\sqrt{n \ln n}}$ states to be accepted by two-way nondeterministic finite automata.

In this paper we improve these results, by obtaining an exponential gap between unary 1-limited automata and finite automata. We show that for each $n > 1$ the singleton language $\{a^{2^n}\}$ can be recognized by a deterministic 1-limited automaton having $2n+1$ many states and a description of size $O(n)$. Since the same language requires $2^n + 1$ states to be accepted by a one-way nondeterministic automaton, it turns out that the state gap between deterministic 1-limited automata and one-way nondeterministic automata in the unary case is the same as in the binary case. We will also observe that the gap does not reduce if we want to convert unary deterministic 1-limited automata into two-way nondeterministic automata. However, when converting finite automata into limited automata, a size reduction corresponding to such a gap is not always achievable, even if we convert a unary finite automaton into a nondeterministic d-limited automaton for any arbitrarily large d.

In the second part of the paper, we consider unary context-free grammars. The cost of the conversion of these grammars into finite automata has been investigated in [9] by proving exponential gaps. Here, we study the conversion of unary context-free grammars into limited automata. With the help of a result presented in [8], we prove that each unary context-free grammar G can be converted into an equivalent 1-limited automaton whose description has a size which is polynomial in the size of G.

2 Preliminaries

In this section we recall some basic definitions useful in the paper. Given a set S, $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ and a string $w \in \Sigma^*$, let us denote by $|w|$ the length of w and by ε the empty string.

We assume the reader familiar with notions from formal languages and automata theory, in particular with the fundamental variants of finite automata (1DFAs, 1NFAs, 2DFAs, 2NFAs, for short, where 1/2 mean *one-way/two-way* and D/N mean *deterministic/nondeterministic*, respectively) and with *context-free grammars* (CFGs, for short). For further details see, e.g., [3].

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*, $\delta : Q \times \Gamma \to 2^{Q \times (\Gamma \setminus \{\rhd, \lhd\}) \times \{-1,+1\}}$ 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_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 content by X, and move the head to *left*, if $m = -1$, or to *right*, if $m = +1$. Furthermore, the head cannot violate 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 d visits, with the exception of the cells containing the end-markers, which are never modified. For technical details see [12].

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. The language accepted by A is denoted by $L(A)$. A is said to be *deterministic* (d-DLA, for short) whenever $\#\delta(q, \sigma) \leq 1$, for any $q \in Q$ and $\sigma \in \Gamma$.

In this paper we are interested to compare the size of the description of devices and formal systems. As customary, to measure the size of a finite automaton we consider the cardinality of the state set. For a context-free grammar we count the total number of symbols which are used to write down its productions.

For d-limited automata, the size depends on the number q of states and on the cardinality m of the working alphabet. In fact, given these two parameters, the possible number of transitions is bounded by $2q^2m^2$. Hence, if q and m

are polynomial with respect to given parameters, also the size of the d-LA is polynomial.

The costs of the simulations of 1-LAs by finite automata have been investigated in [11], proving the following result:

Theorem 1. *Let M be an n-state* 1-LA*. Then M can be simulated by a* 1NFA *with $n \cdot 2^{n^2}$ states and by a* 1DFA *with $2^{n \cdot 2^{n^2}}$ states. Furthermore, if M is deterministic then an equivalent* 1DFA *with no more than $n \cdot (n+1)^n$ states can be obtained.*

Using witness languages over a binary alphabet, it has been shown that the exponential gaps and the double exponential gap in Theorem 1 cannot be reduced [11].

3 On the Size of Unary Limited Automata

In this section we compare the sizes of unary limited automata with the sizes of equivalent finite automata. Our main result is that unary 1-LAs can be exponentially more succinct than finite automata even while comparing unary *deterministic* 1-LAs with *two-way nondeterministic* automata. However, there are unary regular languages that do not have any d-limited automaton which is significantly more succinct than finite automata, even for arbitrarily large d.

Let us start by showing that, for each $n > 1$, the language $L_n = \{a^{2^n}\}$, which requires $2^n + 1$ states to be accepted by a 1NFA, can be accepted by a 1-DLA of size $O(n)$. Let us proceed by steps. In order to illustrate the construction, first it is useful to discuss how L_n can be accepted by a linear bounded automaton M_n (i.e., a Turing machine that can use as storage the tape which initially contains the input, by rewriting its cells an unbounded number of times).

M_n works in the following way:

i. Starting from the first input symbol, M_n scans the input tape from left to right by counting the as until the right end-marker is reached. Each odd-counted a is overwritten by X. A counter modulo 2 is enough to implement this step.
ii. The previous step is repeated n times, after moving backward the head until reaching the left end-marker. If in one of the first $n-1$ iterations M_n discovers that the number of as at the end of the iterations was odd, then M_n rejects. After the last iteration, M_n accepts if only one a is left on the tape.

It is possible to modify M_n, without any increasing of the number of the states, introducing a different kind of writing at step i.: at the first iteration, the machine uses the symbol 0 instead of X for rewriting, at the second one it uses the symbol 1, and so on. During the last check, the symbol n is written on the last cell of the input tape if it contains an a. If the original input is completely overwritten, then the automaton accepts. For example, in the case $n = 4$, at the end of computation, the content of the input tape will be 0102010301020104.

Considering the last extension of M_n, we are now going to introduce a 1-LA N_n, accepting the language L_n, based on the guessing of the final tape content of M_n.

In the first phase, N_n scans the input tape, replacing each a with a nondeterministically chosen symbol in $\{0, \ldots, n\}$. This requires one state. Next, the machine, after moving backward the head to the left end-marker, makes a scan from left to right for each $i = 0, \ldots, n-1$, where it checks if the symbol i occurs in all odd positions, where positions are counted ignoring the cells containing numbers less than i. This control phase needs three states for each value of i: one for moving backward the head and two for counting modulo 2 the positions containing symbols greater or equal to i. Finally, the automaton checks if only the last input cell contains a n (two states), in such case the input is accepted. The total number of states of N_n is $3(n+1)$, that, even in this case, is linear in the parameter n. This gives us a 1-LA of size $O(n)$ accepting L_n.

We are now going to prove that we can do better. In fact, we will show that switching to the deterministic case for the limited automata model, the size of the resulting device does not increase. Actually, we will slightly reduce the number of states, while using the same working alphabet.

Let us observe that the final tape content of M_n and of N_n, if the input is accepted, corresponds to the first 2^n elements of the *binary carry sequence* [13] defined as follows:

- The first two elements of the sequence are 0 and 1.
- The next elements of the sequence are recursively obtained concatenating the just constructed sequence to a copy of itself and by replacing the last element by its successor.

For example, from 01, concatenating itself and adding 1 to the last element of the obtained sequence, we get 0102, from which, iterating the last procedure it is possible to obtain 01020103, and so on.

Remark 2. Each symbol $0 \leq i < n$ of the binary carry sequence occurs 2^{n-i-1} times, starting in position 2^i and at distance 2^{i+1}, i.e., it occurs in positions $2^i(2j-1)$, for $j = 1, \ldots, 2^{n-i-1}$. Instead, the symbol $i = n$ occurs in position 2^n only.

Remark 2 is a direct consequence of the recursive definition of the sequence. Consider, as an example, the sequence $x = 01020103$: reading x from left to the right, the symbol 0 appears for the first time in position 2^0 and then in positions $3, 5, 7$; 1 in positions $2, 6$; 2 in position 4; and, finally, 3 in position 8.

We will show that this sequence can be generated by a 1-DLA and written on its tape, without counting large indices as 2^i.

To this aim, we introduce the function BIS, that associates with a given sequence of integers $s = \sigma_1\sigma_2\cdots\sigma_j$, its *backward increasing sequence*, namely the longest strictly increasing sequence obtained taking the elements of s starting from the end, and using the greedy method. Formally, $BIS(\sigma_1\sigma_2\cdots\sigma_j) = (i_1, i_2, \ldots, i_r)$, $j, r > 0$, if and only if $i_1 = \sigma_{h_1}, i_2 = \sigma_{h_2}, \ldots, i_n = \sigma_{h_r}$ where $h_1 =$

j, $h_t = \max\{h' < h_{t-1} \mid \sigma_{h'} > \sigma_{h_{t-1}}\}$ for $t = 2, \ldots, r$, and $\sigma_{h'} < \sigma_{h_r}$ for $0 < h' < h_r$.

For example, for $j = 11$, $BIS(01020103010) = (0, 1, 3)$. Notice that in the binary representation of j, namely 1011, the bits set to 1 occur, respectively, in position 0, 1, and 3. This fact is true for each j, as proved in the following lemma, i.e., the value of BIS, applied to the first j elements of the binary carry sequence, are the positions of bits equal to 1 in the binary representation of j, from the less significative bit.

Lemma 3. *Let $\sigma_1\sigma_2\cdots\sigma_j$ be the first j elements of the binary carry sequence, $j > 0$. If $BIS(\sigma_1\sigma_2\cdots\sigma_j) = (i_1, i_2, \ldots, i_r)$ then $j = \sum_{t=1}^{r} 2^{i_t}$.*

Proof. Considering the recursive definition of the binary carry sequence, we can proceed by induction on j, where the basis $j = 1, 2$ is trivial from the definition.

- If j is a power of 2, namely $j = 2^k$, for some $k \geq 0$, then, considering Remark 2 with $n = k$, k is the maximum number in the sequence and it occurs in position j only. So, $BIS(\sigma_1\sigma_2\cdots\sigma_j) = (k)$.
- If j is not a power of 2, namely $2^k < j < 2^{k+1}$, $j = 2^k + j'$ for some $k > 0$, $0 < j' < 2^k$, then, by construction, k is the maximum number which occurs in the sequence, and the elements $\sigma_{2^k+1}\cdots\sigma_j$ are equal to the first j' elements $\sigma_1\sigma_2\cdots\sigma_{j'}$. So, $BIS(\sigma_1\sigma_2\cdots\sigma_j)$ is obtained by appending k at the end of $BIS(\sigma_1\sigma_2\cdots\sigma_{j'})$.
  Let $BIS(\sigma_1\sigma_2\cdots\sigma_{j'}) = (i_1, \ldots, i_{r'})$, $r' = r - 1$. Then $BIS(\sigma_1\sigma_2\cdots\sigma_j) = (i_1, \ldots, i_{r'}, i_r)$, $i_r = k$. Furthermore, by induction hypothesis, $j' = \sum_{t=1}^{r'} 2^{i_t}$.
  Then $j = 2^k + j' = 2^k + \sum_{t=1}^{r'} 2^{i_t} = \sum_{t=1}^{r} 2^{i_t}$. □

We are going to define a 1-DLA $A_n = (Q, \Sigma, \Gamma, \delta, q_1, F)$ accepting the language L_n. The automaton A_n works by replacing the content of cell i, in the first visit, by the symbol σ_i of the binary carry sequence.

Remark 4. Given the first $i-1$ elements of the binary carry sequence $\sigma_1\sigma_2\cdots\sigma_{i-1}$, it is possible to deterministically determine the next element σ_i only looking at the previous elements of the sequence. In particular, σ_i has to be determined so that $BIS(\sigma_1\sigma_2\cdots\sigma_i)$ is the positions of bits equal to 1 in the binary representation of i. This can be obtained computing from the binary representantion of $i - 1$ the representation of i. Hence, σ_i is the smallest integer greater than or equal to 0 not occurring in $BIS(\sigma_1\sigma_2\cdots\sigma_{i-1})$.

The automaton A_n implements the procedure summarized in Algorithm 1 — note that, for ease of presentation, the algorithm assumes that the machine starts the computation with the head on the left end-marker — and it is defined as follows: $Q = \{q_I, q_F, q_1, \ldots, q_n, p_1, \ldots, p_{n-1}\}$, $\Sigma = \{a\}$, $\Gamma = \{0, \ldots, n\}$, q_I is the initial state and q_F is the unique final one. The transitions in δ are defined as follows (the remaining transitions are undefined):

i. $\delta(q_I, a) = (p_1, 0, -1)$

Algorithm 1: Recognition of the language L_n

```
1  start with the head on the left end-marker
2  while symbol under the head $\neq n$ do
3      move the head to the right
4      write 0
5      $j \leftarrow 0$
6      repeat
7          while symbol under the head $\leq j$ and $\neq \triangleright$ do
8              move the head to the left
9          $j \leftarrow j + 1$
10     until symbol under the head $\neq j$
11     repeat
12         move the head to the right
13     until symbol under the head $= a$
14     write $j$
15 move the head to the right
16 if symbol under the head $= \triangleleft$ then ACCEPT
17 else REJECT
```

ii. $\delta(p_i, \sigma) = (p_i, \sigma, -1)$, for $i = 2, \ldots, n-1$ and $\sigma < i-1$
iii. $\delta(p_i, i) = (p_{i+1}, i, -1)$, for $i = 1, \ldots, n-2$
iv. $\delta(p_i, \sigma) = (q_i, \sigma, +1)$, for $i = 1, \ldots, n-1$ and ($\sigma > i$ or $\sigma = \triangleright$)
v. $\delta(p_{n-1}, n-1) = (q_n, n-1, +1)$
vi. $\delta(q_i, \sigma) = (q_i, \sigma, +1)$, for $i = 1, \ldots, n$ and $\sigma < i$
vii. $\delta(q_i, a) = (q_I, i, +1)$, for $i = 1, \ldots, n-1$
viii. $\delta(q_n, a) = (q_F, n, +1)$
ix. $\delta(q_F, \triangleleft) = (q_F, \triangleleft, +1)$

We finally observe that A_n has $2n + 1$ states, which is linear in the parameter n.

The machine starts in the initial state q_I. Since each symbol $\sigma \neq 0$ is preceded by 0 (a 0 occurs in each odd position), the automaton moves the head to the right and writes a 0 before of each symbol in $\Gamma \setminus \{0\}$ (transition i. – lines 3 and 4). Everytime the head is in a odd position p, the automaton has to look backward for the minimum integer j such that j is not in $BIS(\sigma_1, \ldots, \sigma_p)$. This is done with transitions from ii. to v. – lines from 6 to 10. After that, A_n moves its head to the right until the first a is reached (transitions vi. – lines from 11 to 13) and writes the symbol j (transitions vii. – line 14). This is repeated until the symbol n is written on the input tape. At this point is sufficient to verify if the next symbol on the input tape is the right end-marker: in this case, the automaton accepts (transitions viii. and ix. – lines from 15 to 17).

Hence we conclude that the language L_n is accepted by a 1-DLA with $O(n)$ many states, while it is an easy observation that each 1NFA accepting it requires $2^n + 1$ states. We can even obtain a stronger result by proving that between unary 1-DLAs and 2NFAs, there is the same gap. This gives the main result of this section:

Theorem 5. *For each integer $n > 1$ there exists a unary language K_n such that K_n is accepted by a deterministic* 1-LA *with $O(n)$ states and a working alphabet of size $O(n)$ while each* 2NFA *accepting it requires 2^n states.*

Proof. (outline) With some minor changes, the above presented automaton A_n can accept $K_n = \{a^{2^n}\}^*$. From Theorem 9 in [7], each 2NFA requires at least 2^n many states to accept the same language. □

We conclude this section, by proving that the exponential gap between unary limited automata and finite automata is not always achievable.

Theorem 6. *There exist constants c, n_0 such that for all integers $n \geq n_0$ there exists a unary* 1DFA *accepting a finite language L with at most n states, such that for any d-*LA *accepting L with $d > 0$, q states, and a working alphabet of m symbols, it holds that $qm \geq cn^{1/2}$.*

Proof. There are $2^{O(q^2m^2)}$ different limited automata such that the cardinalities of the set of states and of the working alphabet are bounded by q and m, respectively. On the other hand, the number of different subsets of $\{a^0, a^1, \ldots, a^{n-1}\}$ is 2^n. Hence $kq^2m^2 \geq n$ for a constant $k > 0$ and each sufficiently large n, which implies $qm \geq cn^{1/2}$, where $c = 1/k^{1/2}$. Note that each subset of $\{a^0, a^1, \ldots, a^{n-1}\}$ is accepted by a (possibly incomplete) 1DFA with at most n states. □

Notice that the result in Theorem 6 does not depend on d, i.e., the lower bound holds even taking an arbitrarily large d. In the case $d = 1$, the argument in the proof can be refined to show that $qm^{1/2} \geq cn^{1/2}$.

4 Unary Grammars versus Limited Automata

In Section 3 we proved an exponential gap between unary 1-LAs and finite automata. A similar gap was obtained between unary CFGs and finite automata [9]. Hence, it is natural to study the size relationships between unary CFGs and 1-LAs. Here, we prove that each context-free grammar specifying a unary language can be converted into an equivalent 1-LA which has a set of states and a working alphabet whose sizes are polynomial with respect to the description of the grammar.

Let us start by presenting some notions and preliminary results useful to reach our goal.

Definition 7. *A* bracket alphabet *Ω_b is a finite set containing an even number of symbols, say $2k$, with $k > 0$, where the first k symbols are interpreted as* left brackets *of k different types, while the remaining symbols are interpreted as the corresponding* right brackets. *The* Dyck language *D_{Ω_b} over Ω_b is the set of all sequences of balanced brackets from Ω_b.*

An extended bracket alphabet *Ω is a nonempty finite set which is the union of two, possibly empty, sets Ω_b and Ω_n, where Ω_b, if not empty, is a bracket* *alphabet, and Ω_n is a set of* neutral *symbols. The* extended Dyck language $\widehat{D}_\Omega$ *over Ω is the set of all the strings that can be obtained by arbitrarily inserting symbols from Ω_n in strings of D_{Ω_b}. Given an integer $d > 0$, the* extended Dyck language with nesting depth bounded by d *over Ω, denoted as $\widehat{D}_\Omega^{(d)}$ is the subset of $\widehat{D}_\Omega$ consisting of all strings where the nesting depth of brackets is at most d.*

Example 8. Let $\Omega_b = \{ (\, , [\, ,)\, ,] \}$, $\Omega_n = \{ | \}$, and $\Omega = \Omega_b \cup \Omega_n$.
Then $([[ ]]) [ ] \in D_{\Omega_b} \subset \widehat{D}_\Omega$, $| (| [[ ] |]) [|] \in \widehat{D}_\Omega^{(3)} \setminus \widehat{D}_\Omega^{(2)}$.

It is well-known that Dyck languages, and so extended Dyck languages, are context-free and nonregular. However, the subset obtained by bounding the nesting depth by each fixed constant is regular. We are interested in the recognition of such languages by "small" two-way automata:

Lemma 9. *Given an extended bracket alphabet with k types of brackets and an integer $d > 0$, the language $\widehat{D}_\Omega^{(d)}$ can be recognized by a* 2DFA *with $O(k \cdot d)$ many states.*

Proof. We can define a 2DFA M which verifies the membership of its input w to $\widehat{D}_\Omega^{(d)}$ by making use of a counter c. In a first scan M checks whether or not the brackets are correctly nested, *regardless* their types. This is done as follows. Starting with 0 in c, M scans the input from left to right, incrementing the counter for each left bracket and decrementing it for each right bracket. If during this process the counter exceeds d or becomes negative then M rejects. M also rejects if at the end of this scan the value which is stored in the counter is positive.

In the remaining part of the computation, M verifies that the corresponding left and right brackets are of the same type. To this aim, starting from the left end-marker, M moves its head to the right, to locate a left bracket. Then, it moves to the right to locate the corresponding right bracket in order to check if they are of the same type. To this aim, M uses the counter c, which initially contains 0 and increments or decrements it for each left or right bracket to the right of the one under consideration. When a cell containing a right bracket is reached with 0 in c, M checks if it is matching with the left bracket. If this is not the case, then M stops and rejects. Otherwise, M should move back its head to the matched left bracket in order to continue the inspection. This can be done by moving the head to the left and incrementing or decrementing the counter for each right or left bracket, respectively, up to reach a cell containing a left bracket when 0 is in c.

This process is stopped when the right end-marker is reached and all pairs of brackets have been inspected. Notice that neutral symbols are completely ignored.

In its finite control, M keeps the counter c, that can assume $d + 1$ different values, and remembers the type of the left bracket, to verify the matching with the corresponding right bracket and then to move back the head to the left bracket. This gives $O(k \cdot d)$ many states. □

The following nonerasing variant of the Chomsky-Schützenberger representation theorem for context-free languages, proved by Okhotin [8], is crucial to obtain our main result:

Theorem 10. *A language $L \subseteq \Sigma^*$ is context-free if and only if there exist an extended bracket alphabet Ω_L, a regular language $R_L \subseteq \Omega_L^*$ and a letter-to-letter homomorphism $h : \Omega_L \to \Sigma$ such that $L = h(\widehat{D}_{\Omega_L} \cap R_L)$.*

In [10], it was observed that the language R_L of Theorem 10 is local[1] and the size of the alphabet Ω is polynomial with respect to the size of a context-free grammar G generating L. This was used to prove that each context-free grammar G can be transformed into an equivalent *strongly limited automaton* (a special kind of 2-LA) whose description has polynomial size with respect to the description of G. In the following, when L is specified by a context-free grammar G, i.e., $L = L(G)$, we will write Ω_G and R_G instead of Ω_L and R_L, respectively.

Our goal, here, is to build 1-LAs of polynomial size from unary context-free grammars. To this aim, using the fact that factors in unary strings commute, by adapting the argument used to obtain Theorem 10, we prove the following result:

Theorem 11. *Let $L \subseteq \{a\}^*$ be a unary regular language and $G = (V, \{a\}, P, S)$ be a context-free grammar of size s generating it. Then, there exist an extended bracket alphabet Ω_G and a regular language $\widehat{R}_G \subseteq \Omega_G^*$ such that $L = h(\widehat{D}_{\Omega_G}^{(\#V)} \cap \widehat{R}_G)$, where:*

- *$\widehat{D}_{\Omega_G}^{(\#V)}$ is the extended Dyck language over Ω_G with nesting depth bounded by $\#V$,*
- *h is the letter-to-letter homomorphism from Ω_G to $\{a\}$.*

Furthermore, the size of Ω_G is polynomial in the size s of the grammar G and the language $\widehat{R}_G$ is recognized by a 2NFA with a number of states polynomial in s.

Proof (outline). Given a context-free grammar $G = (V, \{a\}, P, S)$ specifying a unary language L, we first obtain the representation in Theorem 10. According to Theorem 5.2 in [10], the size of the alphabet Ω_G is polynomial with respect to the size of the description of G. Each pair of brackets in Ω_G represents the root of a derivation tree of G, which starts from a certain variable of G and produces a terminal string.

If a sequence $w \in \Omega_G^*$ contains a pair of brackets corresponding to a variable A which is nested, at some level, in another pair corresponding to the same variable, then w can be replaced by a sequence w', of the same length, which is obtained by replacing the factor of w delimited by the outer pair of brackets corresponding

[1] A language L is *local* if there exist sets $\mathcal{A} \subseteq \Sigma \times \Sigma$, $\mathcal{I} \subseteq \Sigma$, and $\mathcal{F} \subseteq \Sigma$ such that $w \in L$ if and only if all factors of length 2 in w belong to $\mathcal{A}$ and the first and the last symbols of w belong to $\mathcal{I}$ and $\mathcal{F}$, respectively [6].

to A, by the factor delimited by the inner pair, and by moving the removed part at the end of w. For instance, $w = (_S\,(_A\,(_B\,)_B\,(_C\,(_A\,(_B\,)_B\,)_A\,)_C\,)_A\,)_S$ can be replaced by $w' = (_S\,(_A\,(_B\,)_B\,)_A\,)_S\,(_A\,(_B\,)_B\,(_C\,)_C\,)_A$, where, for the sake of simplicity, subscripts represent variables corresponding to brackets. In this way, each time the nesting depth is greater than $\#V$, it can be reduced by repeatedly moving some part to the end. So, from each string in $\widehat{D}_{\Omega_G}$, we can obtain an "equivalent" string of the same length in $\widehat{D}^{(\#V)}_{\Omega_G}$.

The regular language R_G should be modified accordingly. While in the representation in Theorem 10, the first and the last symbol of a string $w \in \widehat{D}_{\Omega_G} \cap R_G$ represent a matching pair corresponding to the variable S, after the above transformation, valid strings should correspond to sequences of blocks of brackets where the first block represents a derivation tree of a terminal string from S, while each of the subsequent blocks represents a *gap tree* from a variable A, namely a tree corresponding to a derivation of the form $A \overset{+}{\Rightarrow} a^i A a^j$, with $i + j > 0$, where A already appeared in some of the previous blocks. This condition, together with the conditions on R_G, can be verified by a 2NFA with a polynomial number of states. $\square$

Notice that if we omit the state bound for the 2NFA accepting $\widehat{R}_G$, the statement of Theorem 11 becomes trivial: just take $L = R_G$ and $\widehat{\Omega}_G = \{a\}$ where a is a neutral symbol.

Using Theorem 11, we now prove the main result of this section:

Theorem 12. *Each context-free grammar of size s generating a unary language can be converted into an equivalent* 1-LA *having a size which is polynomial in s.*

Proof. Let $G = (V, \{a\}, P, S)$ be the given grammar, $L \subseteq \{a\}^*$ be the unary language generated by it, Ω_G be the extended bracket alphabet and $\widehat{R}_G$ be the regular language obtained from G according to Theorem 11.

We define a 1-LA M which works in the following steps:

1. M makes a complete scan of the input tape from left to right, by rewriting each input cell by a nondeterministically chosen symbol from Ω_G. Let $w \in \Omega_G^*$ be the string written on the tape at the end of this phase.
2. M checks whether or not $w \in \widehat{D}^{(\#V)}_{\Omega_G}$.
3. M checks whether or not $w \in \widehat{R}_G$.
4. M accepts if and only if the outcomes of steps 2 and 3 are both positive.

According to Lemma 9, step 2 can be done by simulating a 2DFA with $O(\#\Omega_G \cdot \#V)$ many states, hence a number polynomial in s. Furthermore, by Theorem 11, also step 3 can be performed by simulating a 2NFA with a number of states polynomial in s. Hence M has a size which is polynomial in s. $\square$

We point out that from Theorem 12 and the exponential gap from unary CFGs to 1NFAs proved in [9], we can derive an exponential gap from unary *nondeterministic* 1-LAs to 1NFAs. In Section 3 we proved that the gap remains exponential if we restrict to unary *deterministic* 1-LAs and consider equivalent 2NFAs.

5 Conclusion

In [11], using languages defined over a binary alphabet, exponential size gaps have been proved for the conversion of 1-LAs into 2NFAs and of 1-DLAs into 1DFAs. As a consequence of our results, these exponential gaps hold even if we restrict to unary languages. On the other hand, the size gap between 1-LAs and 1DFAs is double exponential. Even in this case, the proof in [11] relies on witness languages defined over a binary alphabet. We leave as an open question to investigate whether or not a double exponential gap is possible between 1-LAs and 1DFAs even in the unary case.

Another question we leave open is whether or not 1-LAs and CFGs are polynomially related in the unary case. While in Section 4 we proved that from each unary CFG we can build a 1-LA of polynomial size, at the moment we do not know the converse relationship.

References

1. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. J. ACM 9(3), 350–371 (1962), http://doi.acm.org/10.1145/321127.321132
2. Hibbard, T.N.: A generalization of context-free determinism. Information and Control 11(1/2), 196–238 (1967)
3. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
4. Kutrib, M., Pighizzini, G., Wendlandt, M.: Descriptional complexity of limited automata. Information and Computation. To appear.
5. Kutrib, M., Wendlandt, M.: On simulation cost of unary limited automata. In: Shallit, J., Okhotin, A. (eds.) DCFS 2015. Lecture Notes in Computer Science, vol. 9118, pp. 153–164. Springer (2015), http://dx.doi.org/10.1007/978-3-319-19225-3
6. McNaughton, R., Papert, S.A.: Counter-Free Automata (M.I.T. Research Monograph No. 65). The MIT Press (1971)
7. Mereghetti, C., Pighizzini, G.: Two-way automata simulations and unary languages. Journal of Automata, Languages and Combinatorics 5(3), 287–300 (2000)
8. Okhotin, A.: Non-erasing variants of the Chomsky-Schützenberger theorem. In: Yen, H., Ibarra, O.H. (eds.) DLT 2012, Lecture Notes in Computer Science, vol. 7410, pp. 121–129. Springer (2012), http://dx.doi.org/10.1007/978-3-642-31653-1
9. Pighizzini, G., Shallit, J., Wang, M.: Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. J. Computer and System Sciences 65(2), 393–414 (2002)
10. Pighizzini, G.: Strongly limited automata. Fundam. Inform. 148(3-4), 369–392 (2016), http://dx.doi.org/10.3233/FI-2016-1439
11. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. Int. J. Found. Comput. Sci. 25(7), 897–916 (2014), http://dx.doi.org/10.1142/S0129054114400140
12. Pighizzini, G., Pisoni, A.: Limited automata and context-free languages. Fundam. Inform. 136(1-2), 157–176 (2015), http://dx.doi.org/10.3233/FI-2015-1148
13. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. http://oeis.org/A007814
14. Wagner, K.W., Wechsung, G.: Computational Complexity. D. Reidel Publishing Company, Dordrecht (1986)