Fundamenta Informaticae XXI (2001) 1001–1024
DOI 10.3233/FI-2012-0000
IOS Press

Strongly Limited Automata

Giovanni Pighizzini*
Dipartimento di Informatica
Università degli Studi di Milano, Italy
pighizzini@di.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. When $d \geq 2$, these devices characterize the class of context-free languages. In this paper we consider restricted versions of these models which we call *strongly limited automata*, where rewrites, head reversals, and state changes are allowed only at certain points of the computation. Those restrictions are inspired by a simple algorithm for accepting Dyck languages on 2-limited automata. We prove that the models so defined are still able to recognize all context-free languages. We also consider descriptional complexity aspects. We prove that there are polynomial transformations of context-free grammars and pushdown automata into strongly limited automata and vice versa.

1. Introduction

Almost half a century ago, Hibbard discovered a characterization of context-free languages which uses a restricted version of Turing machines, called *limited automata* [7]. For each integer $d \geq 0$, a d-limited automaton is a one-tape 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, 1-limited automata characterize regular languages [18].

Those results have been recently revisited, also investigating descriptional complexity aspects in [15, 14] where, as a preliminary example, a simple 2-limited automaton accepting a Dyck language was presented. It can be easily observed that the algorithm implemented by such an automaton actually does not

Address for correspondence: Dip. di Informatica, Università degli Studi di Milano, via Comelico 39, 20135 Milano, Italy
*Partially supported by MIUR under the project PRIN "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi", code H41J12000190001.

need to rewrite each tape cell two times, so it does not fully use the capabilities of 2-limited automata. On the other hand, due to the well-known Chomsky-Schützenberger representation of context-free languages [1], recognition of context-free languages can be, in some sense, reduced to the recognition of Dyck languages. These observations suggested to us that it might be interesting to investigate if it is further possible to restrict the moves of 2-limited automata without reducing the computational power and, in particular, if a device which closely imitates the way used by 2-limited automata to recognize Dyck languages can accept all context-free languages. This leads us to define *strongly limited automata*, that are machines satisfying the following restrictions:

- While moving to the right, a strongly limited automaton always uses the same state q_0 until the content of a cell (which has not been yet rewritten) is modified. Then it starts to move to the left.
- While moving to the left, the automaton rewrites each cell it meets that is not yet rewritten up to some position where it starts again to move to the right. Furthermore, while moving to the left the automaton does not change its internal state.
- In the final phase of the computation, the automaton inspects all tape cells, to check whether or not the final content belongs to a given 2-strictly locally testable language. Roughly, this means that all the factors of two letters of the string which is finally written on the tape should belong to a given set.

We prove that in spite of those limitations all context-free languages can be accepted by these devices. Hence, *strongly limited automata* characterize the class of context-free languages. To prove that each context-free language is accepted, we use a non-erasing variant of the Chomsky–Schützenberger representation, obtained by Okhotin [13]. We also observe that the description of the resulting machine has polynomial size with respect to the grammar which specifies the language. The converse inclusion derives considering the computational power of 2-limited automata. However, in the paper we directly provide a conversion from strongly limited automata to pushdown automata, which produces a machine having a description of size polynomial with respect to the size of the description of the given strongly limited automaton. In contrast, the conversion of 2-limited automata into pushdown automata requires exponential size [15].

In 1996 by Jancar, Mráz, and Plátek [10] considered a different restriction of limited automata, called *forgetting automata*. These machines can modify tape cells only by erasing their contents, using a unique fixed symbol. They proved that forgetting automata which are forced to erase the content of tape cells while moving to the left characterize the class of context-free languages. It should be clear that those models represent restrictions of 2-limited automata. (For more recent investigations about variants of forgetting automata we refer the reader to [3, 4].) In the last part of the paper we discuss the differences between strongly limited automata, forgetting automata, and other variants of devices which work by rewriting a tape that, at the beginning of the computation, contains the input.

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, by w^R its *reversal*, namely the string obtained by taking the symbols of w in reverse order, and by ϵ the empty string.

We assume the reader to be familiar with standard notions from formal language and automata theory, in particular with the notions of *finite automaton*, *pushdown automaton*, *context-free grammar* (see, e.g., [9, 17]).

The notion of *local language* will be widely used in the paper. In order to recall it, first we remind the reader that a regular language L is *strictly locally testable* [12] if there is an integer k such that membership in L can be "locally" verified by inspecting all factors of length k in the input string. In the case $k = 2$ we simply say that the language is *local*. More precisely, given an alphabet Σ and two extra symbols $\rhd, \lhd \notin \Sigma$, we say that a language $L \subseteq \Sigma^*$ is *local* if and only if there exists a set $F \subseteq (\Sigma \cup \{\rhd, \lhd\})^2$ of *forbidden factors* such that a string $x \in \Sigma^*$ belongs to L if and only if no factor of length 2 of $\rhd x \lhd$ belongs to F.

Given the set F it is easy to define a finite automaton A_F which recognizes L. While scanning a cell containing a symbol a, A_F just remembers the symbol b in the previous cell and moves to a rejecting state in the case ba is a forbidden factor. Otherwise, it moves the head on the next cell remembering the symbol a and continues in the same way. Hence, A_F can be obtained using a different state q_a for each $a \in \Sigma \cup \{\rhd, \lhd\}$. Notice that A_F could contain equivalent states. Hence, after minimization we obtain an automaton that enters a same state q_a each time it reads the symbol a (unless a forbidden factor has been already discovered). However, we could have $q_a = q_b$ for different symbols a and b. We can make a completely similar discussion if we are interested to recognize a local language from right to left, as we will do in the paper. We can also observe that the local language L can be defined in terms of *allowed* factors, namely, the set of 2-letter factors that do not belong to the set F.

We are interested in comparing the description sizes of devices and formal systems. (For surveys in the area of descriptional complexity see, e.g., [5, 8, 11].) We can measure the size of a context-free grammar by counting the total number of symbols which are necessary to write down its productions. In a similar way, in the case of pushdown automata or other kinds of machines, we can consider the total number of symbols specifying the transition table. For a detailed discussion we address the reader to [6].

3. Dyck Language Recognition

The machine model we are discussing in the paper was inspired by a very simple algorithm for the recognition of the language of balanced sequences of brackets. The algorithm, already presented as an example in [15, 14], uses rewriting operations in a very restricted way. Hence, before introducing the model, in this section we are going to illustrate it.

For each integer $k \geq 1$, we denote by Ω_k the alphabet of k types of brackets, which will be represented as $\{ (_1,)_1, (_2,)_2, \ldots, (_k,)_k \}$. The *Dyck language* D_k over the alphabet Ω_k is the set of strings representing well balanced sequences of brackets.

The *Dyck language* D_k can be recognized by a machine M which starts having the input string on its tape, surrounded by two end-markers $\rhd$ and $\lhd$, with the head on the first input symbol. From this configuration, M moves to the right to find a closed bracket $)_i$, $1 \leq i \leq k$. Then M replaces $)_i$ with a symbol $\mathsf{X} \notin \Omega_k$ and changes the head direction, moving to the left. In a similar way, it stops when during this scan it meets for the first time a left bracket $(_j$. If $i \neq j$, i.e., the two brackets are not of the same

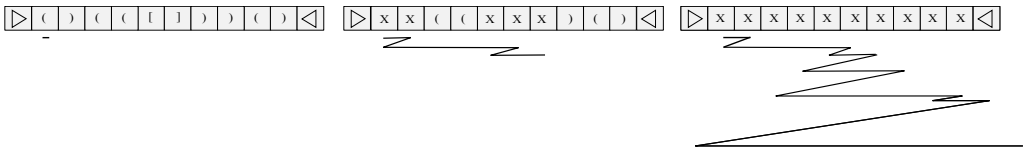

Figure 1. Some steps in an accepting computation of the automaton M of Algorithm 1 on input ( ) (([ ])) ( ).

Algorithm 1: Recognition of the Dyck language D_k

```
1  start with the head on the first input symbol
2  while symbol under the head ≠ ◁ do
3  |  move the head to the right
4  |  if symbol under the head = )_i (with 1 ≤ i ≤ k) then
5  |  |  write X
6  |  |  repeat
7  |  |  |  move the head to the left
8  |  |  until symbol under the head ≠ X
9  |  |  if symbol under the head ≠ (_i then REJECT
10 |  |  write X
11 repeat
12 |  move the head to the left
13 until symbol under the head ≠ X
14 if symbol under the head = ▷ then ACCEPT
15 else REJECT
```

type, then M rejects. Otherwise, M writes X on the cell and changes again the head direction moving to the right. This procedure is repeated until M reaches 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 does not have a matching left bracket. Hence, M rejects.
- If instead the right end-marker is reached, then M 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 M rejects. On the other hand, if M reaches the left end-marker reading only Xs, then it can accept the input.

Now we look more into the details to the implementation of this procedure, which is summarized in Algorithm 1.

The machine M starts the computation in the initial state q_0 (line 1). While moving to the right to search a closed bracket (line 3), M does not need to keep in its finite control any other information, so it can always use the same state q_0. On the other hand, after a closed bracket $)_i$ is found, $1 \leq i \leq k$, M needs to remember the index i in order to find a matching open bracket $(_i$. This is done by using a state q_i for moving to the left while performing the search and the next rewriting on lines 6–10. The final loop and test (lines 11–15) can be performed using a further state $q_{\triangleleft}$. Notice that each cell containing a closed bracket is rewritten in the first visit, while each cell containing an open bracket is rewritten in the second visit.

We now consider a generalization of Dyck languages that will be used in the paper and we show how to modify the machine M to recognize it. The generalization is obtained by introducing extra symbols [13]. More precisely, for each integer ℓ we can add ℓ new symbols to Ω_k, called *neutral* symbols. We denote the alphabet so obtained by $\Omega_{k,\ell}$. The *extended Dyck language* $\widehat{D}_{k,\ell}$ over $\Omega_{k,\ell}$ is the set of all strings $w \in \Omega_{k,\ell}^*$ which can be obtained by padding strings in D_k with neutral symbols. It should be clear that $\widehat{D}_{k,\ell}$ is a context-free language.

We can easily modify the machine M in order to accept the extended Dyck language $\widehat{D}_{k,\ell}$. The new machine $\widehat{M}$ uses the same set of states as M. The only difference in the algorithm is when the head of $\widehat{M}$ reaches a cell containing a neutral symbol. $\widehat{M}$ could just ignore the symbol, moving to the next cell along the same direction, without changing the state. However, for our purposes, it is useful to rewrite it, using X, if the cell is reached while moving to the left (line 7). With this procedure, only neutral symbols at the outer level, namely which are not enclosed in brackets, are not rewritten. Hence, at the end of the final scan, $\widehat{M}$ accepts if and only if the string finally written on the tape between the end-markers is a string formed only by Xs and neutral symbols.

By summarizing, the state set of $\widehat{M}$ consists of the following parts:

- An initial state q_0 which is the *only state* used while moving to the right.
- A set of states $Q_L = \{q_1, \ldots, q_k\}$ which are used to move to the left.
- A set of states Q_Υ which are used to move to the left during the final complete scan of the input.

Furthermore, we observe that $\widehat{M}$ cannot change the head direction when it does not rewrite the current cell (except on the right end-marker). In the state q_0, moving to the right, $\widehat{M}$ can rewrite just one cell and then it has to turn its head to the left. On the other hand, in a state $q \in Q_L$, moving to the left, $\widehat{M}$ rewrites all the cells not yet rewritten that are visited, until it changes the head direction and re-enters state q_0. Up to this moment, $\widehat{M}$ does not change its internal state.

A pair of symbols in a string w belonging to an (extended) Dyck language D is said to be a *matching pair* if it is formed by an open and a closed bracket of the same type which match, namely the factor of w surrounded by them belongs to w. For instance, the first and the last symbols in (( ) (( ))) form a matching pair, but not in ( ) (( )).

4. Strongly Limited Automata

We are now ready to introduce the model we are studying in this paper, which behaves in a way very similar to the machine $\widehat{M}$ described at the end of Section 3.

A *strongly limited automaton* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_\triangleright)$, where:

- Q is a finite *set of states*, which is partitioned in the three disjoint sets $\{q_0\}$, Q_L, and Q_Υ.
- Σ and Γ are two finite and disjoint sets of symbols, called respectively the *input alphabet* and the *working alphabet* of $\mathcal{M}$. Let us denote by Υ the *global alphabet* of $\mathcal{M}$ defined as $\Upsilon = \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$, where $\triangleright, \triangleleft \notin \Sigma \cup \Gamma$ are two special symbols called, respectively, the *left* and the *right end-marker*.

- $\delta : Q \times \Upsilon \to 2^{\{\leftarrow\!\!-, -\!\!\rightarrow, \overset{\mathsf{X}}{\leftarrow}, {}_q\overset{\mathsf{X}}{\leftarrow}, \overset{\mathsf{X}}{\rightarrow}_q, q\not\longleftarrow \mid \mathsf{X} \in \Gamma, q \in Q\}}$ is the *transition function*, which associates a set of possible *operations* with each configuration of $\mathcal{M}$.
- q_0 is the *initial state*.
- $q_\triangleright \in Q_\Upsilon$ is the *final state*.

We now describe how $\mathcal{M}$ works, providing an informal explanation of the meaning of the states and of the operations that $\mathcal{M}$ can perform. First of all, we assume that at the beginning of the computation the tape contains the input string $w \in \Sigma^*$ surrounded by the two end-markers. Tape cells are counted from 0. Hence, cell 0 contains $\triangleright$ and cell $|w|+1$ contains $\triangleleft$. The head is on cell 1, namely scanning the leftmost symbol of w, while the finite control is in q_0.

The initial state q_0 is the only state which is used while moving from left to right. In this state all the cells that have been already rewritten are ignored, just moving one position further, while on all the other cells $\mathcal{M}$ could be allowed either to move to the right or to rewrite the cell content and then turn the head direction to the left, entering a state in the set Q_L. To this aim, in the state q_0 the following operations could be possible:

- *Move to the right* $-\!\!\rightarrow$
  Move the head one position to the right *without* rewriting the cell content and *without* changing the state.
- *Turn to the left* ${}_q\overset{\mathsf{X}}{\leftarrow}$
  Write $\mathsf{X} \in \Gamma$ in the currently scanned tape cell, move one position to the left, entering in state $q \in Q_L$. After a sequence of moves from left to right, with this operation $\mathcal{M}$ rewrites the content of the current cell and changes the head direction, entering a state $q \in Q_L$.

We point out that these two operations are not allowed in states other than q_0. One further operation (${}_{q_\triangleleft}\not\longleftarrow$, described later) is possible in q_0, when the right end-marker is reached, to activate the final phase of the computation.

The states in the set Q_L are used to move to the left. In a state $q \in Q_L$, the automaton $\mathcal{M}$ ignores all the cells that have been already rewritten, just moving to the left. On the remaining cells that $\mathcal{M}$ visits, it always rewrites the content up to some position where it turns its head to the right. During this procedure, $\mathcal{M}$ changes state only at the end, when it enters again in q_0. In a state $q \in Q_L$ the following operations can be allowed:

- *Move to the left* $\leftarrow\!\!-$
  Move the head one position to the left *without* rewriting the cell content and *without* changing the state. This move is used only on cells that have been rewritten.
- *Write and move to the left* $\overset{\mathsf{X}}{\leftarrow}$
  Write $\mathsf{X} \in \Gamma$ in the currently scanned tape cell, move one position to the left, *without* changing the state. This move can be used only on cells not yet rewritten.
- *Turn to the right* $\overset{\mathsf{X}}{\rightarrow}_{q_0}$
  Write $\mathsf{X} \in \Gamma$ in the currently scanned tape cell, move one position to the right, entering in the state q_0. Even this move can be used only on cells not yet rewritten.

If the left end-marker is reached while scanning to the left in a state of Q_L then the computation stops by rejecting (technically the next transition is undefined). On the other hand, if the right end-marker is reached while scanning to the right in q_0, the machine starts a final phase where it completely scans the tape from right to left and then stops. During the last phase $\mathcal{M}$ checks the membership of the final tape content to a local language. If some forbidden factor is detected then the next transition is undefined and hence $\mathcal{M}$ rejects. To this aim, in this phase only states from the set Q_Υ are used. We assume that there is a surjective map from Υ to Q_Υ. We simply denote as q_{X} the state associated with the symbol $\mathsf{X} \in \Upsilon$. Note that $\mathsf{X} \neq \mathsf{Y}$ does not implies $q_{\mathsf{X}} \neq q_{\mathsf{Y}}$. The following operation is used in this phase:

- *Check to the left* ${}_{q_a}\!\longleftarrow\!\!\shortmid$
  On a cell containing symbol $a \in \Upsilon$, move to the left remembering the state associated with a.

If no forbidden factor is found, $\mathcal{M}$ finally violates the left end-marker in the state $q_\triangleright$. In this case the input is accepted. Otherwise, the computation of $\mathcal{M}$ stopped in some previous step and the input is rejected. Hence, we assume that $\mathcal{M}$ *accepts its input if and only if from the cell containing the left end-marker it can further move to the left entering the final state* $q_\triangleright$.

Formally, the transition function δ has to satisfy the conditions listed below.

- For the state q_0:
  - $\delta(q_0, a) = \{\text{-}\text{-}\!\rightarrow\}$ if $a \in \Gamma$,
  - $\delta(q_0, a) \subseteq \{\text{-}\text{-}\!\rightarrow\} \cup \{{}_q\!\overset{\mathsf{X}}{\longleftrightarrow}\mid q \in Q_L, \mathsf{X} \in \Gamma\}$ if $a \in \Sigma$,
  - $\delta(q_0, \lhd) = \{{}_{q_\lhd}\!\longleftarrow\!\!\shortmid\}$,
  - $\delta(q_0, \rhd)$ is undefined.
- For each state $q \in Q_L$:
  - $\delta(q, a) = \{\!\leftarrow\!\text{-}\text{-}\}$ if $a \in \Gamma$,
  - $\delta(q, a) \subseteq \{\overset{\mathsf{X}}{\longleftarrow}, \overset{\mathsf{X}}{\hookrightarrow}_{q_0} \mid \mathsf{X} \in \Gamma\}$ if $a \in \Sigma$,
  - $\delta(q, a)$ is undefined if $a \in \{\rhd, \lhd\}$.
- For each state $q_{\mathsf{X}} \in Q_\Upsilon$:
  - $\delta(q_{\mathsf{X}}, a) \subseteq \{{}_{q_a}\!\longleftarrow\!\!\shortmid\}$, where $a \in \Upsilon$.

Notice that, according to those restrictions, the description of a strongly limited automaton can be written using a number of symbols which is polynomial in the number of the states and in the cardinality of the global alphabet.

Example 4.1. Consider the alphabet Ω_2, with brackets represented by the symbols `(`, `)`, `[`, `]`. The Dyck language D_2 is accepted by a strongly limited automaton with $\Gamma = \{\mathsf{X}\}$, $Q_L = \{q_1, q_2\}$, $Q_\Upsilon = \{q_{\mathsf{X}}, q_\rhd, q_\lhd\}$, and the following transitions (we omit braces for the sake of the brevity):

- $\delta(q_0, \texttt{(}) = \delta(q_0, \texttt{[}) = \text{-}\text{-}\!\rightarrow$, $\delta(q_0, \texttt{)}) = {}_{q_1}\!\overset{\mathsf{X}}{\longleftrightarrow}$, $\delta(q_0, \texttt{]}) = {}_{q_2}\!\overset{\mathsf{X}}{\longleftrightarrow}$,
- $\delta(q_1, \mathsf{X}) = \delta(q_2, \mathsf{X}) = \!\leftarrow\!\text{-}\text{-}$, $\delta(q_1, \texttt{(}) = \delta(q_2, \texttt{[}) = \overset{\mathsf{X}}{\hookrightarrow}_{q_0}$,

- $\delta(q_0, \lhd) = {}_{q_\lhd}\!\!\longleftarrow\!\!\shortmid$, $\delta(q_\lhd, \mathsf{X}) = \delta(q_\mathsf{X}, \mathsf{X}) = {}_{q_\mathsf{X}}\!\!\longleftarrow\!\!\shortmid$, $\delta(q_\mathsf{X}, \rhd) = {}_{q_\rhd}\!\!\longleftarrow\!\!\shortmid$.

It can be observed that the states $q_\lhd, q_\mathsf{X}, q_\rhd$ used for the final scan can be merged in a unique state $q_\rhd$. In fact, in this example the purpose of the final scan is to check that all the input symbols have been rewritten, namely, no symbol $a \in \{\,(\,,\,)\,,\,[\,,\,]\,\}$ is left on the tape. If such a symbol is discovered, then the next transition is not defined and hence the computation rejects.

To accept the *extended Dyck language*, with a neutral symbol $|$, we can add the moves $\delta(q_0, |) = \dashrightarrow$ and $\delta(q_1, |) = \delta(q_2, |) = \overset{\mathsf{X}}{\longleftarrow}$.

In this way, on input w in the language, when the right end-marker is reached all brackets and neutral symbols enclosed in matching brackets are rewritten by X. For example, $w = |\,|\,[\,|\,|\,(\,|\,)\,]\,|\,(\,|\,|\,)\,[\,]\,|$ is transformed into $x = |\,|\,\mathsf{XXXXXXX}\,|\,\mathsf{XXXXXX}\,|$. Hence, in the last scan the automaton has to verify that no brackets have been left. Again, only the state $q_\rhd$ with transitions defined only on symbols $\lhd, \mathsf{X}, |$, and $\rhd$ is enough to this aim. □

Notice that in the previous example the final tape content is a sequence of strings $z_1 z_2 \cdots z_k$, $k \geq 0$, where for $i = 1, \ldots, k$, either z_i is a factor of the input which has not been rewritten, namely, $z_i \in \Sigma^+$, or $z_i \in \Gamma^+$ is a factor which corresponds to a block of matching open and closed brackets (with possible extra neutral symbols inside) which have been rewritten along a "zig-zag" trajectory starting from the innermost closed bracket. For instance, the above displayed final content x can be decomposed as $z_1 z_2 z_3 z_4 z_5 z_6$, where $z_1 = |\,|$, z_2 corresponds to the input block $[\,|\,|\,(\,|\,)\,]$, $z_3 = |$, z_4 corresponds to $(\,|\,|\,)$, z_5 corresponds to $[\,]$, and $z_6 = |$.

A similar decomposition of the tape content can be found while recognizing any other language, when the head of a strongly limited automaton reaches the right end-marker, namely immediately before the final scan. Rewritten blocks z_i's represent sequences of symbols that have been matched during the inspection of the tape. This observation will be helpful for the transformation of strongly limited automata into pushdown automata presented in Section 6.

Example 4.2. The set PAL of palindromes over the alphabet $\Sigma = \{a, b\}$ can be recognized by a strongly limited automaton M with a three letter working alphabet $\Gamma = \{\mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$ and the set $Q_L = \{p_a, p_b\}$. Using a nondeterministic strategy, M starts to inspect the input from the center to check the matching between symbols in the first and in the second half at the same distance from the center. The symbols in the first half of the input are rewritten by Y, those in the second half by X. When the input has an odd length (also this information is guessed by M), its central symbol is rewritten by Z. Hence the final tape content enclosed between the end-markers should belong to $\mathsf{Y}^*(\mathsf{Z} + \epsilon)\mathsf{X}^* + a + b$ (the last two terms in the expression correspond to inputs of length 1 which are accepted without any rewriting).

Formally, M has the following transitions:

- $\delta(q_0, a) = \{\dashrightarrow, {}_{p_a}\!\overset{\mathsf{X}}{\longleftarrow}\}$, $\delta(q_0, b) = \{\dashrightarrow, {}_{p_b}\!\overset{\mathsf{X}}{\longleftarrow}\}$
  Scanning the tape from left to right, reading a cell containing an input symbol the automaton either continues to move to the right, or rewrites the symbol (so guessing that it is the leftmost symbol not yet inspected in the second half of the input) and moves to the left to check if the rightmost symbol left in the first half is matching.

- $\delta(q_0, \gamma) = \{\dashrightarrow\}$ and $\delta(p_a, \gamma) = \delta(p_b, \gamma) = \{\dashleftarrow\}$ for $\gamma \in \Gamma$
  Rewritten cells are ignored.

- $\delta(p_a, a) = \{\xleftarrow{\mathsf{Z}}, \xrightarrow[q_0]{\mathsf{Y}}\}$, $\delta(p_b, b) = \{\xleftarrow{\mathsf{Z}}, \xrightarrow[q_0]{\mathsf{Y}}\}$, $\delta(p_a, b) = \delta(p_b, a) = \{\xleftarrow{\mathsf{Z}}\}$
  Reading an input symbol while moving to the left, M either guesses that it is the central symbol, so rewriting it by Z and further moving to the left, or verifies the matching with the last input symbol inspected in the second half, so rewriting it by Y and reversing the head direction, to start to search again, in the state q_0, the leftmost symbol not yet rewritten in the second half or the right end-marker.

- The transitions in the final phase are defined in such a way to allow exactly the following 2-letter factors: $\triangleright a$, $\triangleright b$, $\triangleright\mathsf{Y}$, YY, YZ, ZX, XX, $\mathsf{X}\triangleleft$, $b\triangleleft$, $a\triangleleft$, $\triangleright\triangleleft$.

In Section 8 we will show that the language PAL cannot be accepted by any strongly limited automaton that uses a working alphabet with fewer than 3 symbols. □

Example 4.3. The deterministic context-free language $L = \{a^n b^{2n} \mid n \geq 0\}$ is accepted by a strongly limited automaton which guesses each second b. While moving from left to right and reading b, the automaton makes a nondeterministic choice between further moving to the right or rewriting the cell by X and turning to the left. Furthermore, while moving to the left, the content of each cell containg b which is visited is rewritten by Y, still moving to the left, and when a cell containing a is visited, its content is replaced by Z, turning to the right. In the final scan the machine accepts if and only if the string inscripted between end-markers is of the form $\mathsf{Z}^*(\mathsf{YX})^*$. In Section 7 we will show that nondeterministic choices are essential to recognize this language L using strongly limited automata. □

Example 4.4. The deterministic context-free language $L = \{ca^n b^n \mid n \geq 0\} \cup \{da^{2n} b^n \mid n \geq 0\}$ is accepted by a strongly limited automaton in the following way. Each time a cell containing a letter b is reached, it is rewritten by X, turning to the left in a state nondeterministically chosen between q_1 and q_2. In q_1, when an a is reached it is rewritten by Z, turning to the right, while in q_2 when an a is reached it is rewritten either by Y, further moving to the left, or by Z, turning to the right. The machine accepts if and only if the final tape content between the end-markers belongs to $c\mathsf{Z}^*\mathsf{X}^* + d(\mathsf{ZY})^*\mathsf{X}^*$. Even for this language L nondeterministic choices are essential, as we will show in Section 7. □

5. Context-Free Language Recognition

In this section we show that each context-free language is accepted by a strongly limited automaton. Since strongly limited automata are a subclass of 2-limited automata which, in turn, characterize context-free languages [7], we then immediately conclude that strongly limited automata characterize context-free languages too. Our result is founded on the well-known theorem of Chomsky-Schützenberger, which states that each context-free language can be represented in the form $h(D_k \cap R)$ where h is an homomorphism, D_k is the *Dyck language* over an alphabet with k types of brackets, and R is a regular language [1]. Recently Okhotin [13] proved some non-erasing variants of this result. The first one states that for each language which does not contain any string of length 1 it is always possible to consider a *non-erasing homomorphism* h. Using this variant, a construction of 2-limited automata from context-free languages has been done in [14]. In that construction, tape cells are rewritten in the first two visits. Here, we provide a different construction, which produces a strongly limited automaton. Our new construction is based on a further variant of the Chomsky-Schützenberger Theorem, which makes use of extended Dyck languages and is restricted to letter-to-letter homomorphisms [13, Thm. 3]:

Theorem 5.1. A language $L \subseteq \Sigma^*$ is context-free if and only if there exist numbers $k, \ell \geq 1$, a regular language $R \subseteq \Omega_{k,\ell}^*$ and a letter-to-letter homomorphism $h : \Omega_{k,\ell} \to \Sigma$ such that $L = h(\widehat{D}_{k,\ell} \cap R)$.

Given a context-free language L, let k, ℓ, R, and h be as in Theorem 5.1. We now describe a machine $\mathcal{M}$ accepting L. In a first version, $\mathcal{M}$ uses a two track tape. On the first track $\mathcal{M}$ keeps the input w, while on the second track it writes a nondeterministically guessed string $x \in \widehat{D}_{k,\ell}$. The string x is generated in the first phase of the computation by simulating another machine $\widehat{M}_g$ discussed later. After that, $\mathcal{M}$ scans the tape in order to verify whether or not $x \in R$ and $w = h(x)$.

Actually, $\mathcal{M}$ does not need to keep a two track tape. When $\widehat{M}_g$ generates the symbol $x_j \in D_{k,\ell}$, $j = 1, \ldots, |w|$, $\mathcal{M}$ immediately verifies whether or not $w_j = h(x_j)$, where w_j and x_j denote the symbols in position j in strings w and x, respectively. If the outcome is positive, then the computation continues by replacing w_j by x_j on the tape, otherwise the computation stops by rejecting. Hence, in a final phase, it remains only to check whether or not $x \in R$.

Before further proceeding, we explain how the machine $\widehat{M}_g$ works. As a preliminary step, we modify the machine M that recognizes D_k (see Section 3), to obtain a machine M_g which, given a finite tape consisting of n cells delimited by two end-markers, writes on it a nondeterministically chosen string $x \in D_k$ of length n (if any). Let us suppose that all the n cells initially contain a blank symbol. The construction can be easily adapted to the case of cells which initially contain symbols from an alphabet which is disjoint from Ω_k. The machine M_g uses the same set of states of the machine M accepting D_k. In the initial state q_0 the machine moves the head to the right. On a blank cell, M_g can continue to move to the right or it can write a closed bracket $)_i$, where i is nondeterministically chosen. In the last case, M_g enters the state q_i and starts to move to the left, up to the point where it reaches a blank cell. Here, M_g writes the open bracket $(_i$ and then it re-enters the state q_0 and starts to move to the right repeating the same procedure. If while moving to the left to search a blank position where to write an open bracket the left end-marker is reached, then the generation is unsuccessful and the computation is aborted, exactly as in line 9 of Algorithm 1. On the other hand, when the right end-marker is reached, a scan of the input from right to left is performed, to verify that all the cells have been filled (this final phase corresponds to lines 14 and 15 of Algorithm 1). We notice that the only nondeterministic decisions are taken in the state q_0. Furthermore, each cell which finally contains a closed bracket is written only in the first visit, while each cell which finally contains an open bracket is written in the second visit.

We can modify M_g in order to obtain the automaton $\widehat{M}_g$ generating the extended Dyck language $\widehat{D}_{k,\ell}$ as follows. While moving to the right, namely in the state q_0, the automaton $\widehat{M}_g$ behaves exactly as M_g. However, moving to the left in a state q_i, when reaching a blank cell the automaton $\widehat{M}_g$ nondeterministically chooses between one of the following two actions: rewrite the content of the cell by $(_i$ and then enter q_0 and move to the right, or rewrite the content of the cell by a neutral symbol and continue to move to the left, remaining in the state q_i. Furthermore, in the final phase, while moving to the left in the state $q_\triangleleft$, each blank cell is rewritten using a neutral symbol.

Going back to the above outlined machine $\mathcal{M}$ for the language L, we observe that in the generation of a string $x \in \widehat{D}_{k,\ell}$, $\mathcal{M}$ does not need to simulate the final scan of $\widehat{M}_g$. In fact, while performing its final scan to check if $x \in R$, each time the head of $\mathcal{M}$ reaches a not rewritten symbol w_j, it can guess a neutral symbol $x_j \in h^{-1}(w_j)$.[1]

[1]Actually, as stated in the next Theorem 5.2, all strings of length at least 2 in the language R start and end with a matching pair of brackets. According to the above explanation, matching pairs of brackets are generated from right to left, rewriting with

By looking at the structure of the machine $\mathcal{M}$ so obtained, we can observe that in order to conclude that it is a strongly limited automaton it remains to prove that the regular language R is local, which allows $\mathcal{M}$ to perform the final scan of the tape as explained in Section 4. We do that by closely inspecting the construction provided in [13] to prove Theorem 5.1. This analysis allows us to rewrite the *only-if* part of that theorem by the following statement and also to prove that $\mathcal{M}$ has a description of size polynomial with respect to the size of a context-free grammar for L.

Theorem 5.2. There exists a polynomial p such that for each context-free grammar G of size s generating a language $L \subseteq \Sigma^*$ there exist numbers k, ℓ, with $1 \leq k, \ell \leq p(s)$, a local language $R \subseteq \Omega_{k,\ell}^*$, and a letter-to-letter homomorphism $h : \Omega_{k,\ell} \to \Sigma$ such that $L = h(\widehat{D}_{k,\ell} \cap R)$. Furthermore, for each $x \in \widehat{D}_{k,\ell} \cap R$ either its first and last symbols form a matching pair or x has length at most 1.

Proof:
We summarize the crucial steps of the construction presented in [13].

- First of all, each context-free grammar generating a language $L \subseteq \Sigma^+$ can be converted into *double Greibach normal form* [16], where the productions have the forms $A \to bC_1 \cdots C_k d$ and $A \to a$, with $a, b, d \in \Sigma$, $k \geq 0$, and $A, C_1, \ldots, C_k$ are variables.

Details are given for the case $L \subseteq (\Sigma^2)^*$, along the following lines [13, Thm. 1]:

- A grammar G in double Greibach normal form, without rules of the form $A \to a$, generating $L \backslash \{\epsilon\}$ is considered. Let P denotes the set of G productions.
- Brackets of the form ${}^{\Xi}_{A \to bC_1 \cdots C_k d}($ and ${})^{\Xi}_{A \to bC_1 \cdots C_k d}$ are used, where $A \to bC_1 \cdots C_k d$ is a production of the grammar (representing the *current production*) and Ξ is either a production (the *previous production*) or $-$ (in the case the current production is applied at the outer level of the derivation tree). Hence, the cardinality of the bracket alphabet Ω_G so defined is quadratic in the number of the productions, namely polynomial in the size of the grammar G.
- Let D_G be the Dyck language over Ω_G.
- A regular language R_G is defined using "local" conditions. These conditions define which factors of length 2 are allowed in a string and which symbols are allowed at the first and at the last position. A string $x \in \Omega_G^*$ belongs to the language R_G if and only if x satisfies those conditions. Hence, the language R_G is local. In particular, the following 2-letter factors are allowed in x:
  - for each production $A \to bC_1 \cdots C_k d$, $k \geq 1$, and $\Xi \in P \cup \{-\}$:

$${}^{\Xi}_{A \to bC_1 \cdots C_k d}({}^{A \to bC_1 \cdots C_k d}_{C_1 \to \gamma_1}, \text{ for } C_1 \to \gamma_1 \in P,$$

$${})^{A \to bC_1 \cdots C_k d}_{C_i \to \gamma_i}({}^{A \to bC_1 \cdots C_k d}_{C_{i+1} \to \gamma_{i+1}}, \text{ for } C_i \to \gamma_i, C_{i+1} \to \gamma_{i+1} \in P, i = 1, \ldots, k-1,$$

$${})^{A \to bC_1 \cdots C_k d}_{C_k \to \gamma_k}){}^{\Xi}_{A \to bC_1 \cdots C_k d} \text{ for } C_k \to \gamma_k \in P,$$

a neutral symbol each blank position in between. Hence we can conclude that all strings in $\widehat{D}_{k,\ell} \cap R$ are generated by $\widehat{M}_g$ without using the final scan to fill blank positions.

– for each production $A \to bd$ and $\Xi \in P \cup \{-\}$:

$$({}^{\Xi}_{A\to bd}){}^{\Xi}_{A\to bd}.$$

Furthermore, the only symbols allowed at the beginning and at the end of x have the form ${}^{-}_{S\to\sigma}($ and $){}^{-}_{S\to\sigma}$, respectively, where $S \to \sigma \in P$.

We point out that according to this last condition, a string $x \in R_G$ could start by ${}^{-}_{S\to\sigma}($ and end by $){}^{-}_{S\to\sigma'}$, with $\sigma \neq \sigma'$. However, for x which also belongs to the Dyck language D_G, $\sigma \neq \sigma'$ would imply the existence of a factor $){}^{-}_{S\to\sigma}({}^{\Xi'}_{\Xi''}$, for some Ξ', Ξ'', which is not allowed according to previous rules. Hence, $\sigma = \sigma'$ for all strings $x \in D_G \cap R_G$. Furthermore, the first and the last symbols in x form a matching pair.

- By considering the homomorphism $h : \Omega_G \to \Sigma$:

$$h\left({}^{\Xi}_{A\to bC_1\cdots C_k d}(\right) = b \qquad h\left(){}^{\Xi}_{A\to bC_1\cdots C_k d}\right) = d$$

it is proven that $L = h(D_G \cap R_G)$.

In the general case, namely $L \subseteq \Sigma^*$, the construction can be extended as follows.

- A grammar G in double Greibach normal form (without any restriction) generating $L \setminus \{\epsilon\}$ is considered.
- The alphabet Ω_G is obtained by defining brackets as before plus neutral symbols of the form ${}^{\Xi}_{A\to a}|$, where $A \to a$ is a production of G generating a single terminal and Ξ is either a production or $-$. The homomorphism h maps the symbol ${}^{\Xi}_{A\to a}|$ into the symbol a. Brackets are mapped as before.
- Let $\widehat{D}_G$ be the extended Dyck language over Ω_G.
- Even in this case, the regular language R_G is defined in terms of local conditions, which can also involve neutral symbols. More precisely, the set of 2-letter factors which are allowed is extended, with respect to the previously given one, by adding:

– for each production $A \to bC_1 \cdots C_k d$, $k \geq 1$, and $\Xi \in P \cup \{-\}$:

$$({}^{\Xi}_{A\to bC_1\cdots C_k d}|^{A\to bC_1\cdots C_k d}_{C_1\to c}, \text{ for } C_1 \to c \in P,$$

$$|^{A\to bC_1\cdots C_k d}_{C_i\to c}({}^{A\to bC_1\cdots C_k d}_{C_{i+1}\to\gamma_{i+1}}, \text{ for } C_i \to c, C_{i+1} \to \gamma_{i+1} \in P,\ i = 1,\ldots,k-1,$$

$$|^{A\to bC_1\cdots C_k d}_{C_i\to c}|^{A\to bC_1\cdots C_k d}_{C_{i+1}\to c'}, \text{ for } C_i \to c, C_{i+1} \to c' \in P,\ i = 1,\ldots,k-1,$$

$$)^{A\to bC_1\cdots C_k d}_{C_i\to\gamma_i}|^{A\to bC_1\cdots C_k d}_{C_{i+1}\to c'}, \text{ for } C_i \to \gamma_i, C_{i+1} \to c' \in P,\ i = 1,\ldots,k-1,$$

$$|^{A\to bC_1\cdots C_k d}_{C_k\to c})^{\Xi}_{A\to bC_1\cdots C_k d} \text{ for } C_k \to c \in P.$$

The sets of symbols that can appear at the beginning and at the end of strings in R_G are extended by adding neutral symbols of the form $|^-_{S\to A}$. Observing the list of allowed 2-symbol factors, we can conclude that a neutral symbol can appear at the beginning or at the end of a string $x \in R_G$ only if $|x| = 1$. Furthermore, if $|x| \geq 2$ and $x \in \widehat{D}_G \cap R_G$ then the first and the last symbol of x form a matching pair.

Finally, the language R_G is defined to contain the empty string if and only if $\epsilon \in L$.

- With this modification, it can be proved that $L = h(\widehat{D}_G \cap R_G)$.

To obtain the polynomial bound claimed in the statement of the theorem, we observe that the cardinality of the alphabet Ω_G obtained in the previous steps is quadratic in the number of the productions of the grammar G that was given in double Greibach normal form. By analyzing the transformations presented in [2, 19], it can be observed that each context-free grammar can be converted into this form with a polynomial increasing of the size of the description. □

Considering the above presented construction of the machine $\mathcal{M}$ and using Theorem 5.2 we have obtained the following simulation result:

Theorem 5.3. Each context-free language L is accepted by a strongly limited automaton $\mathcal{M}$. Furthermore, $\mathcal{M}$ can be obtained with a description of size polynomial with respect to the size of any given context-free grammar generating L or of any given pushdown automaton accepting L.

Concerning the size of the machine $\mathcal{M}$ in Theorem 5.3, an inspection to its construction (see the discussion at the beginning of this section) allows to observe that its states and its working alphabet symbols essentially correspond to symbols of the alphabet $\Omega_{k,\ell}$ used to represent the language L in the form $L = h(\widehat{D}_{k,\ell} \cap R)$, as in Theorem 5.2. This allows to conclude that the size of the description of $\mathcal{M}$ is quadratic with respect to the parameters k and ℓ, namely, it is quadratic with respect to the polynomial p in Theorem 5.2 which, in turn, is quadratic with respect to the number of the productions of the grammar G, in double Greibach normal form, which specifies the language L.

However, the current known upper bound for the conversion of context-free grammars into double Greibach, even if polynomial, is very high. In fact, as a consequence of a result in [19], each grammar in Chomsky normal form with v variables and r productions can be converted into an equivalent grammar in double Greibach normal form with $O(v^6 r^4)$ many productions. Furthermore, it is well-known that the conversion into Chomsky normal form can square the size of a grammar. In summary, an estimation of the upper bounds in Theorems 5.2 and 5.3 produces polynomial with high degrees.

6. Simulation by Pushdown Automata

Limited automata can recognize only context-free languages when the number of rewriting in each cell is bounded by a constant d [7]. From a descriptional point of view, in [15] it has been proved that pushdown automata can have a size exponentially larger than equivalent 2-limited automata, even when the accepted language is regular. Here, we prove that starting from strongly limited automata we always obtain pushdown automata of polynomial size. Hence, strongly limited automata may need descriptions exponentially larger than the descriptions of equivalent 2-limited automata. In other words, the simplicity of the operations of strongly limited automata is paid with exponentially larger descriptions.

Throughout the section, let us consider a strongly limited automaton $\mathcal{M}$ accepting a language L and defined according to the notations given in Section 4. We are going to describe an equivalent pushdown automaton $\mathcal{A}$, which implements Algorithm 2.

The algorithm uses the following strategy. Each cell c of the tape can be rewritten by $\mathcal{M}$ either in the first visit, when moving to the right in the state q_0 $\mathcal{M}$ turns to the left, or in the second visit, while moving to the left in a state of Q_L. Since the pushdown automaton $\mathcal{A}$ can only move to the right, the moves of the latter type are simulated in two steps: when the head of $\mathcal{A}$ reaches the cell c, simulating a move of $\mathcal{M}$ which visits for the first time c and does not rewrite its content, $\mathcal{A}$ saves on the pushdown store the input symbol written in c, to be inspected later when, in a *back mode*, the second visit will be simulated. Due to the restrictions on the state changes, $\mathcal{A}$ does not need to simulate the parts of computation which visit cells that have already been rewritten. With this approach, the difficult point is the simulation of the last scan of the tape from right to left, where the final content has to inspected. In fact, during the simulation, the symbols written by $\mathcal{M}$ are not kept. Furthermore, as emphasized in Example 4.1, cells are not rewritten in the same order as they appear in the tape. In order to overcome this problem, when a cell c is reached, simulating a move of $\mathcal{M}$ which visits it for the first time and moves to the right, $\mathcal{A}$ guesses the symbol that will be written during the second visit and saves it on the pushdown store together with the current symbol, to verify the guess later, while simulating in the back mode the second visit to c.

In this way, while scanning the input from left to right, the pushdown automaton $\mathcal{A}$ can simulate (in reverse order) the last scan of the tape, in parallel with the other moves of $\mathcal{M}$.

We now discuss the algorithm more into details. It makes use of the following macros or abbreviations:

- $\mathsf{nSelect}(S)$: nondeterministically selects one element from the set S.
- $\mathsf{symb}(\mathsf{op})$: returns the symbol X written on the tape by operation $\mathsf{op} \in \{{}_q\overset{\mathsf{X}}{\longleftarrow}, \overset{\mathsf{X}}{\longleftarrow}, \overset{\mathsf{X}}{\longrightarrow}_{q_0}\}$.
- $\mathsf{state}(\mathsf{op})$: returns the state q reached by the operation $\mathsf{op} = {}_q\overset{\mathsf{X}}{\longleftarrow}$.

The pushdown automaton $\mathcal{A}$ works in two modes:

- In the *direct mode*, $\mathcal{A}$ directly simulates the transitions of $\mathcal{M}$ moving to the right. We remind the reader that during these moves $\mathcal{M}$ remains in the initial state q_0. Each iteration of the main loop (lines 3–21) corresponds to one move of $\mathcal{M}$ which visits *for the first time* one tape cell c. In this phase $\mathcal{M}$ has two possibilities: either to move to the right, without changing the content of c, or turn to the left, after rewriting c.

  In the former case (lines 7–9), the content of c will be changed when $\mathcal{M}$ will return on the cell for the second time, moving to the left. $\mathcal{A}$ guesses the symbol X that will be written on the cell in the second visit and saves it on the pushdown store together with the current symbol a, for a future verification (line 18). It is also possible that $\mathcal{M}$ will return on that cell for the second time only in the last scan of the tape, namely, without changing the content of the cell. For this reason, in line 7, the current tape symbol may also be guessed. (The corresponding verification will be done on line 26.)

  In latter case, $\mathcal{A}$ enters the back mode (described below), after saving the symbol X written on the cell (line 11) and the state q (line 12) which will be used in that mode.

Algorithm 2: The pushdown automaton $\mathcal{A}$

```
1  stack initially empty, head on cell 1
2  X_prec ← ▷                                                    // last symbol
3  while end of the tape not yet reached do
4      a ← read()                                                // simulation of state q_0
5      op ← nSelect(δ(q_0, a))                                   // op ∈ {-->, <-X-_q}
6      if op = --> then
7          X ← nSelect(Γ ∪ {a})
8          push(X)
9          push(a)
10     else                                                      // op =_q <-X-
11         X ← symb(op)                                          // saved to be used on lines 20-21
12         q ← state(op)                                         // current state, used while scanning to the left
13         repeat
14             b ← pop()
15             Y ← pop()
16             op ← nSelect(δ(q, b))                             // op ∈ {<-Z-, <-Z-_{q_0}}
17             Z ← symb(op)
18             if Y ≠ Z then REJECT
19         until op = <-Z-_{q_0}                                 // iterate when op =<-Z-
20     if δ(q_X, X_prec) = ∅ then REJECT
21     X_prec ← X
22 if δ(q_◁, X_prec) = ∅ then REJECT
23 while stack not empty do
24     a ← pop()
25     X ← pop()
26     if a ≠ X then REJECT
27 ACCEPT
```

During the direct mode, $\mathcal{A}$ also simulates (in reverse order) the last scan of the input, from the right to the left end-marker (lines 20–21). In this part of the computation, that will be discussed later, the symbol X saved on line 7 or on line 11 is used.

- In the *back mode*, $\mathcal{A}$ verifies that the information previously guessed and saved on the pushdown matches with a sequence of moves of $\mathcal{M}$ to the left. First of all, $\mathcal{A}$ remembers the state q used by $\mathcal{M}$ moving to the left. Then, for each cell c visited by $\mathcal{A}$ while moving to the left, $\mathcal{A}$ works as follows (lines 14–18).

  If the cell c has been visited only one time (i.e., moving from left to right), then $\mathcal{A}$ gets its original content b from the stack, together with a symbol $\mathsf{Y} \in \Gamma$, previously guessed visiting c in normal mode (line 7). Then $\mathcal{A}$ selects a transition of $\mathcal{M}$ from state q with input b and verifies that the symbol written in the cell by such a transition is Y, rejecting when the test fails (line 18). If the selected transition moves to the left, then $\mathcal{A}$ continues to operate in the back mode, otherwise it resumes the direct mode. We remind the reader that when $\mathcal{M}$ visits a cell which has already been rewritten, it just skips it, moving along the same direction and without changing its state (except in the final scan). For this reason, no information is kept for those cells by $\mathcal{A}$ and in the back mode

they are not considered at all. For the same reason, to resume the direct mode $\mathcal{A}$ does not need to simulate any transition from the head position of $\mathcal{M}$ to the current head position of $\mathcal{A}$ (which represents the first cell not yet visited by $\mathcal{M}$) because all those transitions cannot change the tape content and the internal state q_0 of $\mathcal{M}$.

When the head reaches the right end-marker, $\mathcal{M}$ makes a final scan from right to left, in order to verify each 2-letter factor of the string on the tape (including the end-markers). If some forbidden factor is found then the next transition is undefined and the computation stops. The same verification can be easily performed from left to right, simulating the same set of states. This is done by $\mathcal{A}$ on lines 20–21. At each iteration of the main loop (lines 3–21), $\mathcal{A}$ reads a symbol a from its input tape and guesses or directly computes the symbol X that replaces a on the tape (lines 7 and 11, respectively). On line 20 $\mathcal{A}$ verifies that the symbol is compatible with the symbol $\mathsf{X}_{\mathsf{prec}}$ written by $\mathcal{M}$ in the cell immediately to the left. To do that, it is enough to verify that the transition $\delta(q_{\mathsf{X}}, \mathsf{X}_{\mathsf{prec}})$ used by $\mathcal{M}$ to make the same verification (moving from the right) is defined. To complete the simulation of the last scan, when exiting the main loop, $\mathcal{A}$ has to verify the existence of the initial transition of the last scan, from the state $q_{\triangleleft}$ on the rightmost tape symbol. This is done on line 22. At this point, the stack should contain exactly information corresponding to tape cells that have not been rewritten by $\mathcal{M}$. For those cells two copies of the original input symbol have been pushed on the stack (cf. lines 6–9). Hence $\mathcal{A}$ makes a final check of the stack before accepting.

From Algorithm 2, we observe that the pushdown automaton $\mathcal{A}$ keeps in its finite state control the variables $a, b \in \Sigma$, $\mathsf{X}_{\mathsf{prec}}, \mathsf{X}, \mathsf{Y}, \mathsf{Z} \in \Gamma \cup \{\triangleright\}$, $q \in Q$, and op which ranges over the set of possible operations (not including those which are used in the final scan). Actually, a and b are not used at the same time, so only one variable ranging on Σ is enough for both of them. Furthermore, the value of Z derives from that of op. Since the number of possible operations is $O(sw)$, it turns out that the number of possibile states of $\mathcal{A}$ is $O(is^2w^4)$, where $s = \#Q$, $w = \#\Gamma$, and $i = \#\Sigma$. The cardinality of the pushdown alphabet is $i + w$. Since each stack operation can push or pop at most one symbol, this allows to conclude that the size of the description of $\mathcal{A}$ is polynomial with respect to the one of the given strongly limited automaton $\mathcal{M}$.

Theorem 6.1. Each strongly limited automaton can be simulated by a pushdown automaton of polynomial size.

Example 6.2. Let $\mathcal{M}$ be the strongly limited automaton described in Example 4.1, which accepts the extended Dyck language. In Figure 2 some snapshots during an accepting computation of $\mathcal{M}$ on input $w =$ | ((|) | ( )) are depicted. Corresponding snapshots in a simulating computation of the pushdown automaton $\mathcal{A}$ described in Algorithm 2 are represented in Figure 3. Now, we compare the two computations.

(a)→(b) From the initial configuration, moving to the right, $\mathcal{M}$ repeatedly executes the operation $\dashrightarrow$ up to the first closed bracket. This phase is simulated by $\mathcal{A}$ (lines 6–9) storing the guessed symbols on the stack (left column) together with the symbols that are read from the tape (right column).

(b)→(c) $\mathcal{M}$ executes the operation ${}_{q_1}\!\overset{\mathsf{X}}{\dashleftarrow\!\!\dashrightarrow}$, by rewriting the cell and turning to the left. This operation is directly simulated by $\mathcal{A}$ just updating the information in its finite control (lines 11–12) and entering the back mode.

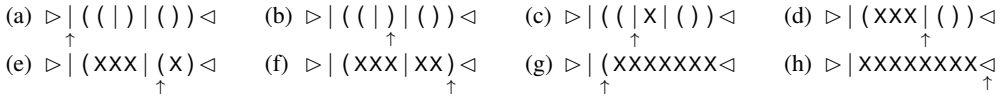

Figure 2. Some snapshots of the tape of the strongly limited automaton $\mathcal{M}$, with the head position, during the accepting computation described in Example 6.2.

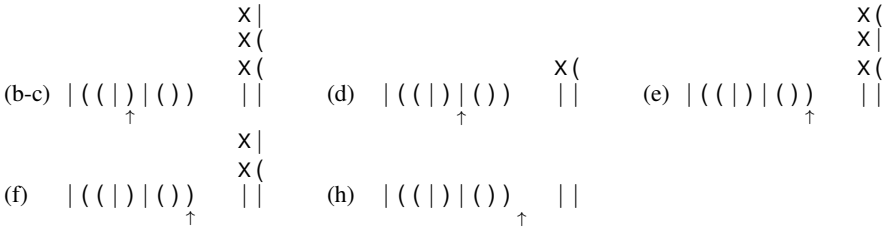

Figure 3. Some snapshots of the pushdown automaton $\mathcal{A}$ during the accepting computation described in Example 6.2. The labels refer to the corresponding configurations of $\mathcal{M}$ in Figure 2. In each snapshot, the tape content with the head position are represented on the left, while the pushdown content is represented on the right. Since each time $\mathcal{A}$ increases its stack, a guessed symbol X is pushed together with an input symbol a (lines 8–9 of Algorithm 2), and, furthermore, each time the stack is decreased in size, two symbols are removed, the stack is represented in a compact way by using two columns. The left column contains guessed symbols, the right column contains copies of input symbols.

(c)→(d) $\mathcal{M}$ continues to move the head to the left and to rewrite, up to a cell containing an open bracket, which is also rewritten. Here, $\mathcal{M}$ turns to the right, executing $\xleftarrow{\mathsf{X}}_{q_0}$, and continues to move to the right, executing $\dashrightarrow$, up to a not yet visited cell. In this phase, each move of $\mathcal{M}$ to the left is simulated by $\mathcal{A}$ checking if it is compatible with the two symbols on the top of the stack (loops on lines 13–19), which are removed.

(d)→(e) Further moving to the right, the head of $\mathcal{M}$ reaches a cell containing the next closed bracket. The cell is rewritten and the head is moved to the left. Even in this case, $\mathcal{A}$ in direct mode pushes symbols on the stack, reaching the configuration in Figure 3(e).

(e)→(f) After rewriting the cell containing the open bracket, and turning to the right, with one further move to the right, the head of $\mathcal{M}$ reaches the last input cell, which contains a closed bracket. Notice that at this point (f), three cells in the part of the tape to the left of the head are not yet rewritten. Those cells are represented on the stack of $\mathcal{A}$.

(f)→(g)→(h) With similar sequences of moves, $\mathcal{M}$ rewrites the cell, turn its head to the left, and moves up to the matching open bracket (and also rewriting the neutral symbol which is visited during these moves). Then the head is moved to the right finally reaching the right end-marker. The pushdown automaton $\mathcal{A}$ simulates the first step by moving the head to the right of the input. The remaining steps are simulated in the back mode, without moving the input head and checking the information on the stack. At the end, the stack contains the information corresponding to the only symbol that does not have been rewritten by $\mathcal{M}$.

At this point, $\mathcal{M}$ has to execute the final phase, inspecting the tape from right to left. This part has been simulated by $\mathcal{A}$ during the previous steps, while moving from left to right. □

7. Determinism vs Nondeterminism

In Section 5 and 6 we proved that strongly limited automata characterize the class of context-free languages. It is quite natural to ask what is the computational power of the deterministic version of these devices and, in particular, if they are able to capture the class of deterministic context-free languages as deterministic 2-limited automata [15].

The answer is negative. On the one hand, we can give examples of deterministic context-free languages which are not accepted by deterministic strongly limited automata. On the other hand, our simulation of strongly limited automata by pushdown automata presented in Section 6 does not preserve determinism. In particular, in line 7 of Algorithm 2, nondeterministic choices are introduced to guess the final content of the tape. This is useful to simulate the final scan of the tape, in parallel with the other phases of the computation. Hence, with our simulation, it is also not clear if each language accepted by a deterministic strongly limited automaton is a deterministic context-free language. This problem is left open for future investigations.

Let us examine some of these aspects more into details. First of all, due to the restrictions on the use of the states while moving to the right, there are simple examples of deterministic context-free languages which require nondeterministic choices to be accepted by strongly limited automata. We give a proof of this fact for the languages presented in Examples 4.3 and 4.4 which are clearly deterministic context-free.

Theorem 7.1. Each strongly limited automaton accepting the language $L = \{a^n b^{2n} \mid n \geq 0\}$ is nondeterministic.

Proof:
By contradiction, let M be a deterministic strongly limited automaton accepting L.

First of all, we notice that $\delta(q_0, a) = _r\overset{\mathsf{Y}}{\leftarrow\!\!\rightarrow}$, for some r, Y, should imply that each input starting with the letter a is rejected, because in the first transition the head would immediately reach the left end-marker. Hence, $\delta(q_0, a) = \dashrightarrow$. Furthermore, if $\delta(q_0, b) = \dashrightarrow$ then M will scan each input from $\{a, b\}^*$ from left to right, reaching the right end-marker without any rewriting. So in the last scan of the tape, M should decide whether or not the input string (read in reverse way) belongs to L. Since L is not regular, this cannot be done. Hence $\delta(q_0, b) = _p\overset{\mathsf{X}}{\leftarrow\!\!\rightarrow}$, for some p, X.

Now, let us consider an input $x = a^n b^{2n} \in L$. Each time the head of M reaches a cell containing a letter b, the cell is rewritten and the head is moved to the left. The only way to reach another b, and finally the right end-marker to start the final scan, is to reverse the head direction when, moving from right to left, a cell containing the symbol a is found. Since the number of a's in x is less than the number of b's, at some step during this computation the left end-marker will be reached, so rejecting x. □

For the language of Example 4.4 we make use of the following result:

Lemma 7.2. Let Σ be an alphabet containing at least three different symbols a, b, d. Consider $x = da^4b^2$, $y = da^5b^2$, and $L \subseteq \Sigma^*$. If $x \in L$ and $y \notin L$ then each strongly limited automaton accepting L is nondeterministic.

Proof:
The argument is similar to the one used to prove Theorem 7.1. Suppose we have a deterministic strongly limited automaton M accepting L. First, we observe that $\delta(q_0, a) = \delta(q_0, d) = \dashrightarrow$, otherwise on each input starting with daa (and hence on $x = da^4b^2 \in L$) the head will hit the left end-marker rejecting.

If also $\delta(q_0, b) = \dashrightarrow$, then M on both inputs x and y will scan the entire tape from left to right without any rewriting, so reaching the right end-marker with $\triangleright daaaabb\triangleleft$ and $\triangleright daaaaabb\triangleleft$ on the tape, respectively. Since the sets of 2-letter factors in these strings coincide, this implies that either both x and y will be accepted in the final scan, or both of them will be rejected, which is a contradiction. Hence, $\delta(q_0, b) = {}_p\overset{\mathsf{X}}{\longleftarrow}$, for some p, X.

We also notice that $\delta(p, a) = \overset{\mathsf{Y}}{\longleftrightarrow}_{q_0}$, for some Y, otherwise x should be rejected because the head will finally hit the left end-marker, in a sequence of moves to the left. (In the case $\delta(p, d) = \overset{\mathsf{Z}}{\longleftarrow}$, this happens in the sequence of moves to the left which starts after rewriting the first b; otherwise $\delta(p, d) = \overset{\mathsf{Z}}{\longleftrightarrow}_{q_0}$ and this happens in the sequence of moves to the left which starts after rewriting the second b).

According to these moves, we can conclude that on x and y, when the head of M will reach the right end-marker the tape contents will be $\triangleright daa\mathsf{YYXX}\triangleleft$ and $\triangleright daaa\mathsf{YYXX}\triangleleft$, respectively. Again, since these two strings contain exactly the same 2-letter factors, this would implies that both x and y are accepted or both of them are rejected, which is a contradiction to the assumption that M recognizes L. □

As a consequence of Lemma 7.2, we immediately get the following result:

Theorem 7.3. Each strongly limited automaton accepting the language $L = \{ca^nb^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$ is nondeterministic.

We could slightly relax the definition of strongly limited automata, by introducing a set of states Q_R, with $q_0 \in Q_R$, used while moving to the right, and allowing transition between states of Q_L and of Q_R while moving to the left and to the right, respectively, but still forbidding state changes on rewritten cells and by keeping all the other restrictions. Let us call the model so obtained an *almost strongly limited automata*. We observe that the simulation by pushdown automata presented in Algorithm 2 can be adapted for almost strongly limited automata. So the recognition power of these devices is the same as strongly limited automata, namely they characterize context-free languages.

Concerning the relationships between determinism and nondetermism, as already pointed out, the simulation by pushdown automata does not preserve determinism. On the other hand, the deterministic context-free languages considered in Theorems 7.1 and 7.3 are accepted in a deterministic way by almost strongly limited automata. In particular:

- To recognize $L = \{a^nb^{2n} \mid n \geq 0\}$, a machine can move from left to right to locate the second b on the tape, rewrite it and move back to rewrite the first b and the last a on the tape. This can be repeated until one end-marker is reached. In more detail, the head moving from left to right scans the a's and the symbols already rewritten remaining in the state q_0. When a cell containing b is reached, the machine still moves to the right entering a state q_1. From this state, if another b is found, the machine rewrites it by X, enters a state p and starts to move to the left. In p, each b is rewritten by X and each rewritten symbol is ignored, still moving to the left, until a cell containing an a is found. This cell is rewritten by Z, reentering q_0 and turning to the right, to repeat the same procedure, up to reach the right end-marker. The input is accepted if and only if the final content of the tape inscripted between end-markers is a string of the form $\mathsf{Z}^*\mathsf{X}^*$.

- To recognize $L = \{ca^n b^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$ it is enough to remember in the finite state control the leftmost input symbol, after the first step. This information is used to decide how many a's need to be rewritten for each b. In particular if the first symbol is a d, a transition in the state Q_L is used while moving to the left, in order to count two letters a.

It would be interesting to know if almost strongly limited automata are able to accept all deterministic context-free languages without taking nondeterministic decisions.

8. Comparisons with Other Models

In the introduction, we mentioned that the class of context-free languages has been characterized in 1996 by Jancar, Mráz, and Plátek in terms of a kind of *forgetting automata* [10].

We briefly mention that forgetting automata can erase tape cells by rewriting their contents with a special symbol. However, rewritten cells are kept on the tape and are still considered during the computation. For instance, the state can be changed while visiting an erased cell. In the variant of forgetting automata that characterizes context-free languages, when a cell which contains an input symbol is visited while moving to the left, its content is rewritten, while no changes are done while moving to the right.

This way of operating is very close to that of strongly limited automata. However, in strongly limited automata the rewriting alphabet can contain more than one symbol. Furthermore, rewritten cells are completely ignored (namely, the head direction and the state cannot be changed while visiting them) except in the final scan of the tape from the right to the left end-marker.

To emphasize the difference between strongly limited automata and forgetting automata, we now prove that each strongly limited automaton accepting the set PAL of palindromes needs a working alphabet of at least 3 symbols (cf. Example 4.2).

The proof is by contradiction. Suppose M is a strongly limited automaton with working alphabet $\Gamma = \{\mathsf{X}, \mathsf{Y}\}$ accepting PAL.

First of all, we observe that $\dashrightarrow \in \delta(q_0, a)$. Otherwise, each input starting with a should be rejected because from the initial configuration either the only possible move turns the head to the left (if $p \xleftarrow{\mathsf{Z}} \in \delta(q_0, a)$, for a state p, symbol $\mathsf{Z} \in \Gamma$) reaching the left end-marker, or the machine stops (if $\delta(q_0, a) = \emptyset$). In a similar way, $\dashrightarrow \in \delta(q_0, b)$.

Lemma 8.1. Any accepting computation of M on inputs aba and bab will rewrite all input symbols.

Proof:
Without loss of generality, we give the proof only for $x = aba$. First, we observe that according to the definition, a strongly limited automaton accepts an input either without any rewriting, or rewriting at least two consecutive cells. Hence, the strings $a, b \in$ PAL should be accepted by M without any rewriting. This implies that the 2-letter factors $\triangleright a$, $\triangleright b$, $a\triangleleft$, $b\triangleleft$ are allowed. On the other hand, factors ab and ba cannot be allowed, otherwise strings such as ab or ba will be accepted. This also implies that in order to accept the string $x = aba$ the machine has to make at least two rewritings.

Now, we suppose that M has an accepting computation on x which rewrites exactly two cells. According to the previous observation, these cells should be consecutive. Hence, there are 8 possibilities for the final content of the tape: $\triangleright\mathsf{YX}a\triangleleft$, $\triangleright a\mathsf{YX}\triangleleft$, $\triangleright\mathsf{XX}a\triangleleft$, $\triangleright a\mathsf{XX}\triangleleft$, and the ones obtained by switching X and Y. Without loss of generality we restrict our analysis to the first 4 possibilities. We can show that

all of them give a contradiction. The details are presented in Table 1, which contains an horizontal block for each possibility and considers several subcases. We illustrate our argument, just discussing some of them.

Suppose the final content of the tape in an accepting computation on x is $\triangleright \mathtt{Y}\mathtt{X} a \triangleleft$. We consider all the possibilities for the final content of the tape in an accepting computation on $y = aa$ (2nd column). For example, on the 5th line we suppose that the final content on y is $\triangleright \mathtt{X}\mathtt{Y} \triangleleft$. Then, on the string $x' = aaba$ (3rd column) there exists a computation which first rewrites the factor ab by $\mathtt{Y}\mathtt{X}$, to obtain the tape content $\triangleright a \mathtt{Y}\mathtt{X} a \triangleleft$ (4rd column), and then rewrites the two remaining a's, with the head which finally reaches the right end-marker and the string on the tape is $\triangleright \mathtt{X}\mathtt{Y}\mathtt{X}\mathtt{Y} \triangleleft$ (5th column). The rewriting of the factor ab is obtained by moving to the right from the leftmost input symbol and, from the second tape cell, by imitating the behavior of M on the prefix ab of x, so reaching the rightmost a, after rewriting ab by $\mathtt{Y}\mathtt{X}$. From the rightmost a, the behavior of M on x' is simulated, moving to the left, and rewriting the two remaining a's (ignoring the cells already rewritten), as in the computation on y, and finally reaching the right end-marker. The 2-letter factors of the final tape content on x', namely $\triangleright \mathtt{X}$, $\mathtt{X}\mathtt{Y}$, $\mathtt{Y}\mathtt{X}$, $\mathtt{Y} \triangleleft$, are also 2-letter factors of the final tape contents in at least one the two considered accepting computation on x and y. This implies that those factor are allowed and hence that x' is accepted by M. This is a contradiction since $x \notin$ PAL.

The other cases in the first and in the second block of the table are proved in a similar way. In the last two blocks, we obtain an accepting computation on x' by combining the computation on x with itself. □

As a consequence of Lemma 8.1, since the working alphabet of M consists of two symbols $\mathtt{X}$ and $\mathtt{Y}$, there are 8 possibilities for the final tape contents in accepting computations on strings $x = aba$ and $y = bab$. Among them, 6 can be excluded as illustrated in Table 2. (The table presents 3 possibilities in the case of the string x. The other 3 are obtained by exchanging $\mathtt{X}$ and $\mathtt{Y}$. Furthermore, the result can be extended to the string z by exchanging a and b.) The only remaining possibilities for the final tape contents in accepting computations on x and z are $\triangleright \mathtt{X}\mathtt{Y}\mathtt{X} \triangleleft$ and $\triangleright \mathtt{Y}\mathtt{X}\mathtt{Y} \triangleleft$. We now inspect all the combinations.

- The final tape contents on x and z are the same.
  Without loss of generality we assume that the content is $\triangleright \mathtt{X}\mathtt{Y}\mathtt{X} \triangleleft$. On input $x' = abababbabba$, M can simulate the computation on x on the factor from position 3 to position 5 and then the computation on z on the factor from position 7 to 9 so obtaining $\triangleright a\, b\, \mathtt{X}\mathtt{Y}\mathtt{X}\, b\, \mathtt{X}\mathtt{Y}\mathtt{X}\, b\, a \triangleleft$ on the tape. Then, moving the head to the rightmost a the machine can simulate the moves of the computation on x. The rightmost a is rewriting by $\mathtt{X}$, while reversing the head direction, all the b's are rewritten by $\mathtt{Y}$, all cells already rewritten are ignored and, finally, the leftmost a is rewritten by $\mathtt{X}$, while turning to the right. Hence the machine continues to move to the right, up to reach the end-marker, with the tape containing $\triangleright \mathtt{X}\mathtt{Y}\mathtt{X}\mathtt{Y}\mathtt{X}\mathtt{Y}\mathtt{X}\mathtt{Y}\mathtt{X}\mathtt{Y}\mathtt{X} \triangleleft$. The 2-letters factors of the tape content are $\triangleright \mathtt{X}$, $\mathtt{X}\mathtt{Y}$, $\mathtt{Y}\mathtt{X}$, and $\mathtt{X} \triangleleft$, which are also factors of the final tape content in the accepting computation on x. Hence, they are allowed. This implies that x' is accepted, which is a contradiction since $x' \notin$ PAL.

- The final tape contents on x and z are different.
  Without loss of generality we assume that the contents are $\triangleright \mathtt{X}\mathtt{Y}\mathtt{X} \triangleleft$ and $\triangleright \mathtt{Y}\mathtt{X}\mathtt{Y} \triangleleft$, respectively. On input $x' = ababab$, M can simulate first the accepting computation on x, rewriting the first three input symbols, and then, the accepting computation on y rewriting the remaining input symbols,

final tape on x	final tape on y	x'	intermediate tape	final tape on x'
$\triangleright$YX$a\triangleleft$	$\triangleright a\,a\triangleleft$	$abaa$	$\triangleright$YX$a\,a\triangleleft$	$\triangleright$YX$a\,a\triangleleft$
$\triangleright$YX$a\triangleleft$	$\triangleright$YY$\triangleleft$	$aaaba$	$\triangleright$YY$a\,b\,a\triangleleft$	$\triangleright$YYYX$a\triangleleft$
$\triangleright$YX$a\triangleleft$	$\triangleright$XX$\triangleleft$	$abaa$	$\triangleright$YX$a\,a\triangleleft$	$\triangleright$YXXX$\triangleleft$
$\triangleright$YX$a\triangleleft$	$\triangleright$YX$\triangleleft$	ab	$\triangleright$YX$\triangleleft$	$\triangleright$YX$\triangleleft$
$\triangleright$YX$a\triangleleft$	$\triangleright$XY$\triangleleft$	$aaba$	$\triangleright a$YX$a\triangleleft$	$\triangleright$XYXY$\triangleleft$
$\triangleright a$YX$\triangleleft$	$\triangleright a\,a\triangleleft$	$aaba$	$\triangleright a\,a$YX$\triangleleft$	$\triangleright a\,a$YX$\triangleleft$
$\triangleright a$YX$\triangleleft$	$\triangleright$YY$\triangleleft$	$aaba$	$\triangleright$YY$b\,a\triangleleft$	$\triangleright$YYYX$\triangleleft$
$\triangleright a$YX$\triangleleft$	$\triangleright$XX$\triangleleft$	$abaaa$	$\triangleright a$YX$a\,a\triangleleft$	$\triangleright a$YXXX$\triangleleft$
$\triangleright a$YX$\triangleleft$	$\triangleright$YX$\triangleleft$	ba	$\triangleright$YX$\triangleleft$	$\triangleright$YX$\triangleleft$
$\triangleright a$YX$\triangleleft$	$\triangleright$XY$\triangleleft$	$abaa$	$\triangleright a$YX$a\triangleleft$	$\triangleright$XYXY$\triangleleft$
$\triangleright$XX$a\triangleleft$	–	$aabba$	$\triangleright a$XX$b\,a\triangleleft$	$\triangleright$XXXX$a\triangleleft$
$\triangleright a$XX$\triangleleft$	–	$abbaa$	$\triangleright a\,b$XX$a\triangleleft$	$\triangleright a$XXXX$\triangleleft$

Table 1. Proof of Lemma 8.1: we cannot have an accepting computation on $x = aba$ which rewrites exactly two tape cells. Each row in the table represents a case of the proof and leads to a contradiction, by the following argument. Suppose that the final tape content in the accepting computation on x is the string in the 1st column and the final tape content in an accepting computation on $y = aa$ is the string in the 2nd column. Combining parts of those two computations, we can prove that on the input x' in the 3rd column there exists a computation such that the head reaches the right end-marker having on the tape the string in the 5th column. (In the 4th column is represented the tape content at the intermediate step of such a computation, when after moving from right to left a factor has been rewritten and the head is turned to the right.) Since all the 2-letter factors of the final tape content on x' are also factors of the final content in at least one of the accepting computations on x and y, the computation on x' is accepting. Being $x' \notin$ PAL, this gives a contradiction. In the last two rows, we do not consider y: the accepting computation on x' is obtained by combining the computation on x with itself.

final tape on x	final tape on y	x'	intermediate tape	final tape on x'
$\triangleright$XXX$\triangleleft$	$\triangleright$XX$\triangleleft$	$abaaa$	$\triangleright$XXX$a\,a\triangleleft$	$\triangleright$XXXXX$\triangleleft$
$\triangleright$YXX$\triangleleft$	$\triangleright$YX$\triangleleft$	$aaaba$	$\triangleright a$YX$b\,a\triangleleft$	$\triangleright$YYXXX$\triangleleft$
$\triangleright$YYX$\triangleleft$	$\triangleright$YX$\triangleleft$	$abaaa$	$\triangleright a\,b$YX$a\triangleleft$	$\triangleright$YYYXX$\triangleleft$

Table 2. Suppose we have an accepting computation on $x = aba$ rewriting all tape cells. This is possible only reaching the rightmost input symbol, rewriting all the input while moving to the left, and then moving to the right reaching the end-marker. For each row in the table, if the final content of the tape is the string on the 1st column then there is a computation on $y = aa$ finally having on the tape the string in the 2nd column. (The computation on y imitates the given computation on x, without the moves on the letter b.) Combining such computations, we can obtain a computation on the string x' on the third column, which first rewrites an input factor obtaining the content in the 4th column and then rewrites the remaining symbols, reaching the right end-marker with the tape content represented in the 5th column. Since all the 2-letter factors of the final tape content on x' are also 2-letter factors of the tape content on x, which belongs to an accepting computation, the string x' should be accepted by M. Since $x' \notin$ PAL this is a contradiction.

to reaching the right end-marker with $\triangleright$XYXYXY$\triangleleft$ on the tape. As in the previous case, x' should be accepted, which is a contradiction since $x' \notin$ PAL.

As a consequence, we obtain the following:

Theorem 8.2. Each strongly limited automaton accepting the set PAL of palindromes needs a working alphabet of at least three symbols.

With similar techniques we can prove that two symbols are necessary to recognize palindromes of even length.

Actually, the above argument uses a finite set S of strings (the palindromes of length ≤ 3 and a small number of strings of length ≤ 11 that are not in PAL). Hence, a result as Theorem 8.2 holds for each language L such that $S \cap L = S \cap$ PAL. Notice that a such L could be a regular or even a finite language.

Corollary 8.3. There are infinitely many finite languages that require a working alphabet of at least three symbols to be recognized by strongly limited automata.

As a consequence of Theorem 8.2, strongly limited automata with a unary working alphabet recognize a *proper* subclass of context-free languages. We conjecture that the same is true for the extended model discussed in Section 7, namely almost strongly limited automata, which are allowed to make state changes while reading input symbols, when the working alphabet is restricted to one symbol. In fact, the devices so obtained are very close to *deleting automata*, a restriction of forgetting automata that deletes tape cells while moving to the left. This means that cells already rewritten are removed from the tape or, equivalently, they are ignored in subsequent steps. As proved in [10], deleting automata recognize a proper subclass of context-free languages. The only difference between almost strongly limited automata with a unary working alphabet and deleting automata is that in the former model a final scan of the tape is performed (including deleted and not rewritten positions) to check the membership to local language.

By the way, the set of palindromes can be recognized by an almost strongly limited automaton M using a one letter working alphabet, in the following way. In order to verify if the input x has the form $w\sigma w^R$, with $\sigma \in \Sigma \cup \{\epsilon\}$, at the beginning of the computation M in the initial state moves its head to some position, where it guess that the second half w^R of the input starts. Furthermore, M guesses whether or not the input length is odd. Then M rewrites the symbol, remembers it in the finite control, and turns its head to the left. If the input length was guessed to be odd, then M rewrites one more input symbol (which should be the central symbol), still moving to the left. Then M compares the current input symbol with the symbol stored in the finite control, if they match then M rewrites the tape cell and turn to the right to search the first cell not yet rewritten. In this way, the computation of M can continue by verifying matching between symbols in the second half of the input and symbol in the first half. The only nondeterministic decision is taken at the beginning, to select the first symbol of the second half and to guess whether the length of the input is odd.

Acknowledgments

The author wish to thank the anonymous reviewers for stimulating comments and useful remarks. In particular, the proof of Theorem 7.3 was strongly simplified according to a referee suggestion. Furthermore, one comment from the same referee gave to the author the inspiration for Corollary 8.3.

References

[1] 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.

[2] Engelfriet, J.: An Elementary Proof of Double Greibach Normal Form, *Inf. Process. Lett.*, **44**(6), 1992, 291–293.

[3] Glöckler, J.: Forgetting Automata and Unary Languages, *Int. J. Found. Comput. Sci.*, **18**(4), 2007, 813–827.

[4] Glöckler, J.: A Taxonomy of Deterministic Forgetting Automata, *Int. J. Found. Comput. Sci.*, **21**(4), 2010, 619–631.

[5] 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.

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

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

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

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

[10] Jancar, P., Mráz, F., Plátek, M.: Forgetting Automata and Context-Free Languages, *Acta Inf.*, **33**(5), 1996, 409–420.

[11] Kutrib, M., Pighizzini, G.: Recent trends in descriptional complexity of formal languages, *EATCS Bulletin*, **111**, 2013, 70–86.

[12] McNaughton, R., Papert, S. A.: *Counter-Free Automata (M.I.T. Research Monograph No. 65)*, The MIT Press, 1971, ISBN 0262130769.

[13] Okhotin, A.: Non-erasing Variants of the Chomsky-Schützenberger Theorem, in: *Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings* (H. Yen, O. H. Ibarra, Eds.), vol. 7410 of *Lecture Notes in Computer Science*, Springer, 2012, ISBN 978-3-642-31652-4, 121–129.

[14] Pighizzini, G., Pisoni, A.: Limited Automata and Regular Languages, *Int. J. Found. Comput. Sci.*, **25**(7), 2014, 897–916.

[15] Pighizzini, G., Pisoni, A.: Limited Automata and Context-Free Languages, *Fundam. Inform.*, **136**(1-2), 2015, 157–176.

[16] Rosenkrantz, D. J.: Matrix Equations and Normal Forms for Context-Free Grammars, *J. ACM*, **14**(3), 1967, 501–507.

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

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

[19] Yoshinaka, R.: An elementary proof of a generalization of double Greibach normal form, *Inf. Process. Lett.*, **109**(10), 2009, 490–492.