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

Bruno Guillon, Giovanni Pighizzini, and Luca Prigioniero

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

Abstract. 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 between non-self-embedding grammars and deterministic finite automata is known. The same size gap is also known from constant height pushdown automata and 1-limited automata to equivalent 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. However, they can be exponentially larger than 1-limited automata.

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 in 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.

To emphasize this capability, in one of his pioneering papers, Chomsky investigated the *self-embedding* property [4]: a context-free grammar is self-embedding if it contains a useful variable A which, in some sentential form, is able to reproduce itself surrounded by two nonempty strings α and β, in symbols $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$. 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 equivalent finite automata have been investigated in [1] and [13]. 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 and these models.

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 [5]. Furthermore, in [2] the authors showed the interesting result that also the gap from nondeterministic and deterministic constant height pushdown automata is double exponential. 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 of the paper, 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 nondetermin-

stic 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 [6]. Furthermore, as shown in [16, Thm. 12.1], 1-limited automata are equivalent to finite automata. This equivalence has been investigated from the descriptional complexity point of view in [11], 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 grammar into an equivalent 1-limited automaton of polynomial 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 automaton with $O(n)$ states, which requires exponentially many states to be accepted even by an unresticted pushdown automaton. From the cost of the conversion of 1-limited automata into equivalent 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.

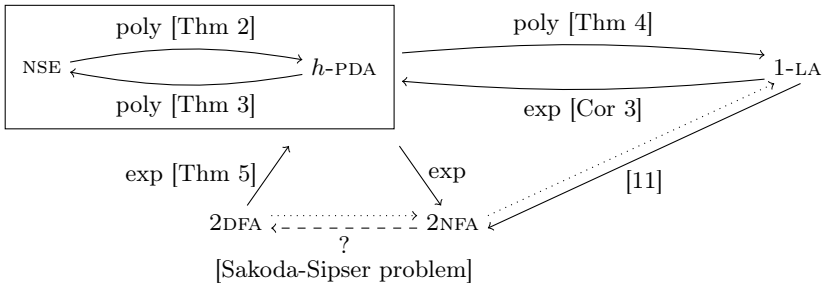

Fig. 1. Some bounds discussed in the paper. Dotted arrows denote trivial relationships, while the dashed arrow indicates the famous question by Sakoda and Sipser.

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.*, [7]. The empty word is denoted by ε. Given a word $u \in \Sigma^*$, we denote by $|u|$ its length. For every $h \geq 0$, Σ^h denotes the set of words of length h over the alphabet Σ, while Σ^+ denotes the set of all nonempty words over Σ. For two-way devices operating on a tape (*e.g.*, 2NFA), 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.

Context-free Grammars

Given a *context-free grammar* (CFG, for short) $G = \langle V, \Sigma, P, S \rangle$, we denote by $L(G)$ the language generated by G. The relations $\Rightarrow$ and $\stackrel{\star}{\Longrightarrow}$ are defined in the usual way. 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 $\alpha, \beta \in (V \cup \Sigma)^*$.

Definition 1 (NSE grammars). *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 $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$. 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 [3,4]. As shown in [1], given a grammar G it is possible to decide in polynomial time whether or not it is NSE.

Pushdown automata

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 [5,2], 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 (Pushdown automata of constant height). *A pushdown automaton of height h (h-PDA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F, h \rangle$ 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, $h \in \mathbb{N}$ is the pushdown height bound, and $\delta \subseteq Q \times (\{\varepsilon\} \cup \Sigma \cup \{-,+\}\Gamma)$ is the transition relation with the following meaning:*

(i) $(p, \varepsilon, q) \in \delta$*: $\mathcal{A}$ 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);*

(ii) $(p, a, q) \in \delta$*: $\mathcal{A}$ can reach the state q from the state p by reading the symbol a from the input but without using the pushdown store;*

(iii) $(p, -X, q) \in \delta$*: 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 the symbol X, not using the input tape;*

(iv) $(p, +X, q) \in \delta$*: 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 the symbol X on top of the pushdown store, without using the input tape.*

The model *accepts* an input word $w \in \Sigma^*$, if, starting from 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 NFAs, since they can never push symbols.

1-limited automata

One-limited automata (1-LAs, for short) extend 2-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 NFAs and DFAs are exponential and double exponential, respectively [11], while 2NFAs can be exponentially larger than deterministic 1-LAs even in the unary case [12].

Definition 3 (1-limited automata). *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* $\Sigma \subset \Gamma$*, and* $\delta : Q \times \Gamma_{\triangleright\triangleleft} \to 2^{Q \times \Gamma_{\triangleright\triangleleft} \times \{-1,+1\}}$ *is the transition function, where* $\Gamma_{\triangleright\triangleleft}$ *denotes the set* $\Gamma \cup \{\triangleright, \triangleleft\}$ *with* $\triangleright, \triangleleft \notin \Gamma$ *the left and the right endmarkers.*

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 visits only. Formally, symbols from Σ shall be replaced by symbols from $\Gamma \setminus \Sigma$, while symbols from $\Gamma_{\triangleright\triangleleft} \setminus \Sigma$ are never overwritten. In particular, at any time, both special symbols $\triangleright$ and $\triangleleft$ occur exactly once on the tape and exactly 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, the computation eventually reaches the right endmarker in an accepting state. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Size of models

For each model under consideration, we evaluate its size as the total number of symbols used to define it. Hence, as a measure for the *size of a grammar* $G = \langle V, \Sigma, P, S \rangle$, we consider the total number of symbols used to specify it, defined as $\mathrm{Symb}(G) = \sum_{(A \to \alpha) \in P} (2 + |\alpha|)$ (*cf.* [8]). The *size of an* h-PDA $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F, h \rangle$ is given by some polynomial in $\#Q$, $\#\Gamma$ and h. Finally, the *size of a* 1-LA $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is given by a polynomial in $\#Q$ and $\#\Gamma$.

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 a string in the language $L\left(G_{2|A}\right)$ 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 property of being non-self-embedding [1]. The decomposition

presented in the following result was obtained in [1, Theorem 2], while its size is discussed in [13].

Theorem 1. *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 $A \stackrel{+}{\Longrightarrow} A\gamma$. 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 SCC of the production graph of the given NSE grammar by one equivalent right-linear grammar, as shown in the following lemma:

Lemma 1. *Each* NSE *grammar can be converted into an equivalent* NSE *grammar of polynomial size which can be expressed as a 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 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, we replace each left-linear G_i by *a family* of right-linear grammars G'_{iA}, with $A \in V_i$, where G'_{iA} is equivalent to the grammar $G_{i|A} = \langle V_i, \Sigma_i, P_i, A \rangle$. It can be verified that the size of the resulting grammar G' is at most quadratic in the size of G. □

Theorem 2. *Let $G = \langle V, \Sigma, P, S \rangle$ be an* NSE *grammar such that $G = G_1 \oplus G_2 \oplus \cdots \oplus G_g$, where each G_i is right-linear. Then there is an equivalent* PDA *of polynomial size such that the height of the pushdown store is bounded by a polynomial in g.*

Proof. First of all, as in the construction presented in [1], we show that, for $i = 1, \ldots, g$, if a variable $A \in V_i$ derives a string $x\alpha$ by a leftmost derivation, *i.e.*, $A \underset{lm}{\overset{\star}{\Longrightarrow}} x\alpha$, 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 $|\alpha| \leq K(g - i) + 1$, where K is the maximum length of production right-hand sides. We are going to prove this claim by induction on $g - i$.

For $g - i = 0$, the only possible derivations from a variable $A \in V_g$ have the forms $A \overset{\star}{\Longrightarrow} x$ and $A \overset{\star}{\Longrightarrow} xB$, with $x \in \Sigma^*$ and $B \in V_g$. Then the claim immediately follows.

For $g - i > 0$, the claim is trivial in case $\alpha = \varepsilon$. Otherwise, suppose that the first production used in the derivation under consideration is $A \to X_1 X_2 \cdots X_s$, *i.e.*,

$$A \Longrightarrow X_1 X_2 \cdots X_s \underset{lm}{\overset{\star}{\Longrightarrow}} \alpha_1 \alpha_2 \cdots \alpha_s = x\alpha$$

where $X_k \in \Sigma \cup \bigcup_{j=i}^{g} V_j$ and $X_k \underset{lm}{\overset{\star}{\Longrightarrow}} \alpha_k$, $k = 1, \ldots, s$. Let ℓ, $1 \leq \ell \leq s$, 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 $\alpha = \alpha' X_{\ell+1}\cdots X_s$.

If $X_\ell \in V_j$, with $j > i$, then, by induction hypothesis, $|\alpha'| \leq K(g-j)+1$. Hence $|\alpha| \leq K(g-j)+1+s-\ell < K(g-i)+1$ due to the fact that $s-\ell < K$.

Since G_i is right-linear, the case $X_\ell \in V_i$ could occur only when $\ell = s$, thus implying that $X_1X_2\cdots X_{s-1}$ is a terminal string. The derivation under consideration can be decomposed as $A \underset{lm}{\overset{\star}{\Longrightarrow}} y_1A_1 \underset{lm}{\overset{\star}{\Longrightarrow}} y_1x_1\alpha$, where $A_1 \in V_i$. Now, we can repeat the previous proof on the subderivation $A_1 \underset{lm}{\overset{\star}{\Longrightarrow}} x_1\alpha$, either reaching the previous case and, hence, proving that $|\alpha| \leq K(g-i)+1$, or returning to this case with a further subderivation $A_2 \underset{lm}{\overset{\star}{\Longrightarrow}} x_2\alpha$. This argument can be iterated. At each step, a subderivation of the derivation at the previous step is considered. So, in a finite number of steps we will reach the previous case obtaining the desired conclusion.

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.*, [7]). After consuming an input prefix y, the pushdown store of M can contain any string $z\alpha$ such that $S \underset{lm}{\overset{\star}{\Longrightarrow}} yz\alpha$, 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 $|z| \leq K$ and, according to the first part of the proof $|\alpha| \leq K(g-1)+1$, 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-PDAs to NSE grammars

Let $\mathcal{A}$ be an h-PDA. We first show that, modulo acceptance of the empty word, with only a polynomial increase in the size we can transform it in a special form.

Lemma 2. *For each h-PDA $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F, h\rangle$ there exists an h-PDA $\mathcal{A}' = \langle Q', \Sigma, \Gamma', \delta', q_-, \{q_+\}, h\rangle$ and a mapping $\widetilde{h} : \Gamma' \to \{1, \ldots, h\}$ such that:*

- $L(\mathcal{A}') = L(\mathcal{A}) \setminus \{\varepsilon\}$;
- *$\mathcal{A}'$ has polynomial size with respect to $\mathcal{A}$;*
- *every nontrivial computation path of $\mathcal{A}'$ starting and ending with the same pushdown content and never popping off symbols under this level, consumes one input letter,* i.e.*, during such a computation, at least one transition from $Q \times \Sigma \times Q$ is performed;*
- *$\mathcal{A}'$ accepts with empty pushdown;*
- *each symbol $X \in \Gamma'$ can appear on the pushdown only at height $\widetilde{h}(X)$.*

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 $Q_\star \times \{0, \ldots, h\}$, 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 $\Gamma' = \Gamma \times \{1, \ldots, h\}$ 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

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

The restriction to states from $Q_\star \times \{i, \dots, h\}$ ensures that the considered computation paths can never pop off symbols under their initial level, while the restriction on the set of actions forbid 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 $(p, -X, q)$ 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 starts and ends with same pushdown content and without popping off symbols under the corresponding level, are necessarily sequences of ε-moves. Hence, each set $E_{(p,i)}$, which is kept unchanged by the above transformation, is now equal to $\{(q,i) \mid (p,i) \vdash^* (q,i) \text{ using only } \varepsilon\text{-moves}\}$.

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 $((p,0), \psi, (r,i))$ with $\psi \neq \varepsilon$, we create a transition $((q_-,0), \psi, (r,i))$. We then remove every ε-moves 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 $((p,0), \psi, (r,i))$ with $\psi \neq \varepsilon$ 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, ψ, q) with $\psi \neq \varepsilon$ and each $r \in E'_q$, we create the transition (p, ψ, r). 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. □

Let $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_-, \{q_+\}, h\rangle$ be an h-PDA in the form of Lemma 2. In particular, $\mathcal{A}$ does not accept the empty word. 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 $[qAp]$ and $\langle qAp\rangle$, for $q, p \in Q$, $A \in \Gamma$. The set P consists of the following productions:

(i) $\langle pAq\rangle \to a$, for $(p, a, q) \in \delta$

(ii) $\langle pAq\rangle \to [p'Xq']$, for $(p, +X, p'), (q', -X, q) \in \delta$, *i.e.*, push and pop of a same symbol X

(iii) $[pAq] \to \langle pAr\rangle[rAq]$, for $p, q, r \in Q$, $A \in \Gamma$

(iv) $[pAq] \to \langle pAq\rangle$, for $p, q \in Q$, $A \in \Gamma$

(v) $S \to [q_-\bot q_+]$, where $\bot \notin \Gamma$ is a new symbol denoting the "missed top" in the empty pushdown store.

The above definition is derived from the classical construction of CFGs from PDAs, introducing two types of triples in order to obtain an NSE grammar. (A simpler construction, as that in [14], would produce a self-embedding grammar.) More precisely, we can prove the following:

Lemma 3. *For each $x \in \Sigma^+$, $p, q \in Q$, $A \in \Gamma$, $[pAq] \stackrel{\star}{\Longrightarrow} x$ if and only if there exists a computation path $\mathcal{C}$ of $\mathcal{A}$ verifying 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 A;

(2) along $\mathcal{C}$ the pushdown is never popped off under its level at the beginning of $\mathcal{C}$;

(3) the input factor consumed along $\mathcal{C}$ is x.

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

(2′) in all configurations of the computation path $\mathcal{C}$ *others than the first and the last one, the pushdown is higher than in the first and in the last configuration of* $\mathcal{C}$*.*

Proof. First of all, we are going to prove that for all x, p, q, A as in the statement of the lemma, $[pAq] \stackrel{\star}{\Longrightarrow} x$ implies (1), (2), and (3), while $\langle pAq\rangle \stackrel{\star}{\Longrightarrow} x$ implies (1), (2′), and (3). We proceed by induction on the length $k \geq 1$ of the derivation $[pAq] \stackrel{k}{\Longrightarrow} x$ or $\langle pAq\rangle \stackrel{k}{\Longrightarrow} x$.

For $k = 1$, there are no derivations $[pAq] \stackrel{1}{\Longrightarrow} x$, while $\langle pAq\rangle \stackrel{1}{\Longrightarrow} x$ 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 $[pAq] \stackrel{k}{\Longrightarrow} x$ is either of the form (iii) or of the form (iv). In the first case we have $[pAq] \Longrightarrow \langle pAr\rangle[rAq] \stackrel{k-1}{\Longrightarrow} x$, $\langle pAr\rangle \stackrel{k'}{\Longrightarrow} x'$, $[rAq] \stackrel{k''}{\Longrightarrow} x''$, 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 A 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 $[pAq] \Longrightarrow \langle pAq\rangle \stackrel{k-1}{\Longrightarrow} x$ (*i.e.*, the first production applied is of the form (iv)), (1), (2), and (3) follow from the induction hypothesis applied to the derivation $\langle pAq\rangle \stackrel{k-1}{\Longrightarrow} x$.

We now consider a derivation $\langle pAq\rangle \stackrel{k}{\Longrightarrow} x$, with $k > 1$. The first step can only be of the form (ii), namely $\langle pAq\rangle \Longrightarrow [p'Xq'] \stackrel{k-1}{\Longrightarrow} x$. From the induction hypothesis, there is a computation path $\mathcal{C}'$ from state p' to state q' which starts and ends with X 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 A at the top of the pushdown, $\mathcal{A}$ can start a computation path which pushes X, simulates $\mathcal{C}'$, and finally pops X off the pushdown. While simulating $\mathcal{C}'$ the pushdown always contains the symbol X over A. 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 $\langle pAq\rangle \Longrightarrow x$ and $[pAq] \Longrightarrow \langle pAq\rangle \Longrightarrow x$.

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 $\langle pAr\rangle \stackrel{\star}{\Longrightarrow} x'$, $[rAq] \stackrel{\star}{\Longrightarrow} x''$, where r is the state reached at the end of $\mathcal{C}'$ and $x'x'' = x$. Using production (iv) we obtain the derivation $[pAq] \Longrightarrow \langle pAr\rangle[rAq] \stackrel{\star}{\Longrightarrow} x'x'' = x$.

If $\mathcal{C}$ satisfies (1), (2′), and (3), then it should start in state p with a push of a symbol X moving in a state p', ends after a pop of the same symbol X from a state q' to state q, where the symbol X 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 $[p'Xq'] \stackrel{\star}{\Longrightarrow} x$ and, considering (ii), $\langle pAq\rangle \Longrightarrow [p'Xq'] \stackrel{\star}{\Longrightarrow} x$. Furthermore, by (iv), we also obtain $[pAq] \stackrel{\star}{\Longrightarrow} x$. □

From Lemma 3, considering production (v), we can easily conclude that the grammar G so defined is equivalent to the given PDA $\mathcal{A}$.

Lemma 4. *The above defined grammar G is non-self-embedding.*

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

Now, we consider any variable of the form $[pAq]$. We observe that in each derivation $[pAq] \stackrel{+}{\Longrightarrow} \alpha[pAq]\beta$, $\alpha, \beta \in (V \cup \Sigma)^*$, the occurence of $[pAq]$ 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 $[pAq]$ is not self-embedded. □

By combining the previous results, we obtain:

Theorem 3. *For each* h-PDA *there exists an equivalent* NSE *grammar of polynomial size.*

Proof. From Lemmas 2 to 4, 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 2 and 3, we obtain that each NSE grammar can be transformed into an equivalent one in a particular form, by paying a polynomial size increase.

Corollary 1. *Each* NSE *grammar is equivalent 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 2, from each NSE grammar G we can obtain an equivalent h-PDA $\mathcal{A}$ of polynomial size which, according to Theorem 3, can be transformed into an equivalent NSE grammar G' as defined above. As observed in the proof of Lemma 4, if $X \stackrel{+}{\Longrightarrow} \alpha X \beta$ 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, it is routine to prove that the unit productions can be 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 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 a 1-LA 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-LAs

We start from an NSE grammar $G = \langle V, \Sigma, P, S\rangle$ in the form given by Corollary 1. Thus, there exists a constant $c \leq \#V$, such that every derivation tree of G has the following properties:

- Each internal node is either *saturated* (*i.e.*, it has exactly two children) and its children are labeled by some variables, or it has a unique child which is labeled by a terminal symbol.
- Along every path, the number of *left turns* (*i.e.*, the number of nodes which are left-child of some node) is bounded by c.

These properties allow us to compress the representation of a derivation tree generating a word w of length m into a word u of length m over the alphabet $\Gamma = \Sigma \times V \times \{0, \ldots, c\}$. The compression is non-injective, thus u may encode different derivation trees. However, each of these derivation trees should generate the same word w, which is the projection of u over Σ. We now describe the compression, which is illustrated in Figure 2. Given a derivation tree T of G, we inductively index its internal nodes according to the following rules:

- the root of T, labeled by the start symbol S, has index 0;

– 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.

In other words, the index of an internal node indicates the number of left turns along the path from the root to it. By assumption on G, this number is bounded by c.

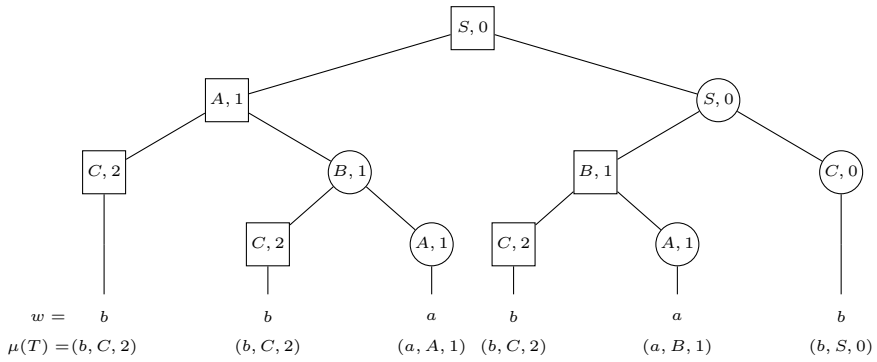

Fig. 2. An example of derivation tree

For a leaf ℓ of the tree labeled by a terminal symbol $a \in \Sigma$, we define $\sigma_\ell = (a, X, i) \in \Gamma$ where (X, i) is the indexed label of the lowest ancestor of ℓ (namely, with minimal i) which is not a right-child of any node (such nodes have square shape in Figure 2). The *compression* of the derivation tree T is defined as the word $\mu(T) = \sigma_{\ell_1} \cdots \sigma_{\ell_m}$ where $\ell_1, \ldots, \ell_m$ are the leaves of T taken from left to right.

We now show how to check that a word $u \in \Gamma^+$ is the compression of some derivation tree. To this end, we highlight the recursive structure of compressions. We first define the three natural projections over Γ: for a symbol $\sigma = (a, X, i) \in \Gamma$, we set $\texttt{letter}(\sigma) = a$, $\texttt{var}(\sigma) = X$ and $\texttt{index}(\sigma) = i$. We fix the convention $\texttt{index}(\triangleright) = \texttt{index}(\triangleleft) = -1$. For $i = 0, \ldots, c$, we define $\Gamma_{>i}$, the restriction of Γ to symbols of index greater than i, *i.e.*, $\Gamma_{>i} = \{\sigma \in \Gamma \mid \texttt{index}(\sigma) > i\}$.

A word $u \in \Gamma^+$ is a *valid compression of level i*, $0 \leq i \leq c$, if, on the one hand, $u = w \cdot (a, X, i)$ for some $w \in \Gamma^*_{>i}$, $a \in \Sigma$ and $X \in V$, 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 $Y, Z \in V$, $b \in \Sigma$, and $v, w' \in \Gamma^*$ such that:
   (a) $X \to YZ$ belongs to P;
   (b) $w = v(b, Y, i+1)w'(a, X, i)$
   (c) $v(b, Y, i+1)$ is a valid compression of level $i+1$;
   (d) $w'(a, Z, i)$ is a valid compression of level i.

In particular, valid compressions of level c are exactly the single-letter word (a, X, c) such that $X \to a \in P$. Observe that in Case 2, Item 2c implies $v \in \Gamma^*_{>i+1}$ and therefore, Y and b are read from the leftmost symbol of index $i+1$ of u. Hence, in order to reconstruct the tree, only the variables "Z's" should be guessed (these variables correspond indeed to nodes that are right-child of some node, represented with circle shape on Figure 2, which are therefore not represented in the compression of the tree). By construction, we have:

Lemma 5. *A word $u \in \Gamma^+$ is a valid compression of level 0 if and only if $u = \mu(T)$ for some derivation tree T of G. Furthermore, T generates a word $w \in \Sigma^+$ which equals the projection of u over Σ^*.*

Proof. We simply observe that Items 1 and 2a check that the recovered tree is consistent with the productions of G, and that Items 2c and 2d are recursive calls on the left and right subtrees, respectively. Conversely, a correct derivation tree can be recovered from its (valid) compression. □

In every compression $\mu(T)$ of some derivation tree T, and for every level index i, a valid compression of level i is a factor delimited to the left, by a symbol of index less than i (not included in the factor), and to the right, by the symbol of index i corresponding to its root node (included in the factor). This allows a reading head to locally detect the boundaries of such a factor of an input. This also implies, that the index

of a symbol preceding a symbol of index i, is always less than or equal to $i+1$. For instance, the compression illustrated in Figure 2 admits two valid compressions of level 1, namely the factors $(b,C,2)(b,C,2)(a,A,1)$ and $(b,C,2)(a,B,1)$ which correspond to the subtree rooted in the square-shape nodes $(A,1)$ and $(B,1)$, respectively.

We now show how a 2NFA can check whether a word $u \in \Gamma^+$ is a valid compression. Basically, the device implements the above-given inductive definition of valid compressions, with the difference that it test each subtree of level from c downto 0 instead of performing recursive calls. This allows to store only one guessed variable Z at each time. We describe the behavior of such a 2NFA $\mathcal{A}$ through a collection of 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.

Procedure `CheckTree`

```
                          /* start with the head on the left endmarker */
1 CheckZeroes
2 for i ← c downto 0 do
3 |   move the head to the right until reaching the next symbol with index i
4 |   while index(σ) = i do
5 |   |   CheckSubtree(i)
6 |   |   move the head to the right until reaching the next symbol with
  |   |     index i
7 |   move the head to the left until reaching the left endmarker
8 Accept
```

As initial phase, the subroutine `CheckZeroes` checks that the input word belongs to $\Gamma_{>0}^* \cdot (a,S,0)$ for some letter $a \in \Sigma$. Then, $\mathcal{A}$ checks the validity of each compressions of each level from c downto 0 (for loop from line 2 to 7). This verification uses the procedure `CheckSubtree` (call line 5).

Procedure `CheckZeroes`

```
9  move the head to the right until reaching the right endmarker
10 move the head to the left
11 if var(σ) ≠ S or index(σ) ≠ 0 then Reject
12 while σ ≠ ▷ do
13     move the head to the left
14     if index(σ) = 0 then Reject
```

This latter subroutine is the direct implementation of the inductive definition of valid compressions, where the recursive call to incremented level (Item 2c) are omitted (the validity of these sub-compressions have already been checked by previous call to the procedure `CheckSubtree`). It uses the subroutine `SelectNext` which finds the leftmost symbol of index $i + 1$ in the factor under consideration, if any, or checks whether the factor has length 1 (if not, the computation rejects, line 19), thus checking Item 2b (or, partially, Item 1). Items 1 and 2a correspond to

Procedure `SelectNext`(i)

```
15 move the head to the right
16 if index(σ) = i − 1 then return
17 else
18     while index(σ) > i do move the head to the right
19     if index(σ) ≠ i then Reject
```

lines 28 and 25, respectively, while C contains the variable Z (line 26) which is initially set to X (line 20), thus allowing to verify Item 2d (while loop lines 23 to 27).

To summarize, we obtained the following result.

Lemma 6. *The language of valid compressions is recognized by a* 2NFA *which uses* $O(\#V^2)$ *states.*

Proof. The construction of such a 2NFA $\mathcal{A}$ has been described above. We now estimate its size. The only memory used in the procedure `CheckTree`,

Procedure `CheckSubtree`(i)

```
                /* start with the head scanning a symbol of index i */
20 C ← var(σ)
21 repeat move the head to the left until index(σ) ≤ i
22 SelectNext(i + 1)
23 while index(σ) ≠ i do
24 |   guess Z
25 |   if C → YZ ∉ P, where Y = var(σ) then Reject
26 |   C ← Z
27 |   SelectNext(i + 1)
28 if C → a ∉ P, where a = letter(σ) then Reject
```

is an index $i \in \{0, \ldots, c\}$, to which both the subroutines `SelectNext` and `CheckSubtree` have read-only access. The subroutines `CheckZeroes` 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 $c \times \#V$. Since $c \leq \#V$, the number of states of $\mathcal{A}$ is in $O(\#V^2)$. □

We are now ready to state our main result.

Theorem 4. *For every* NSE *grammar G, there exist a* 1*-state letter-to-letter nondeterministic transducer $\mathcal{T}$ and a* 2NFA *$\mathcal{A}$ 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 an equivalent* 1-LA *of polynomial size.*

Proof. From an NSE grammar, we obtain an NSE grammar G over Σ of polynomial size in the form given by Corollary 1. The transducer $\mathcal{T}$ replaces each letter $a \in \Sigma$ by a symbol (a, X, i) for some variable X of G and some index $i \leq \#V$. Finally, we build $\mathcal{A}$, using Lemma 6, which recognize an output of $\mathcal{T}$, if and only if its pre-image were generated by G, by Lemma 5. □

The following result directly follows from Theorems 3 and 4.

Corollary 2. *Each h-PDA can be transformed into an equivalent* 1-LA *of polynomial size.*

4.2 From 2DFAs to PDAs: 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 linear size in n, but requires an exponential size in order to be recognized by an h-PDA. We actually prove a stronger result, since each L_n is recognized by a 2DFA 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* [15, Lemma 4.5.1]:

Lemma 7. *Given a context-free language L, there exists a constant $c > 0$, such that for every integer $N \geq 2$, every subset $R \subseteq L \cap \Sigma^N$ of strings of length N, and every integer m with $2 \leq m \leq N$, there exists a subset $Z \subseteq R$ with $Z = \{z_1, z_2, \ldots, z_k\}$ such that $k \geq \frac{\#R}{c(N+1)^2}$, and there exist decompositions $z_i = w_i x_i y_i$, with $1 \leq i \leq k$, such that the following conditions are satisfied:*

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

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

We are now ready to prove the announced exponential gap. For a positive integer n, let L_n be the language of the powers of any string of length n over $\{0,1\}$, *i.e.*, $L_n = \left\{u^k \mid u \in \{0,1\}^n,\ k \geq 0\right\}$.

Theorem 5. *For each $n > 0$, the following holds:*

– *L_n is accepted by a* 2DFA *of size $O(n)$;*

– *Each context-free grammar G 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 a multiple of n. 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 7, with $N = 3n$, $R = \{u^3 \mid u \in \{0,1\}^n\}$ 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 G_n should have at least $\frac{2^n}{(3n+1)^2}$ 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.*, [14, Theorem 8]) we conclude that the size of any PDA accepting L_n is at least exponential in n. □

Corollary 3. *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 5. For the upper bound, in [11] it was proved that each 1-LA can be transformed into an NFA of exponential size from which, by a standard construction, we can obtain a regular (and, so, NSE) grammar, without increasing the size. □

In [2], the question of the cost of the conversion of deterministic h-PDAs into NFAs was raised. To this regard, we observe that the language $(a^{2^n})^*$

is accepted by a deterministic h-PDA (see, *e.g.*, [10]) but, by a standard pumping argument, it requires at least 2^n states to be accepted by NFAs. Actually, as a consequence of state lower bound presented in [9, Theorem 9], 2^n states are also necessary to accept it on each 2NFA. Considering Theorem 5, 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.

References

1. Anselmo, M., Giammarresi, D., Varricchio, S.: Finite automata and non-self-embedding grammars. In: Champarnaud, J., Maurel, D. (eds.) Implementation and Application of Automata, 7th International Conference, CIAA 2002, Tours, France, July 3-5, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2608, pp. 47–56. Springer (2002), http://dx.doi.org/10.1007/3-540-44977-9_4
2. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Removing nondeterminism in constant height pushdown automata. Inf. Comput. 237, 257–267 (2014), https://doi.org/10.1016/j.ic.2014.03.002
3. Chomsky, N.: On certain formal properties of grammars. Information and Control 2(2), 137–167 (1959), https://doi.org/10.1016/S0019-9958(59)90362-6
4. Chomsky, N.: A note on phrase structure grammars. Information and Control 2(4), 393–395 (1959), http://dx.doi.org/10.1016/S0019-9958(59)80017-6
5. Geffert, V., Mereghetti, C., Palano, B.: More concise representation of regular languages by automata and regular expressions. Inf. Comput. 208(4), 385–394 (2010), https://doi.org/10.1016/j.ic.2010.01.002
6. Hibbard, T.N.: A generalization of context-free determinism. Information and Control 11(1/2), 196–238 (1967), https://doi.org/10.1016/S0019-9958(67)90513-X
7. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation - (2. ed.). Addison-Wesley series in computer science, Addison-Wesley-Longman (2001)
8. Kelemenová, A.: Complexity of normal form grammars. Theor. Comput. Sci. 28, 299–314 (1984), http://dx.doi.org/10.1016/0304-3975(83)90026-9
9. Mereghetti, C., Pighizzini, G.: Two-way automata simulations and unary languages. Journal of Automata, Languages and Combinatorics 5(3), 287–300 (2000)
10. Pighizzini, G.: Deterministic pushdown automata and unary languages. Int. J. Found. Comput. Sci. 20(4), 629–645 (2009), https://doi.org/10.1142/S0129054109006784

11. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. Int. J. Found. Comput. Sci. 25(7), 897–916 (2014), https://doi.org/10.1142/S0129054114400140
12. Pighizzini, G., Prigioniero, L.: Limited Automata and Unary Languages. In: Charlier, E., Leroy, J., Rigo, M. (eds.) Developments in Language Theory, vol. 10396, pp. 308–319. Springer International Publishing, Cham (2017), http://link.springer.com/10.1007/978-3-319-62809-7_23
13. Pighizzini, G., Prigioniero, L.: Non-self-embedding grammars and descriptional complexity. In: Freund, R., Mráz, F., Prusa, D. (eds.) Ninth Workshop on Non-Classical Models of Automata and Applications, NCMA 2017, Prague, Czech Republic, August 17-18, 2017. pp. 197–209. Österreichische Computer Gesellschaft (2017)
14. Pighizzini, G., Shallit, J., Wang, M.: Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. J. Comput. Syst. Sci. 65(2), 393–414 (2002), http://dx.doi.org/10.1006/jcss.2002.1855
15. Shallit, J.O.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2008), http://www.cambridge.org/gb/knowledge/isbn/item1173872/?site_locale=en_GB
16. Wagner, K.W., Wechsung, G.: Computational Complexity. D. Reidel Publishing Company, Dordrecht (1986)