Fundamenta Informaticae XXI (2001) 1001–1021
DOI 10.3233/FI-2012-0000
IOS Press

Limited Automata and Context-Free Languages*

Giovanni Pighizzini
Dipartimento di Informatica
Università degli Studi di Milano
via Comelico 39, 20135 Milano, Italy
pighizzini@di.unimi.it

Andrea Pisoni
Dipartimento di Informatica
Università degli Studi di Milano
via Comelico 39, 20135 Milano, Italy
andrea.pisoni1@studenti.unimi.it

Abstract. Limited automata are one-tape Turing machines which are allowed to rewrite each tape cell only in the first d visits, for a given constant d. For each $d \geq 2$, these devices characterize the class of context-free languages. We investigate the equivalence between 2-limited automata and pushdown automata, comparing the relative sizes of their descriptions. We prove exponential upper and lower bounds for the sizes of pushdown automata simulating 2-limited automata. In the case of the conversion of deterministic 2-limited automata into deterministic pushdown automata the upper bound is double exponential and we conjecture that it cannot be reduced. On the other hand, from pushdown automata we can obtain equivalent 2-limited automata of polynomial size, also preserving determinism. From our results, it follows that the class of languages accepted by deterministic 2-limited automata coincides with the class of deterministic context-free languages.

Keywords: finite automata; formal languages; Turing machines; deterministic context-free languages; descriptional complexity.

*Partially supported by MIUR under the project PRIN "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi", code H41J12000190001.

1. Introduction

In formal language theory [7, 10], each class of the *Chomsky hierarchy* is defined in terms of grammars using some special kind of productions, where the form of the productions which are allowed for grammars of type k ($k = 1, 2, 3$) is a restriction of the form used for grammars of type $k - 1$. From the point of view of language acceptors, each class of the hierarchy is characterized by some kind of device. However, while *linear bounded automata*, used to characterize context-sensitive languages (type 1) and *finite state automata*, used to characterize regular languages (type 3), can be seen as restrictions of (one-tape) Turing machines, which characterize type 0 languages, for context-free languages (type 2) the characterization in terms of *pushdown automata* is usually presented. Devices of this kind are very useful to investigate and manipulate context-free languages. They also emphasize the main difference between regular and context-free languages, namely the possibility of representing recursive structures which, in terms of accepting devices, corresponds to increase the power of finite automata by adding a pushdown store. However, in a hierarchical view, pushdown automata do not appear as a special case of linear bounded automata.

Almost half a century ago, Hibbard discovered a different characterization of context-free languages, which uses a restricted version of Turing machines, called *scan limited automata* or, simply, *limited automata* [5]. For each integer $d \geq 0$, a d-limited automaton is a one-way nondeterministic Turing machine which is allowed to rewrite the content of each tape cell *only in the first d visits*. He proved that, for each $d \geq 2$, the class of languages accepted by d-limited automata coincides with the class of context-free languages. Furthermore, since restricting these devices to use only the part of the tape containing the input string, without any available extra storage, does not reduce their computational power, these models can be seen as a restriction of linear bounded automata and, clearly, they are extensions of finite state automata. Hence, together with the above mentioned models related to type 1 and type 3 languages, this gives a hierarchy of classes of Turing machines corresponding to the Chomsky hierarchy.

In this paper we investigate the case of 2-limited automata (2-LAs, for short). We revise and refine some of the results presented by Hibbard, also considering descriptional complexity aspects. First of all, we show how to transform each 2-LA into an equivalent pushdown automaton (PDA). The PDA obtained from this construction has a description whose size is exponential with respect to the size of the description of the given 2-LA. By providing a suitable family of witness languages, we show that such an exponential gap cannot be reduced, even when the given 2-LA is deterministic. With a minor modification, we also obtain a transformation, of the same size cost, which preserves the determinism, if the right end of the input is marked. This restriction can be removed at the price of a further exponentiation of the size. We conjecture that this gap cannot be reduced.

We also study converse simulations. We provide a simulation of PDAs by 2-LAs and also a simulation preserving determinism. Both simulations are polynomial in the size of the descriptions. The main ideas behind them are similar, however, the simulation in the deterministic case uses a lot of technical details which are essentially due to the impossibility of completely removing ϵ-transitions from deterministic pushdown automata.

As a consequence of these results, we obtain that deterministic 2-limited automata characterize the

class of deterministic context-free languages, thus solving a problem left open in [5]. As already mentioned, even for each $d > 2$, languages accepted by d-limited automata are context-free. However, deterministic d-limited automata with $d > 2$ can recognize context-free languages which are not deterministic. As a matter of fact, they define an infinite hierarchy of languages [5]. On the other hand, 0-limited automata clearly accept only regular languages. This is also true for 1-limited automata, as proven in [11]. This result has been recently revised in [8] by investigating descriptional complexity aspects. Among them, a double exponential gap between the size of nondeterministic 1-limited automata and deterministic one-way finite automata has been proved.

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. For each integer $n > 0$ and language $L \subseteq \Sigma^*$, $L^{<n}$ denotes the set of strings of length less than n in L and L^n the set of strings of length n in L

We assume the reader familiar with notions from formal languages and automata theory, in particular with the notions of *finite automaton*, *pushdown automaton* (PDA, for short), *deterministic pushdown automaton* (DPDA, for short), *context-free grammar*, and *Greibach* and *Chomsky normal forms* (see, e.g., [7, 10]). We remind the reader that, while in the nondeterministic case pushdown automata accepting by final states and by empty stack are equivalent, in the deterministic case acceptance by final states is more powerful. Hence, for DPDAs we always refer to such a condition. In particular, DPDAs accepting by final states define the class of *deterministic context-free languages*.

We now introduce the main model we are interested in. Given an integer $d \geq 0$, a *d-limited automaton* (d-LA, for short) 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 $\Sigma \cup \{\rhd, \lhd\} \subseteq \Gamma$, with $\rhd, \lhd \notin \Sigma$ 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_0. In one move, according to δ and to the current state, $\mathcal{A}$ reads a symbol from the tape, changes its state, replaces the symbol just read from the tape by a new symbol, and moves its head to one position forward or backward. In particular, $(q, X, m) \in \delta(p, a)$ 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 subject to some restrictions, which, essentially, allow to modify the content of a cell only during the first d visits. To this aim, the alphabet Γ is partitioned into $d + 1$ sets $\Gamma_0, \Gamma_1, \ldots, \Gamma_d$, where $\Gamma_0 = \Sigma$ and $\rhd, \lhd \in \Gamma_d$. With the exception of the cells containing the end-markers, which are never modified, at the beginning all the cells contain symbols from $\Gamma_0 = \Sigma$. In the k-th visit to a tape

cell, the content of the cell is rewritten by a symbol from Γ_k, up to $k = d$, when the content of the cell is "frozen", i.e., after that, the symbol in the cell cannot be changed further. Actually, on a cell we do not count the visits, but the scans from left to right (corresponding to odd numbered visits) and from right to left (corresponding to even numbered visits). Hence, a move reversing the head direction is counted as a double visit for the cell where it occurs. In this way, when a cell c is visited for the kth time, with $k \leq d$, its content is a symbol from Γ_{k-1}. If the move does not reverse the head direction, then the content of the cell is replaced by a symbol from Γ_k. However, if the head direction is reversed, then in this double visit the symbol is replaced by a symbol from Γ_{k+1}, when $k < d$, and by a symbol of Γ_d that after then is frozen, otherwise.

Formally, for each $(q, \gamma, m) \in \delta(p, \sigma)$, with $p, q \in Q$, $\sigma \in \Gamma_k$, $\gamma \in \Gamma_h$, $m \in \{-1, +1\}$, we require the following:

- if $k = d$ then $\sigma = \gamma$ and $k = h$,
- if $k < d$ and $m = +1$ then $h = \min(\lceil \frac{k}{2} \rceil \cdot 2 + 1, d)$;
- if $k < d$ and $m = -1$ then $h = \min(\lceil \frac{k+1}{2} \rceil \cdot 2, d)$.

An automaton $\mathcal{A}$ is said to be *limited* if it is d-limited for some $d \geq 0$. $\mathcal{A}$ 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 violating the right end-marker. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. $\mathcal{A}$ is said to be *deterministic* 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. (For surveys in the area of descriptional complexity see, e.g., [3, 6].) According to the discussion in [4] the size of a PDA should be defined depending on the number of the states, the cardinality of the pushdown alphabet, the number of input symbols, and the maximum number of stack symbols appearing on the right hand side of transition rules. If a PDA M in one transition can increase the height of its pushdown store at most by one and the input alphabet is fixed, then the size of the description of M is polynomial in the cardinalities of the state set and pushdown alphabet. Hence, the size of M can be studied with respect to these two parameters. In a similar way, for context-free grammars in Chomsky normal form, the total size of the description is polynomial in the cardinality of the set of variables.

For d-limited automata, the size depends on the number s of states and on the cardinality g of the working alphabet. In fact, given these two parameters, the possible number of transitions is bounded by $2s^2g^2$. Hence, if s and g are polynomial with respect to given parameters, also the size of the d-LA is polynomial.

Observe that 0-LAs are exactly two-way finite automata. Furthemore, 1-LAs have the same power as finite automata [11], but they can be double exponentially more succinct [8].

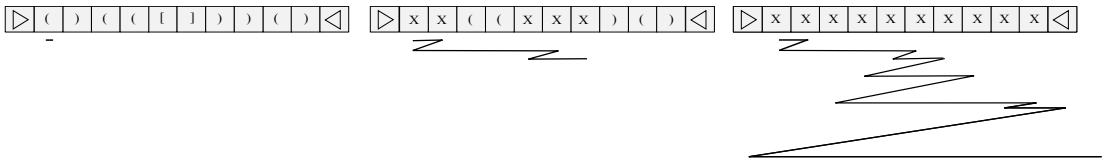

Figure 1. Some steps in an accepting computation of the automaton A of Example 3.1 on input $()((\lceil\rceil))()$.

3. Some examples

In this section we present a few examples of language recognition using limited automata. Some of them will turn out to be useful in the following of the paper. For the sake of simplicity, the d-limited automata we describe in these examples do not necessarily rewrite the content of tape cells during each kth visit, with $k < d$. However, with an easy modification, they can be turned into the form described in Section 2.

Example 3.1. The *Dyck language* D_k over the alphabet $\{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ of $k \geq 1$ types of brackets, namely the set of strings representing well balanced sequences of brackets, can be recognized by a 2-LA A which implements the following procedure. A starts scanning the tape until to find a closed bracket $)_i$. A substitutes $)_i$ with the symbol $X \in \Gamma_2$ and changes the head direction. In a similar way, it stops when it meets the first left bracket $(_j$. If $i \neq j$, i.e., the two brackets are not of the same type, then A rejects. Otherwise, it writes X on the cell and changes again the head direction moving to the right. This procedure is repeated until A does not reach one of the end-markers. (See Figure 1.)

- If the left end-marker is reached, then it means that at least one of the right brackets in the input w does not have a matching left bracket. Hence, A rejects.
- If instead the right end-marker is reached, then A has to make sure that every left bracket has a matching right one. In order to do this, it scans the entire tape from the right to the left and, if it finds a left bracket not marked with X, then A rejects. On the other hand, if A reaches the left end-marker reading only Xs, it enters a state q and scans again the whole tape from the left to the right, A then accepts after violating the right end-marker.

In the following example, we use a 2-LA to recognize a regular language.

Example 3.2. For each integer n, let us denote by K_n the set of all strings on the alphabet $\{0, 1\}$ consisting of the concatenation of blocks of length n, such that at least n blocks are equal to the last one. Formally:

$$\begin{aligned} K_n \;=\; & \{x_1 x_2 \cdots x_k x \mid k \geq 0,\ x_1, x_2, \ldots, x_k, x \in \{0,1\}^n, \\ & \exists i_1 < i_2 < \cdots < i_n \in \{1, \ldots, k\},\ x_{i_1} = x_{i_2} = \ldots = x_{i_n} = x\}\,. \end{aligned}$$

We now describe a 2-LA M accepting K_n. Suppose M receives an input string w of length N.

1. First, M scans the input tape from left to right, to reach the right end-marker.

2. M moves it head $n+1$ positions to the left, namely to the cell $i = N - n$, the one immediately to the left of the input suffix x of length n.

3. Starting from this position i, M counts how many blocks of length n coincide with x. This is done as follows.

   When M, arriving from the right, visits a position $i \leq N - n$ for the first time, it replaces the content a by a special symbol X, after copying a in the finite control. Hence, M starts to move to the right, in order to compare the symbol removed from the cell with the corresponding symbol in the block x. While moving to the right, M counts modulo n and stops when the counter is 0 and a cell containing a symbol other than X is reached. The symbol of x is this cell has to be compared with a. Then, M moves to the left until reach cell $i-1$, namely the first cell which does not contain X, immediately to the left of cells containing X.

   We observe that the end of a block is reached each time a symbol a copied from the tape is compared with the leftmost symbol of x, which lays immediately to the right of a cell containing X. Each time the end of a block is detected, the counter of blocks matching with x is incremented, if in the block just inspected no mismatches have been discovered.

4. When the left end-marker is reached, M accepts if and only if the input length is a multiple of n and the number of blocks matching with x is n.

We can easily observe that the above strategy can modify tape cells only in the first two visits. Furthermore, it can be implemented by a *deterministic* 2-LA which uses $O(n^2)$ states and a constant size alphabet.

Using standard distinguishability arguments, it can be proved that each one-way deterministic automaton accepting language K_n requires a number of states double exponential in n. (A slightly different language L_n which is accepted by a nondeterministic 1-LA with $O(n)$ states, but requires a number of states double exponential in n to be accepted by a one-way deterministic automaton is presented in [8].)

Example 3.3. Let us consider the following language:

$$L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}.$$

To recognize L, a limited automaton A can scan the tape from left to right, to locate the right end-marker and then it can read the last input symbol. If the symbol is a c, then A has to verify whether or not the number of the as coincides with the number of the bs. To this aim, A moves to the left, until to find a cell containing an a, that is rewritten by a symbol X. Then it moves to the right to search a cell containing a b that will be also rewritten by the symbol X. For inputs in the language, the automaton will reach the left end-marker while searching an a. If the last symbol of the input is a d the only difference in the procedure is that for each a the automaton A has to mark two letters b.

It can be easily observed that A can rewrite each tape cell only in the first three visits. Furthermore, the procedure is deterministic. Hence, A is a *deterministic* 3-LA.

We can also give a nondeterministic procedure which firstly guesses the last input symbol, then compares bs with as using a procedure similar to the above described one and, finally, verifies if the initial guess was correct. In this way, we obtain a *nondeterministic* 2-LA.

The following example shows the recognition of a unary (regular) language using the power of 2-LAs:

Example 3.4. For each integer $k > 0$, we present a 2-LA M accepting the language $(a^{2^k})^*$, namely the set of all multiples of 2^k, written in unary notation. M is defined by $Q = \{p_1, p_2, \ldots, p_k, p_B, p_F\}$, $\Gamma = \Gamma_1 \cup \Gamma_2$, with $\Gamma_1 = \{X_2, \ldots, X_k\}$, $\Gamma_2 = \{X_1, \rhd, \lhd\}$, p_k is the initial and unique final state, the transitions are the following:

1. $\delta(p_i, a) = (p_{i-1}, X_i, +1)$, for $i = 2, \ldots, k$
2. $\delta(p_1, a) = (p_B, X_1, -1)$
3. $\delta(p_B, X_1) = (p_B, X_1, -1)$
4. $\delta(p_B, X_i) = (p_{i-1}, X_1, +1)$, for $i = 2, \ldots, k$
5. $\delta(p_i, X_1) = (p_i, X_1, +1)$, for $i = 1, \ldots, k-1$
6. $\delta(p_B, \rhd) = (p_F, \rhd, +1)$
7. $\delta(p_F, X_1) = (p_F, X_1, +1)$
8. $\delta(p_F, a) = (p_k, X_1, +1)$
9. $\delta(p_k, \lhd) = (p_k, +1)$

At the beginning of the computation, M writes in decreasing order the symbols from X_k to X_1 in the first k cells (moves 1 and 2) and then it enters a *back mode*, searching for the last written symbol different from X_1. In this mode, M ignores each cell containing X_1 (move 3). Furthermore, when M finds a symbol different from X_1, it exits from back mode, overwrites the symbol by X_1, and starts moving to the right (move 4), remembering in its state the value $i-1$, if the overwritten symbol was X_i, and skipping all cells containing X_1 (move 5), until reaching the part of the tape not yet visited where, using moves 1 and 2, it marks further $i-1$ cells.

However, if the left end-marker is reached during the back mode, then M enters another special mode, called *final mode* (move 6), where it performs the last necessary checks to verify whether or not the input word has length multiple of 2^k. First, M moves to the right of the cells already marked by X_1 to reach the first unmarked cell (move 7). Notice that when the symbol X_1 is written in a cell, then it cannot be removed, i.e., the cell is frozen. We can prove that if the head visits a cell c for the first time in a state p_i, $1 \leq i \leq k$, then (unless the computation stops before, because the input is too short) the next time the head will reach the cell $c-1$, all the $2^i - 1$ cells from position c to $c + 2^i - 2$ will contain X_1,

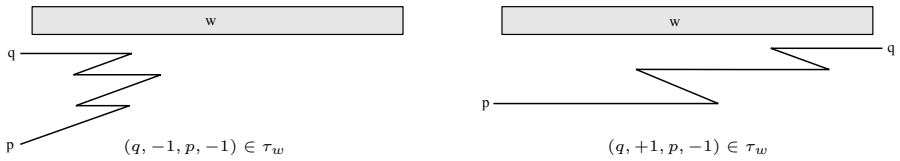

Figure 2. Some computation paths described by a transition table.

while all the cells from position $c + 2^i - 1$ to the right are not yet visited. Since the computation starts in the cell $c = 1$ and in the state p_k, it follows that when the left end-marker is reached for the first time (this can happen only in the back mode and in the state p_B), all the cells from position 1 up to position $2^k - 1$ contain X_1 and all the cells from position 2^k to the right are not yet visited. Hence, entering the final mode and moving to the right (moves 6, 7, and 8), the head will finally reach for the first time the cell $c = 2^k + 1$ in the state p_k. Hence, if the input is long enough, when the head will visit for the first time cell 2^k, all the cells up to position $2 \cdot 2^k - 1$ contain X_1 and all the cells from position $2 \cdot 2^k$ to the right are not yet visited. Hence, M continues the back mode until it reaches the left end-marker, and repeats in the same way. By iterating this argument, we conclude that the left end-marker is reached all the times the leftmost $h \cdot 2^k - 1$ cells have been rewritten by X_1, for $h \geq 1$. Hence, the cell finally reached after iterating move 7 is in position $h \cdot 2^k$. M marks it by X_1 (move 8). If it was the only symbol left to read, namely the next one is the right end-marker, then M accepts (move 9). Otherwise, being M in the initial state p_k, it can start to mark, using the same transitions, another input factor of length 2^k.

If during all the procedure the right end-marker is found in an unexpected position, then the next transition is undefined and, hence, the input is rejected.

4. Converting 2-limited automata into pushdown automata

In this section we study the conversion of 2-limited automata into pushdown automata. As a main result, we prove that from each 2-limited automaton with n states we can obtain an equivalent pushdown automaton of exponential size in n. Furthermore we show that this bound cannot be reduced.

Let us start by presenting our simulation. From now on, let M be a given 2-limited automaton.

- We consider relations from $Q \times \{-1, +1\} \times Q \times \{-1, +1\}$, called in the following *transition tables*. We associate with each $w \in \Gamma_2^*$ a transition table τ_w such that $(q, d', p, d'') \in \tau_w$ if and only if M has a computation path entering a tape segment containing w in the state q from the d'-side and exiting the segment in the state p to the d''-side, where -1 means *left* and $+1$ *right*. (See Figure 2.) Notice that to distinguish a generic transition table T from the transition table associated with a string $w \in \Gamma_2^*$, we denote the latter as τ_w.
- We can define the composition of transition tables, denoted by the symbol $\cdot$, in such a way that $\tau_{wz} = \tau_w \cdot \tau_z$, for $w, z \in \Gamma_2^*$.

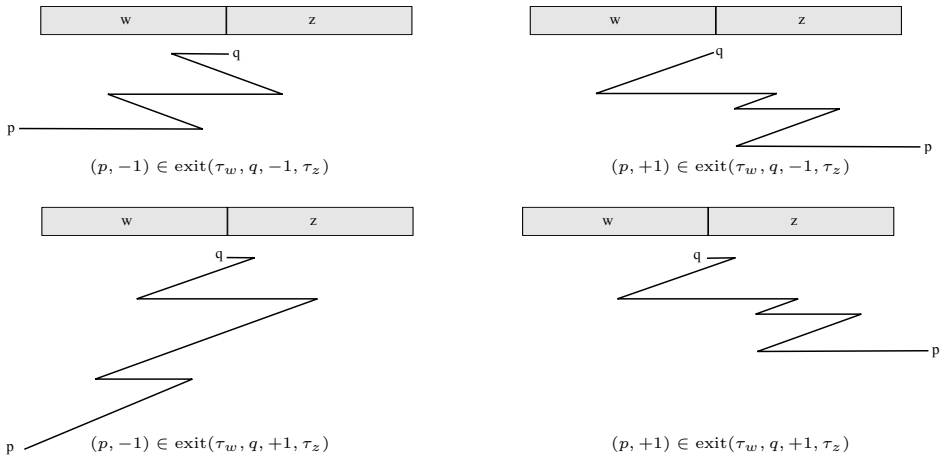

Figure 3. Possible cases for $\mathrm{exit}(\tau_w, \cdot, \cdot, \tau_z)$.

- We are also interested in computation paths starting at some point inside a tape segment. In particular, given two frozen tape segments s_l and s_r, suppose that they are adjacent, so they form a tape segment $s = s_l s_r$. Given a state q, we consider the set of pairs $(p, d) \in Q \times \{-1, +1\}$ such that M has a computation path which starts in the state q with the head entering the segment s_l from the right side and ends in the state p leaving the entire segment s from the d-side. This set of pairs depends only on the transition tables T_l and T_r associated with the segments s_l and s_r, and on the starting state q. Hence, it will be denoted as $\mathrm{exit}(T_l, q, -1, T_r)$, where the third component, -1, means that we are considering computation paths starting by entering the left segment s_l from the right. In a similar way, we denote by $\mathrm{exit}(T_l, q, +1, T_r)$ the set of pairs (p, d) such that M has a computation path which starts entering the right segment s_r from the left and finally leaves the entire segment s in the state p from the d-side. (See Figure 3).

We also use the following macros:

- $\mathrm{nSelect}(S)$: returns a nondeterministically selected element belonging to the set S. If S is empty then the computation rejects.
- $\mathrm{read}()$: returns the tape symbol on the currently scanned cell and moves the head one position to the right.

We now build a PDA M' which simulates M by implementing Algorithm 1. To simplify the presentation, first we suppose that M' receives the input string surrounded by the end-markers. Subsequently, we will explain how to modify M' in order to work without them, as in standard PDAs.

During its computation M' can work in two modes:

- In the *normal mode*, M' simulates the moves of M visiting tape cells for the first time.

Algorithm 1: The simulation

```
1  stack initially empty, head on cell 1
2  push(τ▷)
3  q ← q0                                                    // q = current state
4  repeat                                                    // main loop
5      a ← read()                                            // normal mode
6      (q, X, d) ← nSelect(δ(q, a))
7      if d = +1 then                                        // M moves to the right
8          push(X)
9      else                                                  // M moves to the left:  back mode starts
10         T ← τX                                            // T = current transition table
11         repeat
12             Y ← pop()
13             if Y is a transition relation then
14                 (q, d) ← nSelect(exit(Y, q, −1, T))
15                 T ← Y · T
16             else                                          // Y is a symbol from Γ1
17                 (q, Z, d) ← nSelect(δ(q, Y))
18                 if d = +1 then (q, d) ← nSelect(exit(τZ, q, +1, T))
19                 T ← τZ · T
20         until d = +1
21         push(T)
22 until a = ◁
23 ACCEPT
```

- In the *back mode*, M' simulates the moves of M visiting cells that have been already visited at least one time. Information about the cells already visited is kept on the stack. So, back mode mainly manipulates the pushdown store.

In particular, the back mode is entered when M', visiting a cell i for the first time, has to simulate a move of M to the left, namely a move reaching an already visited cell. However, M' cannot move its head to the left. Using the information stored on the stack, M' computes a state that will be reached by M after a computation path ending by moving from cell i to the right. At this point, the normal mode is restored and M' can examine cell $i+1$, or end the computation when the right end-marker has been inspected.

The content of the stack of M' (from the top) is a string $Y_m Y_{m-1} \cdots Y_1$, where $m \geq 0$ and each Y_j is either a transition table or a symbol from Γ_1. When M' is in the normal mode, before executing one iteration of the main loop (lines 4–22), the stack represents the situation of the tape of M to the left of the current head position i. A transition table Y_j on the stack corresponds to a tape segment whose cells have been visited at least two times and, so, are frozen; a symbol $Y_j \in \Gamma_1$ corresponds to a cell that has been visited only one time and then will be frozen in the next visit (see Figure 4). Note that, because the computation starts on cell 1 and cell 0 is frozen since the beginning, such visit should enter the cell from the right.

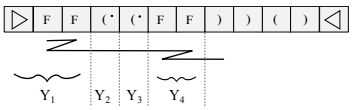

Figure 4. Tape segments corresponding to symbols on the pushdown store. The dotted symbols correspond to brackets that have already been read once.

At the beginning of the computation, the transition table $\tau_{\triangleright}$ associated with the left end-marker is pushed on the stack and M' starts to simulate M on the first input symbol, from the initial state q_0 of M. Each iteration of the main loop (lines 4–22) corresponds to one step in normal mode, inspecting a tape cell i for the first time and moving to the right (line 5). To do that, in line 6 a transition of M is nondeterministically selected and the state is updated according to it. Depending on the head movement, there are two possibilities:

- If the transition moves to the right, then M' saves on the stack the new content $X \in \Gamma_1$ of the cell i and remains in the normal mode (line 8), to continue with the next iteration of the main loop.
- If the transition moves to the left, then M' enters the back mode (lines 10–21), where it performs a series of ϵ-moves to simulate a computation path of M finally reaching cell $i+1$. (Notice that, at this point, the head of M' is already scanning cell $i+1$). In this case, the new symbol X written on the cell i should belong to Γ_2. Hence the cell i is frozen.

In back mode, M' keeps in its finite control (variable T) the transition table of the tape segment which starts from M current head position, and ends in position i, namely of the rightmost frozen tape segment. To this aim, when the back mode is entered, T is initialized with τ_X. In the case M further moves its head to the left, thus enlarging this segment, the variable T is updated. In particular, in the loop on lines 11–20, the behavior of M on cells $j \leq i$ is simulated, starting from $j = i - 1$ (which is implicitly done, since the transition selected on line 6 already moved the head of M one position to the left from position i).

If the symbol Y on the top of the stack is a transition table, then it represents a frozen tape segment from a position $h < j$ to position j. M' can join it with the rightmost frozen segment, by composing the corresponding transition tables (line 15). However, before doing that, M' has to compute how M will leave the new segment with a path starting from its current head position j. To this aim a pair from $\mathrm{exit}(Y, q, -1, T)$ is nondeterministically selected.

If Y is not a transition relation, then it must be a symbol from Γ_1, saved on the stack in a previous iteration during the normal mode (line 8). In this case, M rewrites on its tape the symbol Y by a symbol $Z \in \Gamma_2$, so the cell becomes frozen, and moves the head to direction $d \in \{-1, +1\}$ (line 17). M' updates the variable T, because the rightmost frozen segment is now enlarged (line 19). However, if while rewriting the cell the head is moved to the right, i.e., $d = +1$, then further moves are still possible before leaving the new larger segment. In this case the exit from the segment is nondeterministically computed on line 18.

When M exits the rightmost frozen segment to the right, i.e., $d = +1$ on line 20, M' leaves the back mode. Before to do that, the transition table corresponding to the rightmost frozen block, which is stored in the variable T, is saved on the stack.

According to the definition, a limited automaton can violate the right end-marker only to move in a final state. Furthermore, this is the only possibility to accept the input. Hence, the main loop correctly ends (line 22) only when a computation of M violating the right end-marker is detected. In this case M' accepts. Otherwise, the computation of M' will stop somewhere during the execution of the main loop, possibly because some of the nondeterministic choices are applied to an empty set of possibilities.[1]

Now, we explain how to modify Algorithm 1, in order to make the simulation working without end-markers, as in standard PDAs. First of all, we observe that the computation described in Algorithm 1 starts with the head on cell 1 and, hence, it does not visit the left end-marker. This allows to safely remove it from the tape (still keeping the assignment on line 2, used to simulate in back mode computation paths of M visiting the left end-marker).

For the right end-marker the argument is more involved. Reading it, M could move to the left and make a sequence of transitions possibly involving all tape cells, before violating the end-marker and finally accept. The effect of these transitions could be computed by entering the back mode and manipulating the stack. However, M' does not have the right end-marker on its tape and hence it does not know when to start this phase. This problem can be solved introducing a further nondeterministic choice. At the beginning of the main loop, M' can nondeterministically guess that all the input has been read. To this aim, line 5 is replaced by a nondeterministic choice between assigning to a the next input symbol, or the symbol $\lhd$. In the former case, the normal simulation continues, except when the end of the tape was already reached and so, due to the wrong guess, M' rejects. In the latter case, M' does not read any symbol from the tape, but simulates a move of M on the right end-marker, possibly entering back mode, if the move is to the left. In this way the main loop ends on line 22 exactly when M has a path which enters a final state violating the end-marker, otherwise the computation stops and rejects somewhere. If the end of the loop is reached, M' enters a final state and stops (line 23). In this way, if the end of the input was correctly guessed, then the computation of M' is accepting, otherwise it is rejecting, because it halted before reading the entire input.

Theorem 4.1. Each 2-LA M with n states and a working alphabet of m symbols can be simulated by a PDA M' with $2n(2^{4n^2} + 1) + 1$ states and a pushdown alphabet of $m + 2^{4n^2}$ symbols that, at each step, can push at most one symbol on the stack.

Proof:
The PDA M' implements Algorithm 1 as above explained. In the normal mode, it directly simulates the states of M, while in the back mode it keeps in its finite control, besides a state of M, a transition table. Since the number of possible tables is 2^{4n^2}, this gives $n(2^{4n^2} + 1)$ possible configurations. However,

[1] Notice that, because only one push operation is performed for each input symbol and, on the other hand, each repetition of the body of the internal loop requires a pop operation, the loop cannot be infinite.

at each iteration M' also has to remember whether or not the symbol a is the right end-marker. This doubles the number of states. By adding an accepting state, we get the claimed upper bound. Finally, we observe that the symbols of the pushdown alphabet are those from the working alphabet of M plus those representing transition tables. □

We now show that the exponential gap in Theorem 4.1 cannot be reduced, *even if the given* 2-LA *is deterministic*. To this aim, we will consider the languages K_n of Example 3.2 and use the following *interchange lemma*:

Lemma 4.2. ([10, Lemma 4.5.1])
Given a context-free language L, there exists a constant $c > 0$, depending on L, such that for every integer $n \geq 2$, every subset $R \subseteq L \cap \Sigma^n$ of strings with length n, and every integer m with $2 \leq m \leq n$, there exists a subset $Z \subseteq R$ with $Z = \{z_1, z_2, \ldots, z_k\}$ such that $k \geq \frac{\#R}{c(n+1)^2}$, and there exist decompositions $z_i = w_i x_i y_i$, with $1 \leq i \leq k$, such that the following conditions are satisfied:

1. $|w_1| = |w_2| = \cdots = |w_k|$,
2. $|y_1| = |y_2| = \cdots = |y_k|$,
3. $\frac{m}{2} < |x_1| = |x_2| = \cdots = |x_k| \leq m$,
4. $w_i x_j y_i \in L$ for all $i,\ j,\ 1 \leq i,\ j \leq k$.

Furthermore, c can be taken as the number of variables of any context-free grammar in Chomsky normal form generating L.

Theorem 4.3. For $n \geq 1$, each PDA accepting K_n has a size at least exponential in n.

Proof:
First, given a grammar $G = (V, \Sigma, P, S)$ in Chomsky normal form which generates K_n, let us consider the set $R = \{w^{n+1} \mid w \in \{0,1\}^n\} \subseteq K_n \cap \Sigma^{n(n+1)}$. Since K_n is regular, and hence context-free, by Lemma 4.2 there exists $Z \subseteq R$, with cardinality $\#Z \geq \frac{2^n}{\#V(n^2+n+1)^2}$, as in the statement of the interchange lemma. Observing the definition of R, in order to not obtain a contradiction, we can easily conclude that $\#Z \leq 1$. Therefore, $\#V \geq \frac{2^n}{(n^2+n+1)^2}$. Since each PDA can be converted into an equivalent context-free grammar in Chomsky normal form of polynomial size [9], the claimed result follows. □

We conclude the section by restricting to the deterministic case:

Theorem 4.4. For each deterministic 2-LA M accepting a language L with n states and a working alphabet of m symbols we can construct:

(i) a DPDA accepting $L\triangleleft$ with at most $n(2^{2n\cdot\log(2n+1)} + 1) + 1$ states and a pushdown alphabet of $m + 2^{2n\cdot\log(2n+1)}$ symbols,

(ii) a PDA accepting L with at most $2n(2^{2n\cdot\log(2n+1)}+1)+1$ states and a pushdown alphabet of $m+2^{2n\cdot\log(2n+1)}$ symbols,

(iii) a DPDA accepting L of size at most double exponential in the size of M.

Proof:
Again, the simulation is given by Algorithm 1. If M is deterministic, then each transition table represents a partial function from $Q\times\{-1,+1\}$ in itself. The number of such functions is $(2n+1)^{2n}=2^{2n\cdot\log(2n+1)}$. Furthermore, all the nondeterministic steps in Algorithm 1 choose from sets of at most one element, hence the algorithm is deterministic. Since, its computation starts on the tape cell 1, ignoring the left end-marker, we easily obtain from it a DPDA M' accepting $L\lhd$. As above explained, by introducing a nondeterministic choice at the beginning of each iteration of the normal mode, the automaton can guess the end of the input. The upper bounds for the cardinalities of state set and pushdown alphabet follows as in Theorem 4.1.

Finally, using the closure of deterministic context-free languages under right quotient by a regular language, we conclude that L is accepted by a DPDA M''. By inspecting the construction proving such a property, as presented for instance in [7], we obtain that the size of M'' is at most double exponential in the size of M. □

Concerning the optimality of the upper bounds given in Theorem 4.4, we observe that for (ii) and, with an easy adaption, for (i), it follows from Theorem 4.3. We also conjecture that the double exponential upper bound in (iii) cannot be reduced. This bound has been derived by using the standard construction of *predicting machines*, which allows to obtain, from a DPDA accepting a language L' and a finite automaton accepting a language R, a DPDA accepting the quotient L'/R. In our case, $L'=L\lhd$ and $R=\lhd$. Even if R consists just of one string of only one symbol, we conjecture that the cost of the construction cannot be reduced. In particular, we conjecture that each DPDA for the language K_n should have a size double exponential in n.

5. Converting pushdown automata into 2-limited automata

In [5] it was proved that each context-free language can be recognized by a 2-limited automaton. The original proof was given considering grammars in Greibach normal form. In [8], we gave a different construction based on the Chomsky-Schützenberger Theorem [2]. Both approaches do not deal with the determinism. To overcome this problem, in [5] it was also given a construction transforming *deterministic* pushdown automata (expressed as particular rewriting systems) into equivalent *deterministic* 2-limited automata. In this section we reformulate such a transformation, with the aim of also considering descriptional complexity aspects. In particular, we prove that the deterministic 2-LA resulting from the construction has a size which is polynomial with respect to the size of the given DPDA. We obtain a similar result for the nondeterministic case.

In both cases, the main idea is to simulate a PDA M by a 2-LA which uses the part of the tape to the left of the head to represent the pushdown content. To do that, we have to be sure that at each step the

height of the pushdown does not exceed the number of input symbols already inspected. This is done by converting PDAs in such a way that they can increase the stack height of at most one symbol at each step and make a restricted use of ϵ-moves. In the nondeterministic case, ϵ-moves can be completely eliminated, in the deterministic case they cannot be eliminated, but they can be used only for decreasing the stack height.

Let us start by describing the deterministic case. The construction is obtained by using a normal form for DPDAs which will be given in Lemma 5.4. To obtain it, we use of a series of constructions. In each of them, given a DPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, we describe an equivalent DPDA $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F')$. Sometimes we use pairs $(q, X) \in Q \times \Gamma$ that will be denoted in the form $[qX]$. Configurations of M will be denoted as (q, w, α), where $q \in Q$ is the state, $w \in \Sigma^*$ is the portion of the input from the head position to the right, and $\alpha \in \Gamma^*$ is the content of the pushdown store, where the leftmost symbol corresponds to the top of the stack. The symbol $\vdash^*$ denotes reachability between configurations of M.

Lemma 5.1. (Restriction of ϵ-transitions)
For every DPDA M there exists an equivalent DPDA M' such that ϵ-moves are used only to pop symbols off the stack.

Proof:
The main idea is to use a single move of M' to reproduce the situation after a sequence of ϵ-moves of M followed by one push or by one pop. More precisely, in an accepting computation of M, for each maximal sequence of ϵ-moves starting in a configuration with A on the stack top and never reading stack symbols below A, we have the following possibilities:

- the sequence is followed by one move which consumes an input symbol,
- the sequence ends by decreasing the stack height by one,
- the sequence occurs after reading all the input string and visits a final state.

Notice that, being M deterministic, we are interested in finitely many sequences of ϵ-moves which can be effectively computed from the transition table of M.

The first two possibilities can be easily simulated by single moves (see (a) and (b) below). The third one is more difficult. In fact, a DPDA is not able to detect the end of the input and, hence, it cannot directly simulate these ϵ-moves, unless that they could also be simulated even when the input is not finished, possibly making the automaton nondeterministic. This problem is solved as follows. A stack configuration $A_1A_2 \cdots A_m$ of M, with $A_i \in \Gamma$, $i = 1, \ldots, m$, is simulated in M' by the configuration $\mathrm{dup}(A_1A_2 \cdots A_m) = \bar{A}_1A_1\bar{A}_2A_2 \cdots \bar{A}_mA_m$, where $\bar{A}_i$ is a marked copy of A_i. In particular, each time a symbol A is pushed on the stack, its copy $\bar{A}$ is pushed over it. In this way, when a move of M from a state q with an A on the top stack has to be simulated, M' will find $\bar{A}$ on the top. From this configuration, M' pop $\bar{A}$ off the stack, entering one of two copies of q, q_y or q_n, depending on whether or not M has a computation path that starting from q with A on the top of the stack can reach a final state only using

ϵ-moves, without reading the stack symbols below A (see (c) below). The only accepting states are the copies q_y of original states. In this way, if q_y is entered at the end of the input, then M' accepts. From the states q_y and q_n all the transitions defined from the original q are possible. In this way, if the input is not yet finished, the simulation of M continues. Furthermore, if the next move is an ϵ-transition which pops A, then, again, from the marked symbol below A, M' can verify whether or not a final configuration is reachable only using ϵ-moves.

Formally, M' is defined by $Q' = \{q_y, q_n \mid q \in Q\}$, $\Gamma' = \{A, \bar{A} \mid A \in \Gamma\}$, $Z'_0 = Z_0$, $q'_0 = q_{0y}$ if M accepts ϵ, $q'_0 = q_{0n}$ otherwise, $F' = \{q_y \mid q \in Q\}$, and the following moves, for $p, q, r \in Q$, $A, B \in \Gamma$, $\alpha, \beta \in \Gamma^*$, $a \in \Sigma$:

(a) If $(r, \epsilon, A) \vdash^* (q, \epsilon, B\alpha)$ and $\delta(q, a, B) = (p, \beta)$ then $\delta'(r_n, a, A) = \delta'(r_y, a, A) = (p_n, \mathrm{dup}(\beta\alpha))$. (This also includes the case $r = q$, $A = B\alpha$.)

(b) If $(r, \epsilon, A) \vdash^* (q, \epsilon, B)$ and $\delta(q, \epsilon, B) = (p, \epsilon)$ then $\delta'(r_n, \epsilon, A) = \delta'(r_y, \epsilon, A) = (p_n, \epsilon)$.

(c) $\delta'(q_n, \epsilon, \bar{A}) = \begin{cases} (q_y, \epsilon) & \text{if } \exists f \in F, \gamma \in \Gamma^* \text{ s.t. } (q, \epsilon, A) \vdash^* (f, \epsilon, \gamma), \\ (q_n, \epsilon) & \text{otherwise.} \end{cases}$

□

Lemma 5.2. (Push of at most one symbol)
For each DPDA M there exists an equivalent DPDA M' such that each move can either push one symbol on the stack, or replace, leave unchanged, or pop off the stack the symbol at the top. Futhermore, if ϵ-moves can be used in M only to pop symbols off the stack, then ϵ-moves in M' cannot increase the stack height.

Proof:
In order to make the stack size increase only by one symbol for every push move, we can use special symbols to encode limited sequences of symbols. The drawback of this system is that the new automaton M' will have on the top of the stack encoded sequences instead of simple symbols, and that makes it more difficult to simulate the moves of M. However, this problem can be solved by using states of the form $[qX]$, where $q \in Q$ is the state of the starting automaton, and $X \in \Gamma$ is the symbol on the top of its stack.

The DPDA M' is defined by taking $Q' = Q \times \Gamma$ and Γ' the set of symbols $[A_1 \cdots A_k]$ encoding strings $\alpha = A_1 \cdots A_k \in \Gamma^+$ such that α is a non empty suffix of a string which can be pushed by M on the stack in a single move, $q'_0 = [q_0 Z_0]$, and the first symbol on the stack is the new symbol $Z'_0 \in \Gamma' \setminus \Gamma$. The transitions of M' are defined as follows. For each transition $\delta(q, a, X) = (p, \alpha)$ of M, with $q, p \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $X \in \Gamma$, $\alpha \in \Gamma^*$, and for each $Y \in \Gamma$, $\alpha' \in \Gamma^+$, $\xi \in \Gamma'$, with $\xi = [A_1 \cdots A_k]$, $k \geq 1$, M' has the following transitions:

$$\delta'([qX], a, \xi) = \begin{cases} ([pY], [\alpha']\xi) & \text{if } |\alpha| \geq 2 \text{ and } \alpha = Y\alpha', \\ ([pY], \xi) & \text{if } |\alpha| = 1 \text{ and } \alpha = Y, \\ ([pA_1], [A_2 \cdots A_k]) & \text{if } \alpha = \epsilon \text{ and } k \geq 2, \\ ([pA_1], \epsilon) & \text{if } \alpha = \epsilon \text{ and } k = 1. \end{cases}$$

The set final states of M' is $F' = \{[qX] \mid q \in F\}$.

Notice that the DPDA M' so obtained is in the claimed form. Furthermore, if M uses ϵ-moves only to pop symbols off the stack, then the only possible ϵ-moves of M' correspond to the last two cases in the definition of δ'. Hence, they do not increase the stack height. □

In the form obtained in Lemma 5.2, ϵ-moves are still capable of replacing the symbol at the top of the stack. To our purposes it is suitable to eliminate such capability. This is done in the next construction, which eliminates *all* replace moves.

Lemma 5.3. (Elimination of replace moves)
Given a DPDA M where ϵ-moves cannot increase the height of the stack and each other move can either push one symbol on the stack, or replace, leave unchanged, or pop off the stack the symbol at the top, there exists an equivalent DPDA M' without the ability of replacing the symbol on the top of the stack in a single move, but still satisfying the other restrictions of M.

Proof:
In order to avoid transitions replacing the top of the stack, M' simulates M by keeping at each step the top of the stack of M in its finite control. Formally, $Q' = Q \times \Gamma$, $\Gamma' = \Gamma \cup \{Z_0'\}$, where Z_0' is a new symbol, $q_0' = [q_0 Z_0]$ and $F' = \{[qA] \mid \exists p \in F, B \in \Gamma \text{ s.t. } (q, \epsilon, A) \vdash^* (p, \epsilon, B)\}$. Let us consider the following moves, for each $q, p \in Q$, $a \in \Sigma$, $X, Y \in \Gamma$, $Z \in \Gamma'$:

(a) if $\delta(q, a, X) = (p, Y)$ then $\delta'([qX], a, Z) = ([pY], Z)$,

(b) if $\delta(q, a, X) = (p, YX)$ then $\delta'([qX], a, Z) = ([pY], XZ)$,

(c) if $\delta(q, a, X) = (p, \epsilon)$ then $\delta'([qX], a, Z) = ([pZ], \epsilon)$,

(d) if the sequence $(r, \epsilon, B) \vdash^* (q, \epsilon, X)$ can be followed by one of the previous moves of M in (a)-(c), then $\delta'([rB], a, Z)$ is defined exactly as $\delta'([qX], a, Z)$. On the other hand, if $(r, \epsilon, B) \vdash^* (q, \epsilon, X)$ can be followed by the move $\delta(q, \epsilon, X) = (p, \epsilon)$, then $\delta'([rB], \epsilon, Z) = ([pZ], \epsilon)$.

□

Now, we are finally able to obtain the normal form for DPDA we are interested in:

Lemma 5.4. Every DPDA can be transformed into an equivalent DPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ which uses only moves of the following kinds:

1. $\delta(q, a, X) = (p, X)$
2. $\delta(q, a, X) = (p, YX)$
3. $\delta(q, a, X) = (p, \epsilon)$
4. $\delta(q, \epsilon, X) = (p, \epsilon)$

where each move 4 is possible only immediately after a move 3 or another move 4. Furthermore, the size of M is polynomial in the size of the given DPDA.

Proof:
Given a DPDA, by applying in the order the constructions in Lemmata 5.1, 5.2, and 5.3, we get an equivalent DPDA using only moves 1, 2, 3, and 4 (but not necessary satisfying the claimed restriction on moves 4, which will be managed later, with a further transformation). To show that the size of the obtained DPDA is polynomial with respect to the size of the given DPDA, we consider each step in the conversion.

First, we observe that the elimination of ϵ-transitions (Lemma 5.1) doubles the states, the working symbols, and the maximum number of symbols that can be pushed on the stack in one single move. The limitation of the push capability to one symbol (Lemma 5.2) uses a working alphabet where each symbol represents a sequence of symbols of the given DPDA. Since only suffixes of sequences that the original DPDA is able to push on the stack in a single move are used, the size of the new working alphabet is polynomial with respect to the size of the original one. Even the number of states is easily seen to be polynomial. Finally, the elimination of replace moves (Lemma 5.3) transforms the states in couples formed by one state and one symbol and uses just one more symbol in the working alphabet.

Now, to obtain the claimed normal form, we make a final transformation, in such a way that each move 4 can follow only moves 3 or 4.

- Suppose $\delta(q, a, X) = (p, X)$ and $\delta(p, \epsilon, X) = (r, \epsilon)$. We remove the first transition and replace it by a new transition that simulates the application of the two moves. We have to consider the fact that if the end of the input is reached after the transition reading a, then the input must be accepted if at least one of the states p and r is final. To this aim, we consider two cases. If $p \notin F$, namely acceptance is decided according to r, then the new transition is $\delta(q, a, X) = (r, \epsilon)$. Otherwise, the new transition is $\delta(q, a, X) = (r', \epsilon)$, where r' is a new state which is final and from r' the same transitions defined from r are possible.
- Suppose $\delta(q, a, X) = (p, YX)$ and $\delta(p, \epsilon, Y) = (r, \epsilon)$. Even in this case, we replace the first transition either by $\delta(q, a, X) = (r, X)$, or by $\delta(q, a, X) = (r', X)$.

This transformation can double the number of the states, but does not change the working alphabet. Observing that in the resulting DPDA each move increases the stack height of at most one, according to the discussion in Section 2 we conclude that the size of the resulting DPDA is polynomial with respect to the size of the original one. □

Now we are able to prove the following:

Theorem 5.5. For each DPDA M there exists an equivalent deterministic 2-LA whose size is polynomial with respect to the size of M.

Proof:
First of all, we can suppose that $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is in the normal form given in Lemma 5.4. Let us consider the deterministic 2-LA $M' = (Q', \Sigma, \Gamma', \delta', q'_0, F')$ with

- $Q' = \{[qX] \mid q \in Q, X \in \Gamma\} \cup \{q \mid q \in Q\}$
- $\Gamma' = \Gamma \cup \{\Box\}$
- $q'_0 = [q_0 Z_0]$
- $F' = \{[qX] \mid q \in F, X \in \Gamma\}$

and, for $p, q, r \in Q$, $a \in \Sigma$, $X, Y \in \Gamma$, the following moves:

(a) if $\delta(q, a, X) = (p, X)$ then $\delta'([qX], a) = ([pX], \Box, +1)$

(b) if $\delta(q, a, X) = (p, YX)$ then $\delta'([qX], a) = ([pY], X, +1)$

(c) if $\delta(q, a, X) = (p, \epsilon)$ then $\delta'([qX], a) = (p, \Box, -1)$

(d) if $\delta(p, \epsilon, Y) = (r, \epsilon)$ then $\delta'(p, Y) = (r, \Box, -1)$

(e) $\delta'(p, \Box) = (p, \Box, -1)$

(f) if $\delta(p, \epsilon, Y) = \emptyset$ then $\delta'(p, Y) = ([pY], \Box, +1)$

(g) $\delta'([pY], \Box) = ([pY], \Box, +1)$

Clearly, the size of M' is polynomial in that of M. The automaton M' simulates M by using the part of tape to the left of the head to represent the stack of M. The remaining part of the tape contains the portion of input still to be read by M. Furthermore, in the states of the form $[qX]$, the top of the stack is represented by X. (See moves (a) and (b), simulating moves 1 and 2 obtained from Lemma 5.4.) After a first pop (move 3), M is able to remove symbols from the stack using ϵ-moves. In order to simulate this behavior, M' enters a special *pop mode* (move (c)), where its finite control contains single states from Q. In this mode, M' moving to the left and rewriting the tape cells by a special symbol $\Box$ simulates moves 4 (move (d)). The symbol $\Box$ means that the cell does not belong to the stack of M and hence it will be skipped in the future steps (moves (e) and (g)). Once the pop sequence is finished, namely no more moves 4 are possible, M' enters a *search mode* (move (f)) and moves to the right until it finds an unread symbol (move (g)).

We notice that with those moves the automaton can rewrite each input cell in the first two visits only. In particular, when the symbol $\Box$ is written, a cell becomes frozen. In this form, the automaton M' does not use the end-markers. When the head reaches the first cell to the right of the input string, it could accept or reject. (This special model, considered in [5], is called J_2-automaton.) With a trivial modification, we can manage an end-marked tape, thus obtaining a 2-LA. □

Now, we switch to the nondeterministic case. Given a PDA M we can obtain an equivalent 2-LA M' according to the following steps:

1. The PDA M is transformed into an equivalent context-free grammar G.

2. The grammar G is converted into an equivalent grammar G_2 in 2-*Greibach normal form*, namely a grammar where the right-hand side of every production starts with a terminal, possibly followed by at most 2 variables.

3. From G_2, an equivalent PDA M_2 is obtained. This PDA does not use ϵ-moves and at each step can increase the stack height of at most one.

4. Finally, the PDA M_2 is transformed into an equivalent 2-LA M' that, hence, is equivalent to the original PDA M.

Evaluating the costs of the above listed steps, we obtain the following:

Theorem 5.6. For each PDA M there exists an equivalent 2-LA whose size is polynomial with respect to the size of M.

Proof:
Steps 1 and 3 use standard transformations which are polynomial in size (as described, e.g., in [7]). The conversion to 2-Greibach normal form, used in step 2, has been proved to be polynomial in [1]. Finally, step 4 is simple adaption of the conversion for the deterministic case presented in Theorem 5.5. □

6. Final observations

The model presented in the original paper of Hibbard [5] was slightly different: the tape is infinite to the right and the limited automaton can use an unbounded number of tape cells to the right of the input string. These cells initially contain a blank symbol. Moreover, rewriting on them is restricted to the same constraints as on input cells. The machine accepts if there is a computation which halts in a final state, while reaching a blank cell for the first time. The simulation in Algorithm 1 can be changed as follows:

- Line 2 is removed.
- Once M' reaches the right end of the input (which can be guessed, as we explained), it does not further move its head. However, it continues the simulation by assigning to the variable a the blank symbol (line 5).
- The main loop (lines 4–22) ends when an halting configuration is reached, with M in the normal mode. If the state is accepting then M' accepts, otherwise it rejects.

Besides d-limited automata, Hibbard also considered a weaker model, called J_d-*automata* (we already mentioned J_2-automata in the proof of Theorem 5.5). He proved that the class of deterministic context-free languages coincides with the class of languages accepted by J_2-automata and it is included in the class of languages accepted by deterministic 2-limited automata, leaving as an open problem to characterize the computational power of deterministic 2-LAs. As a consequence of our Theorem 4.4, this problem is now solved:

Theorem 6.1. The class of languages accepted by deterministic 2-LAs coincides with the class of deterministic context-free languages.

On the other hand, for $d > 2$ there are deterministic d-LAs which are able to recognize context-free languages which are nondeterministic, as for instance, the language in Example 3.3. In [5] an infinite hierarchy of deterministic languages was presented: for each $d \geq 2$ there is a language that can be recognized by a deterministic $(d+1)$-limited automaton, such that each equivalent d-limited automaton must be *nondeterministic*.

References

[1] Blum, N., Koch, R.: Greibach Normal Form Transformation Revisited, *Information and Computation*, **150**(1), 1999, 112–118, ISSN 0890-5401.

[2] Chomsky, N., Schützenberger, M.: The Algebraic Theory of Context-Free Languages, in: *Computer Programming and Formal Systems* (P. Braffort, D. Hirschberg, Eds.), vol. 35 of *Studies in Logic and the Foundations of Mathematics*, Elsevier, 1963, 118–161.

[3] Goldstine, J., Kappes, M., Kintala, C. M. R., Leung, H., Malcher, A., Wotschke, D.: Descriptional Complexity of Machines with Limited Resources, *J. UCS*, **8**(2), 2002, 193–234.

[4] Harrison, M. A.: *Introduction to Formal Language Theory*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978, ISBN 0201029553.

[5] Hibbard, T. N.: A Generalization of Context-Free Determinism, *Information and Control*, **11**(1/2), 1967, 196–238.

[6] Holzer, M., Kutrib, M.: Descriptional Complexity — An Introductory Survey, in: *Scientific Applications of Language Methods*, chapter 1, Imperial College Press, 2010, 1–58.

[7] Hopcroft, J. E., Ullman, J. D.: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[8] Pighizzini, G., Pisoni, A.: Limited Automata and Regular Languages, in: *Descriptional Complexity of Formal Systems* (H. Jürgensen, R. Reis, Eds.), vol. 8031 of *Lecture Notes in Computer Science*, Springer, 2013, 253–264, An extended version will appear on the *International Journal of Foundations of Computer Science*.

[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), 2002, 393–414.

[10] Shallit, J. O.: *A Second Course in Formal Languages and Automata Theory*, Cambridge University Press, 2008.

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