International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Non-Self-Embedding Grammars, Constant-Height Pushdown Automata, and Limited Automata*

Bruno Guillon
Université Clermont-Auvergne, France
bruno.guillon@uca.fr

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

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

Non-self-embedding grammars are a restriction of context-free grammars which does not allow to describe recursive structures and, hence, which characterizes only the class of regular languages. A double exponential gap in size from non-self-embedding grammars to deterministic finite automata is known. The same size gap is also known from constant-height pushdown automata and 1-limited automata to deterministic finite automata. Constant-height pushdown automata and 1-limited automata are compared with non-self-embedding grammars. It is proved that non-self-embedding grammars and constant-height pushdown automata are polynomially related in size. Furthermore, a polynomial size simulation by 1-limited automata is presented. However, the converse transformation is proved to cost exponential. Finally, a different simulation shows that also the conversion of deterministic constant-height pushdown automata into deterministic 1-limited automata costs polynomial.

1. Introduction

It is well known that the extra capability of context-free grammars with respect to regular ones is that of describing recursive structures as, for instance, nested parentheses, arithmetic expressions, typical programming language constructs. In terms of recognizing devices, this capability is implemented through the pushdown store, which is used to extend finite automata in order to make the resulting model, namely pushdown automata, equivalent to context-free grammars.

*A preliminary version of this paper has been presented at CIAA 2018 [B. Guillon, G. Pighizzini and L. Prigioniero, Non-Self-Embedding Grammars, Constant-Height Pushdown Automata, and Limited Automata, *CIAA 2018*, *LNCS* **10977** (Springer, 2018), pp. 186–197].

To emphasize this capability, in one of his pioneering papers, Chomsky investigated the *self-embedding* property [5]: a context-free grammar is self-embedding if it contains a variable A which, in some sentential form, is able to reproduce itself surrounded by two nonempty strings α and β, in symbols $A \overset{\star}{\Longrightarrow} \alpha A \beta$. Roughly speaking, this means that the variable A is "truly" recursive. He proved that, among all context-free grammars, only self-embedding ones can generate nonregular languages. Hence, *non-self-embedding grammars* are no more powerful than finite automata.

The relationships between the description sizes of non-self-embedding grammars and finite automata have been investigated in [1] and [17]. In the worst case, the size of a deterministic automaton equivalent to a given non-self-embedding grammar is doubly exponential in the size of the grammar. The gap reduces to a simple exponential in the case of nondeterministic automata.

Other formal models characterizing the class of regular languages and exhibiting gaps of the same order with respect to deterministic and nondeterministic automata have been investigated in the literature. Two of them are *constant-height pushdown automata* and 1-*limited automata*. The aim of this paper is to study the size relationships between non-self-embedding grammars, constant-height pushdown automata, and 1-limited automata, three models that restrict context-free acceptors to the level of regular recognizers.

Constant-height pushdown automata are standard nondeterministic pushdown automata where the amount of available pushdown store is fixed. Hence, the number of their possible configurations is finite. This implies that they are no more powerful than finite automata. The exponential and double exponential gaps from constant-height pushdown automata to nondeterministic and deterministic automata have been proved in [6]. Furthermore, in [2] the authors showed the interesting result that the gap from nondeterministic to deterministic constant-height pushdown automata is double exponential also. We can observe that both non-self-embedding grammars and constant-height pushdown automata are restrictions of the corresponding general models, where true recursions are not possible. In the first part of the paper we compare these two models by proving that they are polynomially related in size.

In the second part, we turn our attention to the size relationships between 1-limited automata and non-self-embedding grammars. For each integer $d > 0$, a *d-limited automaton* is a one-tape nondeterminstic Turing machine which is allowed to rewrite the content of each tape cell only in the first d visits. These models have been introduced by Hibbard in 1967, who proved that for each $d \geq 2$ they characterize context-free languages [7]. This yields a hierarchy of acceptors, merely obtained by restricting one-tape Turing machines, corresponding to Chomsky's classification. Furthermore, as shown in [21, Thm. 12.1], 1-limited automata are equivalent to finite automata. This equivalence has been investigated from the descriptional complexity point of view in [16], by proving exponential and double exponential gaps from 1-limited automata to nondeterministic and deterministic finite automata, respectively. Our main result is a construction transforming each non-self-embedding

$$\begin{tikzcd}
\text{NSE} \arrow[rr, bend left, "\text{poly (Thm 6)}"] & & h\text{-PDA} \arrow[ll, bend left, "\text{poly (Thm 10)}"] \arrow[rr, bend left, "\text{poly (Thm 15)}"] \arrow[dr, "\text{exp}"] & & 1\text{-LA} \arrow[ll, bend left, "\text{exp (Cor 18)}"] \arrow[dl, "\text{exp [16, Thm 4]}"] \\
& 2\text{DFA} \arrow[ur, "\text{exp (Thm 17)}"] \arrow[rr, dotted, bend left] & & 2\text{NFA} \arrow[ll, dashed, bend left, "\substack{?\\ \text{[Sakoda-Sipser problem]}}"] \arrow[ur, dotted] &
\end{tikzcd}$$

Figure 1. Some bounds discussed in the paper. Dotted arrows denote trivial relationships, while the dashed arrow indicates the famous Sakoda and Sipser's question [20]. The exponential cost of the simulation of h-PDAs by 2NFAs is discussed at the end of Section 4.2.

grammar into a 1-limited automaton of polynomial size. As a consequence, each constant-height pushdown automaton can be transformed into an equivalent 1-limited automaton of polynomial size. We also prove, using a different construction, that even the conversion of deterministic constant-height pushdown automata into deterministic 1-limited automata costs polynomial in size. For the converse transformation, we show that an exponential size is necessary. Indeed, we prove a stronger result by exhibiting, for each $n > 0$, a language L_n accepted by a two-way deterministic finite automaton with $O(n)$ states, which requires exponentially many states to be accepted even by an unrestricted pushdown automaton. From the cost of the conversion of 1-limited automata into nondeterministic automata, it turns out that for the conversion of 1-limited automata into non-self-embedding grammars an exponential size is also sufficient. Figure 1 summarizes the main results discussed in the paper.

2. Preliminaries

Given a set S, we denote by $\#S$ its cardinality, and by 2^S the family of all its subsets. We assume the reader familiar with notions from formal languages and automata theory, in particular with the fundamental variants of finite automata (1DFA*s*, 1NFA*s*, 2DFA*s*, 2NFA*s*, for short, where 1 and 2 mean *one-way* and *two-way*, respectively, and D and N mean *deterministic* and *nondeterministic*, respectively). For further details see, *e.g.*, [8]. The empty word is denoted by ε. Given a word $u \in \Sigma^*$, we denote by $|u|$ its length. For two-way devices operating on a tape, we use the special symbols $\triangleright$ and $\triangleleft$ not belonging to the input alphabet, respectively called the *left* and the *right endmarkers*, that surround the input word.

Given a *context-free grammar* (CFG, for short) $G = \langle V, \Sigma, P, S\rangle$, we denote by $L(G)$ the language it generates. The relations $\Rightarrow$ and $\overset{\star}{\Longrightarrow}$ are defined in the usual way. For a fixed alphabet Σ, we measure the *size of* G by considering the total number of symbols used to specify it, defined as $\sum_{(A\to\alpha)\in P}(2+|\alpha|)$, *cf.* [10]. The *production graph* of G is a directed graph which has V as vertex set and contains an edge from A to B if and only if there is a production $A \to \alpha B\beta$ in P, for $A, B \in V$ and some $\alpha, \beta \in (V \cup \Sigma)^*$. The strongly connected components of the production graph induce a partial order on variables: a variable A is *smaller than* B if there exists a path from A to B and no path from B to A.

Definition 1. *Let $G = \langle V, \Sigma, P, S\rangle$ be a context-free grammar. A variable $A \in V$ is said to be* self-embedded *when there are two strings $\alpha, \beta \in (V \cup \Sigma)^+$ such that $A \overset{\star}{\Longrightarrow} \alpha A \beta$. The grammar G is* self-embedding *if it contains at least one self-embedded variable, otherwise G is* non-self-embedding *(*NSE*, for short).*

Chomsky proved that NSE grammars generate only regular languages, *i.e.*, they are no more powerful than finite automata [4,5]. As shown in [1], given a grammar G it is possible to decide in polynomial time whether or not it is NSE.

A *pushdown automaton* (PDA) is usually obtained from a nondeterministic finite automaton by adding a pushdown store, containing symbols from a *pushdown alphabet* Γ. Following [2,6], we consider PDAs in the following form, where the transitions manipulating the pushdown store are clearly distinguished from those reading the input tape. Furthermore, we consider a restriction of the model in which the capacity of the pushdown store is bounded by some constant $h \in \mathbb{N}$.

Definition 2. *For $h \in \mathbb{N}$, a* pushdown automaton of height h *(h-*PDA*) is a tuple $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ where Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, Σ is the input alphabet, Γ is the pushdown alphabet, and $\delta \subseteq Q \times (\{\varepsilon\} \cup \Sigma \cup \{-, +\}\Gamma) \times Q$ is the transition relation with the following meaning:*

- $(p, \varepsilon, q) \in \delta$: *$\mathcal{A}$ can reach the state q from the state p without using the input tape nor the pushdown store (these transitions are also called ε-*moves*);*
- $(p, a, q) \in \delta$: *$\mathcal{A}$ can reach the state q from the state p by reading the symbol a from the input but without using the pushdown store;*
- $(p, -X, q) \in \delta$: *if the symbol on the top of the pushdown store is X, $\mathcal{A}$ can reach the state q from the state p by popping off X, not using the input tape;*
- $(p, +X, q) \in \delta$: *if the number of symbols contained in the pushdown store is less than h, $\mathcal{A}$ can reach the state q from the state p by pushing X on the pushdown store, without using the input tape.*

The model *accepts* an input word $w \in \Sigma^*$ if, starting from the leftmost tape cell in the initial state q_0 with an empty pushdown store, it can eventually reach an accepting state $q_f \in F$, after having read all the input symbols.

Without the restriction on the pushdown height, the model is equivalent to classical pushdown automata, while preserving comparable size (namely, translations both ways have at most polynomial costs, see [2]). By contrast, 0-PDAs are exactly 1NFAs, since they can never push symbols.

An h-PDA can be replaced by an equivalent standard PDA without the built-in limit on pushdown size, by counting in the finite control the pushdown height, with an increase in the number of states that is linear in h. For this reason, the size of an h-PDA over a fixed input alphabet Σ is given by a polynomial in $\#Q$, $\#\Gamma$, and h [6].

One-limited automata (1-LAs, for short) extend two-way finite automata by providing the ability to overwrite each tape cell at its first visit by the head. This extension does not increase the expressiveness of the model. However, they can be significantly smaller than equivalent finite automata. For instance, the size gaps from 1-LAs to 1NFAs and 1DFAs are exponential and double exponential, respectively [16], while 2NFAs can require a size that is exponential with respect to that of deterministic 1-LAs even in the unary case, as shown in [18] improving [11,12].

Definition 3. *A* 1-limited automaton *is a tuple* $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$*, where* Q, Σ, q_0, F *are defined as for* 2NFA*s,* Γ *is a finite* working alphabet *such that* $\Sigma \subseteq \Gamma$*, and* $\delta : Q \times \Gamma_{\rhd\lhd} \to 2^{Q\times\Gamma_{\rhd\lhd}\times\{-1,+1\}}$ *is the* transition function*, where* $\Gamma_{\rhd\lhd} = \Gamma \cup \{\rhd, \lhd\}$ *with* $\rhd, \lhd \notin \Gamma$*.*

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 by a new symbol, and moves its head to one position backward or forward. However, replacing symbols is subject to some restrictions, which, essentially, allow to modify the content of a cell during the first visit only. Formally, symbols from Σ shall be replaced by symbols from $\Gamma \setminus \Sigma$, while symbols from $\Gamma_{\rhd\lhd} \setminus \Sigma$ are never overwritten. In particular, at any time, both special symbols $\rhd$ and $\lhd$ occur exactly once on the tape at the respective left and right boundaries. *Acceptance* for 1-LAs can be defined in several ways, for instance we can say that a 1-LA $\mathcal{A}$ accepts an input word if, starting from the left endmarker in the initial state, a computation eventually reaches the right endmarker in an accepting state. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

The *size of a* 1-LA $\langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ over a fixed alphabet Σ is given by a polynomial in $\#Q$ and $\#\Gamma$. The *size of an n-state* NFA (resp., DFA) is quadratic (resp., linear) in n.

3. NSE Grammars versus h-PDAs

In this section we prove that NSE grammars and h-PDAs are polynomially related in size.

3.1. *From* NSE *grammars to h-PDAs*

In [1], the authors showed that NSE grammars admit a particular form based on a decomposition into finitely many simpler grammars, that will be now recalled.

First of all, we remind the reader that a grammar is said *right-linear* (resp., *left-linear*), if each production is either of the form $A \to wB$ (resp., $A \to Bw$), or of the form $A \to w$, for some $A, B \in V$ and $w \in \Sigma^*$. It is well known that right- or left-linear grammars generate exactly the class of regular languages.

Given two CFGs $G_1 = \langle V_1, \Sigma_1, P_1, S_1 \rangle$ and $G_2 = \langle V_2, \Sigma_2, P_2, S_2 \rangle$ with $V_1 \cap V_2 = \emptyset$, the *$\oplus$-composition* of G_1 and G_2 is the grammar $G_1 \oplus G_2 = \langle V, \Sigma, P, S \rangle$, where $V = V_1 \cup V_2$, $\Sigma = (\Sigma_1 \setminus V_2) \cup \Sigma_2$, $P = P_1 \cup P_2$, and $S = S_1$. Intuitively, the grammar $G_1 \oplus G_2$ generates all the strings which can be obtained by replacing in any string $w \in L(G_1)$ each symbol $A \in \Sigma_1 \cap V_2$ with some string derived in G_2 from the variable A (notice that the definition of $G_1 \oplus G_2$ does not depend on the start symbol S_2 of G_2). The $\oplus$-composition is associative and preserves the non-self-embedding property of grammars [1]. The decomposition presented in the following result was obtained in [1], while its size is discussed in [17].

Theorem 4. *For each* NSE *grammar G there exist g grammars $G_1, G_2, \ldots, G_g$ such that $G = G_1 \oplus G_2 \oplus \cdots \oplus G_g$, where each G_i is either left- or right-linear. Furthermore the sum of sizes of G_i's is linear in the size of G.*

Studying the relationships between NSE grammars and PDAs, in [1] the authors claimed that from any NSE grammar in *canonical normal form* (namely with productions $A \to a\gamma$ or $A \to \gamma$, $A \in V$, $a \in \Sigma$ and $\gamma \in V^*$), by applying a standard transformation, it is possible to obtain an equivalent constant-height PDA. Unfortunately, the argument fails when the grammar contains left-recursive derivations, *i.e.*, derivations of the form $A \overset{\star}{\Longrightarrow} A\gamma$, with $\gamma \neq \varepsilon$. For them, the resulting PDA has computations with arbitrarily high pushdown stores. This problem can be fixed by replacing each left-linear grammar corresponding to a strongly connected component of the production graph of the given NSE grammar by a set of equivalent right-linear grammars, as shown in the following lemma:

Lemma 5. *Each* NSE *grammar can be converted into an equivalent* NSE *grammar of polynomial size which can be expressed as a $\oplus$-composition of right-linear grammars.*

Proof. First of all, we observe that from each left-linear grammar $G = \langle V, \Sigma, P, S \rangle$ we can obtain an equivalent right-linear grammar $G' = \langle V \cup \{S'\}, \Sigma, P', S' \rangle$ whose size is linear in the size of G, with one more variable $S' \notin V$ and the following productions:

- $B \to w$, for each $S \to Bw$ in P,
- $B \to wA$, for each $A \to Bw$ in P (including $S \to Bw$),
- $S' \to wA$, for each $A \to w$ in P.

We point out that if we apply the above transformation to the grammar $G_{|\hat{S}} = \langle V, \Sigma, P, \hat{S} \rangle$ obtained by changing the initial symbol of G into $\hat{S} \in V$, then the set of productions of the resulting grammar could be different from P'.

Now suppose to have an NSE grammar G, where $G = G_1 \oplus G_2 \oplus \cdots \oplus G_g$ and each G_i is left-linear or right-linear. The idea is to define an equivalent grammar G' by keeping each right-linear G_i, and by replacing each left-linear G_i by an equivalent right-linear grammar. However, when doing grammar $\oplus$-composition, we could use derivations of G_i that begin from variables of V_i different than the start symbol S_i. For this reason, from each left-linear G_i we obtain *a family* of right-linear grammars G'_{iA}, with $A \in V_i$, where G'_{iA} is equivalent to the grammar $G_{i|A} = \langle V_i, \Sigma_i, P_i, A\rangle$, and all the variables of $G_{i|A}$, with the exception of A, are renamed in such a way that the grammars in the family do not share any variable. Let G'_i be a grammar whose sets of variables and productions are the unions of the corresponding sets in $G_{i|A}$. We then replace each left-linear G_i by the right linear grammar G'_i. It can be verified that the size of the resulting grammar G' is at most quadratic in the size of G. $\square$

We now prove that each NSE grammar can be transformed into an h-PDA of polynomial size.

Theorem 6. *Each* NSE *grammar* $G = \langle V, \Sigma, P, S\rangle$ *can be converted into an* h-PDA $\mathcal{A}$ *with both* h *and the size of* $\mathcal{A}$ *polynomial in the size of* G.

Proof. We start from a NSE grammar $G = G_1 \oplus G_2 \oplus \cdots \oplus G_g$ such that the sum of the sizes of the G_i's is polynomial in the size of G, and where each $G_i = \langle V_i, \Sigma_i, P_i, S_i\rangle$ is right-linear, by Theorem 4 and Lemma 5. First, as in the construction presented in [1], we show that if a variable $A \in V_i$, $1 \le i \le g$, derives a string $x\alpha$ by a leftmost derivation, *i.e.*, $A \xRightarrow[lm]{\star} x\alpha$, where x is the longest prefix of $x\alpha$ consisting only of terminal symbols, then the length of α is linear in $g - i$. More precisely, we claim that $|\alpha| \le K(g-i)+1$, where K is the maximum length of production right-hand sides. We are going to prove this claim by induction on the number h of steps of the derivation $A \xRightarrow[lm]{\star} x\alpha$.

For $h = 0$ we have $|\alpha| = 1$ and, hence, the claim is trivial.

Consider now $h > 0$. If $\alpha = \varepsilon$ then the claim is obvious. Otherwise, let $A \to X_1X_2\cdots X_s$ be the first production used in the derivation under consideration, *i.e.*,

$$A \Longrightarrow X_1X_2\cdots X_s \xRightarrow[lm]{\star} \alpha_1\alpha_2\cdots\alpha_s = x\alpha$$

where $X_k \in \Sigma \cup \bigcup_{j=i}^{g} V_j$ and $X_k \xRightarrow[lm]{\star} \alpha_k$, for $k = 1, \ldots, s$. Let ℓ, $1 \le \ell \le s$, be the smallest index such that α_ℓ contains at least one variable. Hence, we can write $x = x'x''$ where $x' = \alpha_1\alpha_2\cdots\alpha_{\ell-1}$, $x''\alpha' = \alpha_\ell$, $\alpha_k = X_k$ for $k = \ell+1, \ldots, s$, and $\alpha = \alpha' X_{\ell+1}\cdots X_s$.

Since the derivation $X_\ell \xRightarrow[lm]{\star} x''\alpha'$ consists of less than h steps, from the induction hypothesis we get that $|\alpha'| \le K(g-j)+1$, where j is the index satisfying $X_\ell \in V_j$. Thus, $|\alpha| = |\alpha'| + s - \ell \le K(g-j) + 1 + s - \ell$.

Due to the fact that $s - \ell < K$, when $j > i$ from the last inequality we obtain $|\alpha| < K(g-i)+1$. Furthermore, since G_i is right-linear, the case $j = i$ could occur only when $\ell = s$, thus implying $\alpha = \alpha'$ and, hence $|\alpha| = |\alpha'| \le K(g-i)+1$. This completes the proof of the claim.

From the grammar G we can apply a standard construction to obtain a PDA M which simulates a leftmost derivation of G, by replacing any variable A occurring on the top of the pushdown by the right-hand side of a production $A \to \alpha$, and by popping off the pushdown any terminal symbol occurring on the top and matching the next input symbol (for details see, *e.g.*, [8]). After consuming an input prefix y, the pushdown store of M can contain any string $z\alpha$ such that $S \overset{\star}{\underset{lm}{\Longrightarrow}} yz\alpha$, yz is the longest prefix of $yz\alpha$ consisting only of terminal symbols, and z is a suitable factor of the string which was most recently pushed on the pushdown. Since $|z| \le K$ and, according to the first part of the proof $|\alpha| \le K(g-1)+1$, we conclude that the pushdown height is bounded by $Kg+1$. Hence, M is a constant-height PDA. Finally, M can be converted in the form given in Definition 2, by keeping its size polynomial. □

3.2. *From h-*PDA*s to* NSE *grammars*

We first show that, modulo acceptance of the empty word, with only a polynomial increase in the size we can transform any h-PDA in a special form. Subsequently, we will associate to any h-PDA in such form, a NSE grammar and show that it is equivalent to the h-PDA.

Lemma 7. *For each h-PDA $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ there exists an h-PDA $\mathcal{A}' = \langle Q', \Sigma, \Gamma', \delta', q_-, \{q_+\}\rangle$ and a mapping $\widetilde{h} : \Gamma' \to \{1, \dots, h\}$ such that:*

- $L(\mathcal{A}') = L(\mathcal{A}) \setminus \{\varepsilon\}$*;*
- *$\mathcal{A}'$ has polynomial size with respect to $\mathcal{A}$;*
- *$\mathcal{A}'$ accepts with empty pushdown;*
- *$\mathcal{A}'$ has no ε-moves;*
- *each symbol $X \in \Gamma'$ can appear on the pushdown only at height $\widetilde{h}(X)$;*
- *every nonempty computation path of $\mathcal{A}'$ starting and ending with the same symbol X on the top of the pushdown, and never popping off X in the meantime, consumes some input letters.*

Proof. First, we create two new states q_- and q_+, intuitively the new initial and unique final states, respectively, and we add transitions (q_-, ε, q_0) and (p, ε, q_+) for each $p \in F$. Furthermore, in order to empty the pushdown store at the end of the accepting computations, we add the transition $(q_+, -X, q_+)$ for each $X \in \Gamma$. We denote by $Q_\star$ the set $Q \cup \{q_-, q_+\}$.

Second, by extending $Q_\star$ to $Q_\star \times \{0, \dots, h\}$, we can suppose that each state stores the current height of the pushdown as second component. After this change, we set $(q_-, 0)$ as initial state and $(q_+, 0)$ as unique final state (as a consequence, acceptance is necessarily made with empty pushdown). We then set $\Gamma' = \Gamma \times \{1, \dots, h\}$ and we modify the transitions in such a way that a symbol $(\gamma, i) \in \Gamma'$ can be pushed only from a state in $Q \times \{i-1\}$, *i.e.*, only at pushdown height i. The mapping $\widetilde{h}$ is then defined on Γ' as the projection over the second component.

Now, for each state $(p,i) \in Q_\star \times \{0,\ldots,h\}$, we define the set $E_{(p,i)}$ of states (q,i) which are accessible from (p,i) by using only transitions in

$$(Q_\star \times \{i,\ldots,h\}) \times (\{\varepsilon\} \cup \{-,+\}\Gamma) \times (Q_\star \times \{i,\ldots,h\}).$$

The restriction to states from $Q_\star \times \{i,\ldots,h\}$ ensures that the considered computation paths can never pop off symbols under their initial level, while the restriction on the set of actions forbids any reading of the input. We first replace such computations by a single ε-move. This can be achieved as follows:

- we create a transition $((p,i),\varepsilon,(q,i))$ for each $(q,i) \in E_{(p,i)}$;
- we add a new state component storing an element in $\{\texttt{push},\texttt{pop},\texttt{read}\}$ that saves the last operation performed during the computation (with the natural meaning, ε-moves being not considered as operations) and we forbid transitions of the form $((p,j),-X,(q,j-1))$ whenever the last operation is `push`. (For simplicity, the newly-introduced component will not appear in the end of the proof.)

After such transformation, the only computations that start and end with same symbol on the top of the pushdown, without popping off symbols under the corresponding level, and without scanning any input letter, are necessarily sequences of ε-moves. Hence, each set $E_{(p,i)}$, which is kept unchanged by the above transformation, is now equal to the set of states accessible from (p,i) through a sequence of ε-moves.

We finally proceed to the elimination of ε-moves, using classical techniques. First, we consider the set $E_{(q_-,0)}$ of states that are accessible from the initial configuration through a sequence of ε-moves. For each state $(p,0) \in E_{(q_-,0)}$ and each transition $((p,0),\varkappa,(r,i))$ with $\varkappa \neq \varepsilon$, we create a transition $((q_-,0),\varkappa,(r,i))$. We then remove every ε-move from q_-, *i.e.*, every transition of the form $((q_-,0),\varepsilon,(p,0))$. As a consequence, the empty word cannot be accepted by the resulting h-PDA. However, since every computation of $\mathcal{A}$ accepting a nonempty word should perform a transition of the form $((p,0),\varkappa,(r,i))$ with $\varkappa \neq \varepsilon$ at some point, our transformation preserves acceptance of nonempty words.

Lastly, the remaining ε-moves are eliminated as follows. For each transition of the form $((p,i),\varkappa,(q,j))$ with $\varkappa \neq \varepsilon$ and each $(r,j) \in E_{(q,j)}$, we create the transition $((p,i),\varkappa,(r,j))$. We finally remove all remaining ε-moves.

The complete construction has polynomial cost and the resulting h-PDA $\mathcal{A}'$ accepts an input word if and only if the word is nonempty and was accepted by the original h-PDA $\mathcal{A}$. Acceptance is furthermore done by empty pushdown, indeed the only final state $(q_+,0)$ stores the information that the current pushdown height is 0. Moreover, the projection $\widetilde{h}$ over the pushdown alphabet Γ', associates to each pushdown symbol, the only height index to which it may appear in the pushdown store. $\square$

By adapting the classical construction of CFGs from PDAs (see, *e.g.*, [8, Sec. 6.3]), from an h-PDA $\mathcal{A} = \langle Q,\Sigma,\Gamma,\delta,q_-,\{q_+\}\rangle$ in the form of Lemma 7, we define a grammar $G = \langle V, \Sigma, P, S\rangle$, where V consists of an initial symbol S and of triples of the form $[pXq]$ and $\langle pXq\rangle$, for $q, p \in Q$, $X \in \Gamma_\perp = \Gamma \cup \{\perp\}$, with the new symbol $\perp \notin \Gamma$ denoting the "missed top" in the empty pushdown store.

Before defining the set P of productions, we give a short explanation of the meaning of the variables we just introduced. Each triple $[pXq]$ is used to generate any string which is consumed in a computation path $\mathcal{C}$ that starts in the state p with X on the top of the stack and ends in the state q with the *same occurrence* of X on the top of the stack, i.e., the height of the stack at the beginning and at the end of $\mathcal{C}$ is the same and it cannot be lower in between. Notice that this implies that $\mathcal{C}$ does not depend on the symbols that are on the stack at the beginning and at the end of $\mathcal{C}$. Any string generated by a variable $\langle pXq\rangle$ is consumed in a computation path $\mathcal{C}$ which, besides the previous conditions, does not visit other configurations with same stack height, namely, either $\mathcal{C}$ consists of a single step, or it starts by pushing a symbol which is popped off only at its last step. As we will prove, the use of two types of triples allows to obtain an NSE grammar.

We now list the productions in the set P and then we will prove that the grammar has the desired behavior:

(i) $\langle pXq\rangle \to a$, for $(p, a, q) \in \delta$
(ii) $\langle pXq\rangle \to [p'Yq']$, for $(p, +Y, p'), (q', -Y, q) \in \delta$, *i.e.*, push and pop of a same symbol Y
(iii) $[pXq] \to \langle pXr\rangle[rXq]$, for $p, q, r \in Q$, $X \in \Gamma_\perp$
(iv) $[pXq] \to \langle pXq\rangle$, for $p, q \in Q$, $X \in \Gamma_\perp$
(v) $S \to [q_-\perp q_+]$.

Lemma 8. *For each $x \in \Sigma^*$, $p, q \in Q$ and $X \in \Gamma_\perp$, $[pXq] \overset{\star}{\Longrightarrow} x$ if and only if there exists a computation path $\mathcal{C}$ of $\mathcal{A}$ satisfying the following conditions:*

(1) $\mathcal{C}$ starts in the state p and ends in the state q, in both these configurations the symbol at the top of the pushdown is X;[a]
(2) along $\mathcal{C}$ the pushdown is never popped off under height $\widetilde{h}(X)$.
(3) the input factor consumed along $\mathcal{C}$ is x.

Furthermore, $\langle pXq\rangle \overset{\star}{\Longrightarrow} x$ if and only if besides the above conditions (1) and (3), the following condition (stronger than (2)) is satisfied:

(2′) in all configurations of $\mathcal{C}$ other than the first and the last one, the pushdown height is greater than $\widetilde{h}(X)$.

Proof. First of all, we are going to prove that for all x, p, q, X as in the statement of the lemma, $[pXq] \overset{\star}{\Longrightarrow} x$ implies (1), (2), and (3), while $\langle pXq\rangle \overset{\star}{\Longrightarrow} x$ implies (1), (2′),

[a]When $X = \perp$ the pushdown store in these configurations is empty. Furthermore, we stipulate $\widetilde{h}(\perp) = 0$.

and (3). We proceed by induction on the length $k \geq 1$ of the derivation $[pXq] \overset{k}{\Longrightarrow} x$ or $\langle pXq \rangle \overset{k}{\Longrightarrow} x$.

For $k = 1$, there are no derivations $[pXq] \overset{1}{\Longrightarrow} x$, while $\langle pXq \rangle \overset{1}{\Longrightarrow} x$ implies that x is a terminal symbol and $(p, x, q) \in \delta$ (the production is of the form (i)), from which (1), (2′), and (3) trivially follow.

Suppose now $k > 1$. The first production applied in a derivation $[pXq] \overset{k}{\Longrightarrow} x$ is either of the form (iii) or of the form (iv). In the first case we have $[pXq] \Longrightarrow \langle pXr \rangle [rXq] \overset{k-1}{\Longrightarrow} x$, $\langle pXr \rangle \overset{k'}{\Longrightarrow} x'$, $[rXq] \overset{k''}{\Longrightarrow} x''$, for some $r \in Q$, $1 \leq k', k'' < k$, $k' + k'' = k - 1$, $x', x'' \in \Sigma^+$ such that $x'x'' = x$. Using the induction hypothesis, we can find two computations path $\mathcal{C}'$ and $\mathcal{C}''$, from state p to r and from state r to q, respectively, with X at the top of the pushdown at the beginning and at the end, such that the pushdown is never popped under its level at the beginning of these paths, and consuming the factors x' and x'', respectively. By concatenating these two paths, we find the path $\mathcal{C}$ satisfying in (1), (2), and (3).

If $[pXq] \Longrightarrow \langle pXq \rangle \overset{k-1}{\Longrightarrow} x$ (*i.e.*, the first production applied is of the form (iv)), (1), (2), and (3) follow from the induction hypothesis applied to the derivation $\langle pXq \rangle \overset{k-1}{\Longrightarrow} x$.

We now consider a derivation $\langle pXq \rangle \overset{k}{\Longrightarrow} x$, with $k > 1$. The first step can only be of the form (ii), namely $\langle pXq \rangle \Longrightarrow [p'Yq'] \overset{k-1}{\Longrightarrow} x$. From the induction hypothesis, there is a computation path $\mathcal{C}'$ from state p' to state q' which starts and ends with Y at the top of the pushdown, never popping off the pushdown under the initial level, and consuming x from the input tape. From a configuration with state p and X at the top of the pushdown, $\mathcal{A}$ can start a computation path which pushes Y, simulates $\mathcal{C}'$, and finally pops Y off the pushdown. While simulating $\mathcal{C}'$ the pushdown always contains the symbol Y over X. Hence, it is higher than in the first and in the last configuration of $\mathcal{C}$. This proves (1), (2′), and (3).

To prove the converse implications, we proceed by induction on the length k of the computation path $\mathcal{C}$ satisfying conditions (1), (2), (3), and, possibly, the further condition (2′).

If $k = 1$ then $\mathcal{C}$ should consist only of one move, which consumes the input symbol $x = a$ and does not modify the pushdown store. According to the definition of G, the only possible derivations corresponding to such path are $\langle pXq \rangle \Longrightarrow x$ and $[pXq] \Longrightarrow \langle pXq \rangle \Longrightarrow x$.

For $k > 1$ we consider two cases. First we suppose that $\mathcal{C}$ satisfies (1), (2), (3), but does not satisfy (2′). We decompose $\mathcal{C}$ in two shorter paths $\mathcal{C}'$ and $\mathcal{C}''$ that are delimited by the *first configuration* which is reached in $\mathcal{C}$ with the same pushdown height as at the beginning and at the end of $\mathcal{C}$. These two paths satisfy (1), (2), (3). Furthermore, $\mathcal{C}'$ satisfies also (2′). By the induction hypothesis, we get that $\langle pXr \rangle \overset{\star}{\Longrightarrow} x'$, $[rXq] \overset{\star}{\Longrightarrow} x''$, where r is the state reached at the end of $\mathcal{C}'$ and $x'x'' = x$. Using production (iv) we obtain the derivation $[pXq] \Longrightarrow \langle pXr \rangle [rXq] \overset{\star}{\Longrightarrow} x'x'' = x$.

If $\mathcal{C}$ satisfies (1), (2′), and (3), then it should start in state p with a push of a symbol Y moving in a state p', ends after a pop of the same symbol Y from a state q' to state q, where the symbol Y is never popped off the pushdown in between. The path $\mathcal{C}'$, consisting of $k-2$ moves, obtained by removing from $\mathcal{C}$ the first and the last move, consumes the same input string x which is consumed by $\mathcal{C}$. From the induction hypothesis, we obtain that $[p'Yq'] \overset{\star}{\Longrightarrow} x$ and, considering (ii), $\langle pXq\rangle \Longrightarrow [p'Yq'] \overset{\star}{\Longrightarrow} x$. Furthermore, by (iv), we also obtain $[pXq] \overset{\star}{\Longrightarrow} x$. □

From Lemma 8, considering production (v), we can conclude that the grammar G so defined is equivalent to the given PDA $\mathcal{A}$.

Lemma 9. *The above-defined grammar G is non-self-embedding.*

Proof. From productions (ii), we observe that $\widetilde{h}(X') > \widetilde{h}(X)$ for any possible variable $\langle p'X'q'\rangle$ or $[p'X'q']$ that can appear in a sentential form from a variable $\langle pXq\rangle$. Hence, each variable $\langle pXq\rangle$ is not self-embedded.

Now, we consider any variable of the form $[pXq]$. We observe that in each derivation $[pXq] \overset{+}{\Longrightarrow} \alpha[pXq]\beta$, $\alpha, \beta \in (V \cup \Sigma)^*$, the occurence of $[pXq]$ on the right-hand side can be obtained only if, each time the rightmost variable is rewritten, productions of the form (iii) are used. Hence, the string β must be empty. This allows us to conclude that each $[pXq]$ is not self-embedded. □

By combining the previous results, we obtain:

Theorem 10. *For each h-PDA there exists an equivalent NSE grammar of polynomial size.*

Proof. From Lemmas 7, 8, and 9, from an h-PDA $\mathcal{A}$ we can obtain an NSE grammar G of polynomial size generating $L(\mathcal{A}) \setminus \{\varepsilon\}$. In case $\varepsilon \in L(\mathcal{A})$, in order to make G equivalent to $\mathcal{A}$, we add the production $S \to \varepsilon$. □

As a consequence of Theorems 6, and 10, by paying a polynomial size increase, each NSE grammar can be transformed into an equivalent one in a particular form.

Corollary 11. *Each NSE grammar is equivalent (modulo the empty word) to a grammar in Chomsky normal form of polynomial size, in which, for each production $X \to YZ$, Y is greater than X according to the order induced by the production graph.*

Proof. By Theorem 6, from each NSE grammar G we can obtain an equivalent h-PDA $\mathcal{A}$ of polynomial size which, according to Theorem 10, can be transformed into an equivalent NSE grammar G' as defined above. We can observe that if $X \overset{+}{\Longrightarrow} \alpha X\beta$ in G', then β should be empty. This implies that for each production $X \to YZ$, Y is greater than X according to the order induced by the production graph. Furthermore, unit productions, namely productions (ii), (iv) and (v), can be easily eliminated, yielding a grammar G'' of the desired form. □

4. NSE Grammars versus 1-LAs

In this section, we compare the sizes of NSE grammars and of h-PDAs with the size of equivalent 1-limited automata. We prove that for each NSE grammar there exists an equivalent 1-LA of polynomial size. As a consequence, the simulation of constant-height PDAs by 1-LAs is polynomial in size.

Concerning the converse transformation, we prove that 1-LAs can be more succinct than NSE grammars and constant-height PDAs. Actually, we prove a stronger result showing the existence of a family $(L_n)_{n>0}$ of languages such that each L_n is accepted by a 2DFA with $O(n)$ states, while each Chomsky normal form grammar or PDA accepting L_n would require an exponential size in n.

4.1. *From* NSE *grammars to* 1-LA*s*

We start from an NSE grammar $G = \langle V, \Sigma, P, S\rangle$ in the form given by Corollary 11. Thus, every derivation tree of G has a particular form which can be expressed using the notion we now introduce. For any constant j, we call *j-tree* any labeled tree satisfying:

- internal nodes are labeled by variables, while leaves are labeled by terminal symbols;
- each internal node either has exactly two children which are internal nodes, or has a unique child which is a leaf;
- the root is an internal node;
- along every branch, the number of *left turns* (*i.e.*, the number of nodes which are left child of some node) is bounded by j.

We observe that any 0-tree consists of a root labeled by some variable which has as unique child a leaf labeled by a terminal. Moreover, a j-tree is also a $(j+1)$-tree. We also point out that we do not require that j-trees are consistent with the production rules of G. However, due to the form given by Corollary 11, there exists a constant c such that any derivation tree of G is a c-tree with root label S.

We now describe how we encode a j-tree T with m leaves into a word u of length m over the alphabet $\Gamma_j = \Sigma \times V \times V \times \{0, \ldots, j\}$. The construction is illustrated in Figure 2. First, we inductively index the nodes of T according to the following rules:

- the root of T has index j;
- the left child of a node with index i has index $i - 1$;
- the right child of a node with index i has index i;
- a leaf has the same index as its parent.

In other words, the index of a node is an upper limit to the number of left turns from that node to a leaf. From now on, we fix a parameter $Y \in V$ whose meaning will be discussed later. For a leaf ℓ of the j-tree labeled by a symbol $a \in \Sigma$, we consider the symbol $\sigma_\ell = \langle a, X, Z, i\rangle \in \Gamma_j$ where (X, i) is the indexed label of the

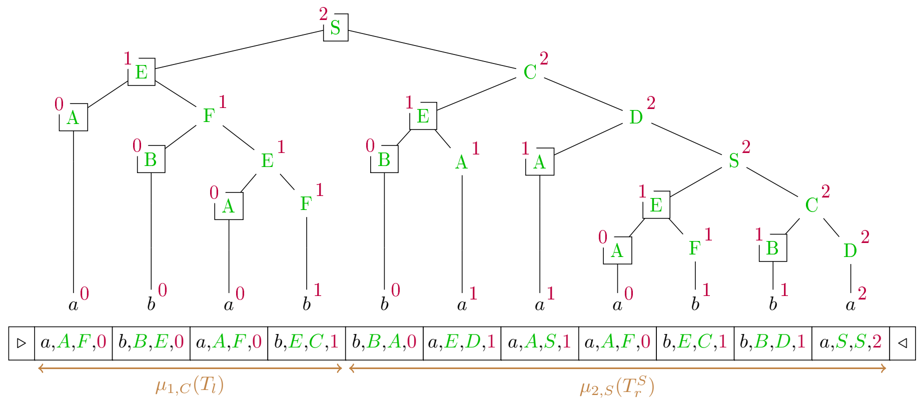

Figure 2. An example of a 2-tree and its 2-compression. Here T_l and T_r denote the left and right subtrees at depth 1 and T_r^S denotes the subtree T_r in which the label of the root has been changed to S.

closest ancestor of ℓ which is not a right child of any node (such nodes have square shape in Figure 2) and Z is the label of its right sibling if any, or equals Y otherwise, namely when that node is the root of the tree. Intuitively, the *j-compression* of the j-tree T is the word $\sigma_{\ell_1} \cdots \sigma_{\ell_m}$ where $\ell_1, \ldots, \ell_m$ are the leaves of T taken from left to right. Formally, it is inductively defined as follows. Let $Y \in V$ and T be a j-tree with root label X, for some $j \geq 0$. If T consists of its root with only one child being a leaf labeled by $a \in \Sigma$ (which is always the case when $j = 0$ by definition of j-trees), then $\mu_{j,Y}(T) = \langle a, X, Y, j\rangle$. Otherwise, $j > 0$ and T consists of a root node yielding a left subtree T_l and a right subtree T_r. Let Z be the root label of T_r. Then, denoting T_r^X the tree T_r in which the label of the root has been changed to X, $\mu_{j,Y}(T) = \mu_{j-1,Z}(T_l) \cdot \mu_{j,Y}(T_r^X)$ (see Figure 2).

Let us shortly discuss the parameter Y, which does not influence much the j-compression: it occurs only in the rightmost symbol of the word encoding the given j-tree. The parameter is meant to indicate the right sibling label of the root node, when considering left subtrees of some $(j-1)$-tree. Hence, when considering full derivation trees of G, this parameter is meaningless, and we can fix it to some arbitrarily chosen variable, say S (as in Figure 2). So-defined, the c-compression (or simply *compression*) of a derivation tree of G is a word $u \in \Gamma_c^+$ whose projection on Σ^* is the word generated by the tree. The following remark, which directly follows from the inductive definition of $\mu_{j,Y}$, is instrumental for our later proofs.

Remark 12. *If $\mu_{j,Y}(T) = v \cdot \langle a, Z, Y, j\rangle$ then $\mu_{j,Y}(T^X) = v \cdot \langle a, X, Y, j\rangle$, where T^X denotes the tree T in which the root label has been changed to X.*

Not every word over Γ_j is the j-compression of a j-tree, and not every j-tree is a derivation tree. We now introduce a property which allows us to check that a word $u \in \Gamma_j^*$ is a correct j-compression and that the tree it encodes is a derivation tree, namely it is consistent with the production rules of G. We set $\Gamma_{-1} = \emptyset$. A word $u \in \Gamma_j^+$ is a *valid j-compression*, $0 \le j \le c$, if, on the one hand, $u = w \cdot \langle a, X, Y, j\rangle$ for some $w \in \Gamma_{j-1}^*$, $a \in \Sigma$ and $X, Y \in V$, and, on the other hand, one of the following two cases holds:

(1) $w = \varepsilon$, and $X \to a$ belongs to P;
(2) there exist $v, w' \in \Gamma_{j-1}^*$, $b \in \Sigma$, and $W, Z \in V$ such that:
   (a) $w = v \cdot \langle b, W, Z, j-1\rangle w'$
   (b) $X \to WZ$ belongs to P;
   (c) $v \cdot \langle b, W, Z, j-1\rangle$ is a valid $(j-1)$-compression;
   (d) $w' \cdot \langle a, Z, Y, j\rangle$ is a valid j-compression.

In particular, valid 0-compressions are exactly the single-letter words $\langle a, X, Y, 0\rangle$ such that $X \to a \in P$. Observe that Item 2c implies $v \in \Gamma_{j-2}^*$ and therefore, the decomposition of w (Item 2a) as well as W, Z, and b are determined by the leftmost symbol of index $j-1$ of u. Notice furthermore that validity does not depend on the variable Y.

Intuitively, validity of compressions corresponds to derivation consistency of encoded trees. This is stated formally in the following lemma (remember that $G_{|X}$ denotes the grammar G in which the starting symbol has been replaced by X).

Lemma 13. *Let $j \in \{0, \dots, c\}$, $X, Y \in V$, $a \in \Sigma$, and $u \in \Gamma_{j-1}^* \cdot \langle a, X, Y, j\rangle$. Then u is a valid j-compression if and only if $u = \mu_{j,Y}(T)$ for some derivation tree T of $G_{|X}$. In particular, the projection w of u to Σ^* is generated by $G_{|X}$ through T.*

Proof. We fix a valid j-compression $u \in \Gamma_{j-1}^* \cdot \langle a, X, Y, j\rangle$. We show, by induction on the length of u, that there exists a derivation tree T of $G_{|X}$ such that $\mu_{j,Y}(T) = u$.

If $|u| = 1$, then $u = \langle a, X, Y, j\rangle$ and $X \to a \in P$ by Item 1. Hence, $u = \mu_{j,Y}(T)$ for T the derivation tree of $G_{|X}$ which derives the word a from X in one step.

Otherwise, $u = v \cdot \langle b, W, Z, j-1\rangle \cdot w' \cdot \langle a, X, Y, j\rangle$ by Item 2a, where $X \to WZ$ by Item 2b. Moreover, on the one hand $v \cdot \langle b, W, Z, j-1\rangle$ is a valid $(j-1)$-compression by Item 2c, on the other hand $w' \cdot \langle a, Z, Y, j\rangle$ is a valid j-compression by Item 2d. By induction, there exist two derivation trees T_l and T_r, respectively of $G_{|W}$ and $G_{|Z}$, such that $\mu_{j-1,Z}(T_l) = v \cdot \langle b, W, Z, j-1\rangle$ and $\mu_{j,Y}(T_r) = w' \cdot \langle a, Z, Y, j\rangle$. Since changing the label of the root node does affect only the rightmost symbol of its compression (Remark 12), we have $\mu_{j,Y}(T_r^X) = w' \cdot \langle a, X, Y, j\rangle$ where T_r^X is the tree T_r in which the root label has been changed to X. Consider the tree T consisting of a root labeled by X which has T_l as left subtree and T_r as right subtree. By the above properties, T is a derivation tree of $G_{|X}$. Moreover, $\mu_{j,Y}(T) = \mu_{j-1,Z}(T_l) \cdot \mu_{j,Y}(T_r^X) = u$.

Conversely, we fix a derivation tree T of $G_{|X}$ that we supposed to be a j-tree. We show by induction on the structure of T, that $\mu_{j,Y}(T)$ is valid for any $Y \in V$.

If T is a trivial derivation tree consisting of the root node which has as unique child a leaf labeled by a, then, $X \to a \in P$ by definition, whence $\mu_{j,Y}(T) = \langle a, X, Y, j\rangle$ is valid through Item 1 for any Y and any j.

Otherwise, the root of T has two children, yielding a left subtree T_l and a right subtree T_r. Let W and Z be the respective root labels of T_l and T_r. By definition, $u = \mu_{j,Y}(T) = \mu_{j-1,Z}(T_l) \cdot \mu_{j,Y}(T_r^X)$ where T_r^X denotes the tree T_r in which the root label has been changed to X. Since T is a derivation tree, we have $X \to WZ \in P$ (*i.e.*, Item 2b). By induction, $\mu_{j-1,Z}(T_l) = v \cdot \langle b, W, Z, j-1\rangle$ is a valid $(j-1)$-compression (Item 2c), and $\mu_{j,Y}(T_r) = w' \cdot \langle a, Z, Y, j\rangle$ is a valid j-compression (Item 2d). Finally, since modifying the root label of a tree does only affect the rightmost letter of its compressions, by Remark 12, we obtain that $u = v \cdot \langle b, W, Z, j-1\rangle w' \cdot \langle a, X, Y, j\rangle$ (*i.e.*, Item 2a). □

Lemma 13 yields a strategy to check whether a word $w \in \Sigma^+$ is generated by G, using the property that all its derivation trees are c-trees. We first guess the c-compression of a derivation tree generating w, thus obtaining a word $u \in \Gamma_c^+$ whose projection to Σ equals w. We then check that u is a valid c-compression with parameter $Y = S$. Although the initial guess makes use of nondeterminism, the verification can be performed deterministically once the guessed symbols have been fixed. We now show how to do this verification with a 2DFA. From now on, we set $\Gamma = \Gamma_c$.

In any valid j-compression of length greater than 1, some factors represent the $(j-1)$-compressions of some subtrees of the encoded tree. They are exactly the factors delimited to the left by a symbol of index greater than or equal to j or by the left endmarker (not included in the factor), and to the right by the symbol of index j corresponding to its root node (included in the factor). In other words, they are maximal factors in $\Gamma_{j-1}^* \cdot \Gamma_j$. This allows a reading head to locally detect the boundaries of such factors when scanning the j-compression. This also implies that the index of a symbol preceding a symbol of index j is always less than or equal to $j-1$. For instance, the compression illustrated in Figure 2 admits five valid 1-compression factors, namely the factor $\langle a, A, F, 0\rangle\langle b, B, E, 0\rangle\langle a, A, F, 0\rangle\langle b, E, C, 1\rangle$, the factor $\langle b, B, A, 0\rangle\langle a, E, D, 1\rangle$, the factor $\langle a, A, S, 1\rangle$, the factor $\langle a, A, F, 0\rangle\langle b, E, C, 1\rangle$, and the factor $\langle b, B, D, 1\rangle$ which respectively correspond to the five subtrees rooted in the square-shape nodes which have index 1.

We now describe how a 2DFA $\mathcal{A}$ can check that a word $u \in \Gamma_c^+$ is a valid c-compression. First of all, the device checks that u belongs to $\Gamma_{c-1}^* \cdot \langle a, S, S, c\rangle$ for some letter $a \in \Sigma$. Then, it iteratively verifies that every maximal factor of the form $\Gamma_{j-1}^* \cdot \langle a, X, Y, j\rangle$ is a valid j-compression. To this end, once the verification has been performed for the level $j-1$, it just needs to check that the letter just before $\langle a, X, Y, j\rangle$, if any, is of index at most $j-1$, and that there is consistency between letters of index $j-1$ and $\langle a, X, Y, j\rangle$ of such maximal factor, as follows: sweeping these letters $\langle a_1, W_1, Z_1, j-1\rangle, \ldots, \langle a_k, W_k, Z_k, j-1\rangle$ from left to right

Procedure 1: `CheckTree`

```
                                   /* start with the head on the left endmarker */
 1 CheckRoot
 2 for $j \leftarrow 0$ to $c$ do
 3     repeat move the head to the right until index($\sigma$) $\geq j$
 4     while index($\sigma$) $= j$ do
 5         CheckSubtree($j$)
 6         repeat move the head to the right until index($\sigma$) $\geq j$
 7     repeat move the head to the left until $\sigma = \triangleright$
 8 Accept
```

and setting $Z_0 = X$, the 2DFA sequentially checks that $Z_{i-1} \to W_i Z_i \in P$ for $i = 1, \ldots, k$, and $Z_k \to a \in P$. In other words, the device implements the above-given inductive definition of valid compressions, with the difference that it tests each subtree of level from 0 to c instead of performing recursive calls. This allows to store only one variable Z at each time.

The 2DFA $\mathcal{A}$ implements a collection of deterministic subroutines, the top-level of which is the procedure `CheckTree`. In each subroutine, σ denotes the symbol currently scanned by the head, which is automatically updated at each head move. Moreover, the special instruction REJECT causes the whole computation to halt and reject. We furthermore use the four natural projections over Γ: for a symbol $\sigma = \langle a, X, Y, j \rangle \in \Gamma$, we set `letter`$(\sigma) = a$, `varLeft`$(\sigma) = X$, `varRight`$(\sigma) = Y$, and `index`$(\sigma) = j$. We fix the convention `index`$(\triangleright) =$ `index`$(\triangleleft) = c + 1$.

Procedure 2: `CheckRoot`

```
 9 repeat move the head to the right until $\sigma = \triangleleft$
10 move the head to the left
11 if varLeft($\sigma$) $\neq S$ or varRight($\sigma$) $\neq S$ or index($\sigma$) $\neq c$ then Reject
12 while $\sigma \neq \triangleright$ do
13     move the head to the left
14     if index($\sigma$) $= c$ then Reject
```

As initial phase, the subroutine `CheckRoot` checks that the input word belongs to $\Gamma_{c-1}^{*} \cdot \langle a, S, S, c \rangle$ for some letter $a \in \Sigma$. Then, $\mathcal{A}$ checks the validity of each compression of each level from 0 to c (Lines 2 to 7). This verification uses the procedure `CheckSubtree` (Line 5).

This latter subroutine is the direct implementation of the inductive definition of valid compressions, where the recursive call to incremented level (Item 2c) is omitted (the validity of these sub-compressions have already been checked by previous call to `CheckSubtree`). It uses the subroutine `SelectNext` to locate the leftmost symbol of index $j - 1$ in the factor under consideration, if any, or to check if the factor has length 1, otherwise, thus checking Item 2a (or, partially, Item 1). Items 1

Procedure 3: `CheckSubtree`(j)

```
                    /* start with the head scanning a symbol of index $j$ */
15 $C \leftarrow$ varLeft($\sigma$)
16 repeat move the head to the left until index($\sigma$) $\geq j$
17 SelectNext($j-1$)
18 while index($\sigma$) $\neq j$ do
19     if $C \to$ varLeft($\sigma$)varRight($\sigma$) $\notin P$ then REJECT
20     $C \leftarrow$ varRight($\sigma$)
21     SelectNext($j-1$)
22 if $C \to$ letter($\sigma$) $\notin P$ then REJECT
```

Procedure 4: `SelectNext`(j)

```
23 move the head to the right
24 if index($\sigma$) $\neq j+1$ then
25     while index($\sigma$) $< j$ do move the head to the right
26     if index($\sigma$) $\neq j$ then REJECT
```

and 2b correspond to Lines 22 and 19, respectively, where C contains the variable Z (Line 20), the variable label of the root of the subtree which is initially set to X (Line 15), thus allowing to verify Item 2d (Lines 18 to 21).

To summarize, we obtained the following result.

Lemma 14. *The language of valid compressions of derivation trees of G is recognized by a* 2DFA *which uses $O(c \cdot \#V)$ states.*

Proof. The construction of such a 2DFA $\mathcal{A}$ has been described above. We now estimate its size. The only memory used in the procedure `CheckTree`, is an index $j \in \{0, \dots, c\}$, to which both the subroutines `SelectNext` and `CheckSubtree` have read-only access. The subroutines `CheckRoot` and `SelectNext` use no additional memory. The procedure `CheckSubtree` stores one variable, namely C, which ranges over V. Hence, the number of states of $\mathcal{A}$ is linear in $c \cdot \#V$. □

We are now ready to state our main result.

Theorem 15. *For every* NSE *grammar G, there exist a 1-state letter-to-letter nondeterministic transducer $\mathcal{T}$ and a* 2DFA *$\mathcal{A}$ of polynomial size such that a word w is generated by G if and only if $\mathcal{A}$ accepts an image u of w by $\mathcal{T}$. As a consequence, G can be transformed into a* 1-LA *of polynomial size.*

Proof. From an NSE grammar G, we obtain an NSE grammar G' over Σ of polynomial size in the form given by Corollary 11, such that $L(G') = L(G) \setminus \{\varepsilon\}$. The transducer $\mathcal{T}$ replaces each letter $a \in \Sigma$ with a symbol $\langle a, X, Y, j\rangle$ for some variables X and Y of G' and some index $j \leq \#V$. Finally, we build $\mathcal{A}$, using Lemma 14, which recognizes an output of $\mathcal{T}$, if and only if its pre-image was generated by G', by Lemma 13. In case $\varepsilon \in L(G)$, we modify $\mathcal{A}$ in order to accept ε.

The composition of $\mathcal{T}$ and $\mathcal{A}$ yields a 1-LA which first performs a left-to-right traversal during which each tape cell is nondeterministically rewritten according to $\mathcal{T}$, and then deterministically simulates $\mathcal{A}$. □

4.2. *From* 1-LA*s to* NSE *grammars: An exponential gap*

In this section, we exhibit an infinite family $(L_n)_{n\geq 0}$ of languages over the alphabet $\{0,1\}$, such that each L_n is recognized by a 1-LA with size polynomial in n, but requires an exponential size in order to be recognized by any h-PDA or NSE grammars. We can actually prove a stronger result, since each L_n is recognized by a 2DFA (and even by a *rotating* deterministic automaton, in which all passes over the input are from left to right [9]) of linear size, while any grammar in Chomsky normal form generating L_n requires an exponential number of variables. As a consequence, every PDA recognizing L_n requires an exponential size. The proof of this lower bound is obtained by using the *interchange lemma for context-free languages* [14]:

Lemma 16. *Let G be a context-free grammar in Chomsky normal form, with c variables, and let L be the language it generates. For all integers N, m, with $2 \leq m \leq N$, and all subsets R of $L \cap \Sigma^N$, there exists a subset $Z \subseteq R$ with $Z = \{z_1, z_2, \ldots, z_k\}$ such that $k \geq \frac{\#R}{cN^2}$, 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 *with* $1 \leq i, j \leq k$.

Theorem 17. *For each $n > 0$, let L_n be the language $\{uuu \mid u \in \{0,1\}^n\}$. Then:*

- *L_n is accepted by a* 2DFA *of size $O(n)$;*
- *each context-free grammar in Chomsky normal form needs exponentially many variables in n to generate L_n;*
- *the size of any* PDA *accepting L_n is at least exponential in n.*

Proof. A 2DFA $\mathcal{A}$ with $O(n)$ states can accept L_n as follows. First $\mathcal{A}$ traverses the whole input tape, in order to verify that the input length is $3n$. Then $\mathcal{A}$, by moving the head back and forth, verifies that all two symbols at distance n are equal. It is not difficult to observe that $\mathcal{A}$ can be implemented using $O(n)$ states.

To prove that each context-free grammar generating L_n requires an exponential number of variables, we observe that given $u, u' \in \{0,1\}^n$, if we decompose the strings $z = uuu$ and $z' = u'u'u' \in L_n$ as $z = wxy$ and $z' = w'x'y'$, with $|w| = |w'|$, $|y| = |y'|$, $n < |x| = |x'| \leq 2n$, then $|wy| \geq n$, thus implying that $u = u_l u_r$ where u_l is a prefix of w and u_r is a suffix of y. If $u \neq u'$ then $x \neq x'$ and the string $wx'y$

cannot belong to L_n. Applying Lemma 16, with $N = 3n$, $R = L_n$ and $m = 2n$, from the previous argument it follows that the resulting set Z cannot contain more than one string. Hence, we conclude that each context-free grammar in Chomsky normal form generating L_n should have at least $\frac{2^n}{9n^2}$ variables.

Finally, since each PDA can be converted into an equivalent context-free grammar in Chomsky normal form with a polynomial number of variables, *e.g.*, [19, Theorem 8] we conclude that the size of any PDA accepting L_n is at least exponential in n. □

Corollary 18. *The size cost of the conversion of* 1-LA*s into* NSE *grammars and* h-PDA*s is exponential.*

Proof. The lower bound derives from Theorem 17. For the upper bound, in [16] it was proved that each 1-LA can be transformed into a 1NFA of exponential size from which, by a standard construction, we can obtain a regular (and, so, NSE) grammar, without increasing the size asymptotically. □

In [2], the question of the cost of the conversion of deterministic h-PDAs into 1NFAs was raised. To this regard, we observe that the language $(a^{2^n})^*$ is accepted by a deterministic h-PDA of size polynomial in n for large enough h (see, *e.g.*, [15]) but, by a standard pumping argument, it requires at least 2^n states to be accepted by 1NFAs. Actually, as a consequence of state lower bound presented in [13], 2^n states are also necessary to accept it on each 2NFA. Considering Theorem 17, we can conclude that both simulations from two-way automata to h-PDAs and from h-PDAs to two-way automata cost at least exponential.

5. Deterministic h-PDAs versus Deterministic 1-LAs

From the results in Sections 3 and 4, it turns out that there is a polynomial-size conversion of h-PDAs into 1-LAs. Here, we consider the *deterministic case.* We present a polynomial size conversion of deterministic h-PDAs into deterministic 1-LAs.

We now recall the definition of *deterministic* h-PDA*s*, according to [3]. Let $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ be an h-PDA. Let Q_Σ, Q_+, and Q_- be the sets of states p such that there exists a transition (p, op, q) with op belonging to Σ, to $\{+\}\Gamma$, or to $\{-\}\Gamma$, respectively. $\mathcal{A}$ is *deterministic* if it satisfies the following properties:

(1) it does not allow transitions of the form (p, ε, q);
(2) Q_Σ, Q_+, and Q_- form a partition of Q;
(3) if (p, op, q) and (p, op', q') are distinct transitions, then $p \in Q_\Sigma \cup Q_-$ and op and op' are distinct elements of Σ if $p \in Q_\Sigma$, or of $\{-\}\Gamma$ if $p \in Q_-$ (notice that this implies that there exists exactly one outgoing transition from each state in Q_+);
(4) $F \subseteq Q_\Sigma$.

Item 2 ensures that the action to perform is fully determined by the current state. Based on this, Item 3 states that for any configuration there exists at most one outgoing transition, while Item 4 constrains acceptance, as explained in the following. As for nondeterministic h-PDA, the machine accepts the input word if it reaches an accepting state after having read all the input symbols. However, in order to avoid exiting an accepting configuration, Item 4 requires that the machine halts by waiting for a next symbol to scan. We point out that, in the definition of deterministic h-PDA given in [3], states with no outgoing transitions are present and transitions of the form (p, ε, q) are allowed under the restriction that from each state at most one such transition is allowed, which should further be the unique transition outgoing that state. With classical transformations, states without any outgoing transition as well as ε-moves can be avoided without increasing the size of the automaton, while preserving determinism. Hence Item 1 as well as the statement of Item 2 which is stronger than those given in [3] can be ensured without loss of generality.

Moreover, given a deterministic h-PDA, it is always possible to obtain an equivalent deterministic h-PDA of at most the same size, in which there are no push transitions entering a state of Q_-. Indeed, in any computation, if a pop $(p_-, -Y, q)$ immediately follows a push $(p_+, +X, p_-)$ then $X = Y$. Thus, in presence of a transition $(p_+, +X, p_-)$ with $p_- \in Q_-$, we can eliminate the state p_+ and its outgoing transition after modifying the machine as follows: if there exists q such that $(p_-, -X, q) \in \delta$ then we replace each transition $(p, \mathtt{op}, p_+)$ with the transition $(p, \mathtt{op}, q)$ and, furthermore, q becomes the initial state whenever p_+ is initial.[b] By iteratively applying this transformation, an equivalent deterministic h-PDA without any transition in $Q_+ \times \{+\}\Gamma \times Q_-$ is obtained. Finally, we can assume without loss of generality that deterministic h-PDAs do not contain any loop composed by push transitions only. Indeed, without increasing the size of the model, these loops can be eliminated since when entering such a loop, the machine surely halts and rejects after at most h steps as the pushdown height will exceed its bound.

From now on, we assume without loss of generality that deterministic h-PDAs have neither a transition in $Q_+ \times \{+\}\Gamma \times Q_-$ nor a loop with push transitions only.

Let $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ be a deterministic h-PDA, with Q_Σ, Q_+, Q_- defined as above. By determinism, a state from Q_+ has a unique outgoing transition. Hence, until leaving Q_+, the successive transitions from a state $p \in Q_+$ generate a finite sequence of push transitions, which is fully determined from p. For ease of presentation we will use the following functions that are defined for each state from Q_+:

- $\eta : Q_+ \to Q_\Sigma$ maps each state in Q_+ to the first reachable state not belonging to Q_+ after a sequence of push transitions, which exists and belongs to Q_Σ by assumption.

[b]If p_+ is the initial state but there exists no q such that $(p_-, -X, q) \in \delta$, then the complete device recognizes the empty language, and can thus be replaced by a simpler single-state one.

Procedure 5: `simulatePush` is called when a push has to be simulated, *i.e.*, when the variable simulatedState contains a state $q \in Q_+$

```
27 move the head rightward until reaching the leftmost symbol $\sigma \in \Sigma \cup \{\triangleleft\}$
28 if $\sigma = \triangleleft$ then
29     if $\eta(q) \in F$ then ACCEPT else REJECT
30 detect move $(\eta(q), \sigma, r)$ to be simulated
31 write($\langle q, \mathsf{height}\rangle$)
32 if $\mathsf{height} + \ell(q) > h$ then REJECT
33 $\mathsf{height} \leftarrow \mathsf{height} + \ell(q)$
34 $\mathsf{simulatedState} \leftarrow r$
```

- $\ell : Q_+ \to \{1, \ldots, h\}$ maps each state $p_+ \in Q_+$ to the maximum number of consecutive push transitions that can be performed starting from p_+.
- $\omega : Q_+ \to \Gamma^{\leq h}$ maps each state $p_+ \in Q_+$ to the string that can be pushed during a maximal sequence of consecutive push transitions starting from p_+. Notice that the length of such a string is given by $\ell(p_+) \leq h$.

For instance, consider the maximal sequence of consecutive push transitions

$$(p_0, +X_1, p_1), (p_1, +X_2, p_2), \ldots, (p_{n-1}, +X_n, p_n),$$

where $p_n \in Q_\Sigma$ and, for each i such that $0 \leq i < n \leq h$, $p_i \in Q_+$ and $X_{i+1} \in \Gamma$. Then, $\eta(p_0) = p_n$, $\ell(p_0) = n$, and $\omega(p_0) = X_1 \cdots X_n$. These functions can be always computed by analyzing the transition function δ.

Let us now show how the simulation works.

Theorem 19. *Each deterministic h-PDA admits an equivalent deterministic 1-LA of polynomial size.*

Proof. Let $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ be a deterministic h-PDA where, using the above notations, $Q = Q_\Sigma \cup Q_+ \cup Q_-$. Hence, in every accepting computation, a sequence of push transitions ends by entering a state from Q_Σ from which $\mathcal{A}$ scans the next input symbol, or accepts. We define a simulating deterministic 1-LA $\mathcal{A}' = \langle Q', \Sigma, \Gamma', \delta', q_0', F'\rangle$.

The main idea of the simulation is that at each step of its computation $\mathcal{A}'$ is able to recover the content $w \in \Gamma^{\leq h}$ of the pushdown store of the simulated machine without storing w in its finite control, but from the information written on the already visited tape cells. The simulation, detailed below, is described in Procedures 5, 6, and 7.

More precisely, we assume that $\mathcal{A}'$ stores in its finite control the current pushdown height, in a variable height with maximum value h that is initially set to 0, and the current state, in a variable simulatedState initially containing the initial state

Procedure 6: `simulateRead` is called when a read has to be simulated, *i.e.*, when the variable simulatedState contains a state $q \in Q_\Sigma$

```
35 move the head rightward until reaching the leftmost symbol $\sigma \in \Sigma \cup \{\lhd\}$
36 if $\sigma = \lhd$ then
37     if $\eta(q) \in F$ then ACCEPT else REJECT
38 detect move $(q, \sigma, r)$ to be simulated
39 write($\sharp$)
40 simulatedState ← $r$
```

of $\mathcal{A}$, of the configuration reached in the simulated computation of $\mathcal{A}$. In order to simulate a maximal sequence of push transitions

$$(p_0, +X_1, p_1), (p_1, +X_2, p_2), \ldots, (p_{n-1}, +X_n, p_n)$$

where $p_0, \ldots, p_{n-1} \in Q_+$, $X_1, \ldots, X_n \in \Gamma$, and $p_n \in Q_\Sigma$, $\mathcal{A}'$ moves its head rightward to the leftmost tape cell which has not been visited so far (*i.e.*, which has not been rewritten) and performs, in one step, the following actions:

- it scans the input symbol $a \in \Sigma$, and determines the transition (p_n, a, q) (Line 30);
- it overwrites the cell content with the pair (p_0, i) where $i \leq h - \ell(p_0)$ is the current pushdown height, stored in its finite control (Line 31);
- if the height of the stack after pushing $\ell(p_0)$ symbols does not exceed h (Line 32), it updates the pushdown height component to $i + \ell(p_0) \leq h$ (Line 33);
- it updates the state component to q (Line 34).

Hence, $\mathcal{A}'$ does not only simulate the sequence of push, but also the successive scan step. When all the cells have already been visited, *i.e.*, when the head of $\mathcal{A}'$ has reached the right endmarker, then it halts and accepts if and only if $p_n \in F$ (Line 29).

When the next transition to be simulated has to scan the input (not just after a sequence of push), $\mathcal{A}'$ proceeds similarly (Procedure 6), but simply updates the variable simulatedState accordingly (Line 40) without modifying the value of height, and rewrites the corresponding cell content with the special symbol $\sharp$ (Line 39).

When $\mathcal{A}'$ has to access the content of the pushdown, namely when a pop transition has to be simulated, the simulating machine looks for the last sequence of simulated push transitions whose first symbol was pushed from a level lower than or equal to the current pushdown height. This can be done by scanning leftward the tape, until reaching the rightmost cell containing a pair (p_+, i), with i less than or equal to the value of height (Line 41). At this point, by using i and $\omega(p_+)$, $\mathcal{A}'$ recovers the symbol at level equal to the height stored in the finite control and detects a suitable pop transition to be simulated (Line 42). If such a transition exists then $\mathcal{A}'$ updates both its state and pushdown height components according to this

Procedure 7: `simulatePop` is called when a pop has to be simulated, *i.e.*, when the variable simulatedState contains a state $q \in Q_-$.

```
41 move the head leftward until reaching a symbol $(p_+, i)$ with $i \leq \mathsf{height}$
42 detect move $(q, -X, r)$ to be simulated where $X$ is the $(\mathsf{height} - i)$-th symbol
     of $\omega(p_+)$
43 $\mathsf{height} \leftarrow \mathsf{height} - 1$
44 $\mathsf{simulatedState} \leftarrow r$
```

transition (Lines 44, and 43), and continues the simulation. Notice that $\mathcal{A}'$ does not need to recover the original head position, *i.e.*, the head position of $\mathcal{A}$ in the simulated computation, until it enters a state from $Q_\Sigma \cup Q_+$. When this happens, $\mathcal{A}'$ proceed as explained previously.

If no move in δ can be simulated because no suitable transition is defined (Lines 30, 38, and 42), then the simulating machine halts and rejects.

We now evaluate the size of the simulating deterministic 1-LA $\mathcal{A}'$. Notice that the quantities computed by η, ℓ, and ω do not depend on the input, whence are precomputed and hardly encoded in the transition table of $\mathcal{A}'$. The working alphabet of $\mathcal{A}'$ is included in $Q_+ \times \{0, \dots, h-1\} \cup \{\sharp\}$, while the finite control stores two variables: one state in Q and one pushdown height in $\{0, \dots, h\}$. Hence, the size of $\mathcal{A}'$ is polynomial in $\#Q$ and h. We point out that it does not depend on the size of the pushdown alphabet of $\mathcal{A}$. □

Bibliography

[1] M. Anselmo, D. Giammarresi and S. Varricchio, Finite automata and non-self-embedding grammars, *CIAA 2002*, *LNCS* **2608** (2002), pp. 47–56.

[2] Z. Bednárová, V. Geffert, C. Mereghetti and B. Palano, Removing nondeterminism in constant height pushdown automata, *Inf. Comput.* **237** (2014) 257–267.

[3] Z. Bednárová, V. Geffert, C. Mereghetti and B. Palano, The size-cost of Boolean operations on constant height deterministic pushdown automata, *Theoretical Computer Science* **449** (2012) 23–36.

[4] N. Chomsky, On certain formal properties of grammars, *Information and Control* **2**(2) (1959) 137–167.

[5] N. Chomsky, A note on phrase structure grammars, *Information and Control* **2**(4) (1959) 393–395.

[6] V. Geffert, C. Mereghetti and B. Palano, More concise representation of regular languages by automata and regular expressions, *Inf. Comput.* **208**(4) (2010) 385–394.

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

[8] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to automata theory, languages, and computation - (2. ed.)* (Addison-Wesley-Longman, 2001).

[9] C. A. Kapoutsis, R. Královic and T. Mömke, Size complexity of rotating and sweeping automata, *J. Comput. Syst. Sci.* **78**(2) (2012) 537–558.

[10] A. Kelemenová, Complexity of normal form grammars, *Theor. Comput. Sci.* **28** (1984) 299–314.

[11] M. Kutrib, G. Pighizzini and M. Wendlandt, Descriptional complexity of limited automata, *Information and Computation* **259** (2018) 259–276.
[12] M. Kutrib and M. Wendlandt, On simulation cost of unary limited automata, *DCFS 2015*, *LNCS* **9118**, (Springer, 2015), pp. 153–164.
[13] C. Mereghetti and G. Pighizzini, Two-way automata simulations and unary languages, *Journal of Automata, Languages and Combinatorics* **5**(3) (2000) 287–300.
[14] W. F. Ogden, R. J. Ross and K. Winklmann, An "interchange lemma" for context-free languages, *SIAM J. Comput.* **14**(2) (1985) 410–415.
[15] G. Pighizzini, Deterministic pushdown automata and unary languages, *Int. J. Found. Comput. Sci.* **20**(4) (2009) 629–645.
[16] G. Pighizzini and A. Pisoni, Limited automata and regular languages, *Int. J. Found. Comput. Sci.* **25**(7) (2014) 897–916.
[17] G. Pighizzini and L. Prigioniero, Non-self-embedding grammars and descriptional complexity, *Ninth Workshop on Non-Classical Models of Automata and Applications, NCMA 2017, Prague, Czech Republic, August 17-18, 2017*, (Österreichische Computer Gesellschaft, 2017), pp. 197–209.
[18] G. Pighizzini and L. Prigioniero, Limited automata and unary languages, *Inf. Comput.* **266** (2019) 60–74.
[19] G. Pighizzini, J. Shallit and M. Wang, Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds, *J. Comput. Syst. Sci.* **65**(2) (2002) 393–414.
[20] W. J. Sakoda and M. Sipser, Nondeterminism and the size of two way finite automata, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, (ACM, 1978), pp. 275–286.
[21] K. W. Wagner and G. Wechsung, *Computational Complexity* (D. Reidel Publishing Company, Dordrecht, 1986).