Acta Informatica
https://doi.org/10.1007/s00236-022-00434-0

ORIGINAL ARTICLE

Pushdown automata and constant height: decidability and bounds

Giovanni Pighizzini[1] · **Luca Prigioniero**[1]

Received: 29 September 2020 / Accepted: 6 July 2022

Abstract
It cannot be decided whether a pushdown automaton accepts using a pushdown height, which does not depend on the input length, i.e., when it accepts using constant height. Furthermore, when a pushdown automaton accepts in constant height, the height can be arbitrarily large with respect to the size of the description of the machine, namely it does not exist any recursive function in the size of the description of the machine bounding the height of the pushdown. In contrast, in the restricted case of pushdown automata over a one-letter input alphabet, i.e., unary pushdown automata, the situation is different. First, acceptance in constant height is decidable. Moreover, in the case of acceptance in constant height, the height is at most exponential with respect to the size of the description of the pushdown automaton. We also prove a matching lower bound. Finally, if a unary pushdown automaton uses nonconstant height to accept, then the height should grow at least as the logarithm of the input length. This bound is optimal.

1 Introduction

The investigation of computational devices working with a limited amount of resources is a classical topic in automata theory. It is well known that by limiting the memory size of a device by some constant, the computational power of the resulting model cannot exceed that of finite automata. For instance, if we consider pushdown automata in which the maximum height of the pushdown is limited by some constant, the resulting devices, called *constant-height pushdown automata*, can recognize regular languages only. Despite their limited computational power, constant-height pushdown automata are interesting since they allow more succinct representations of regular languages than finite automata [8]. Further properties of these

Extended version of a paper presented at the conference *DCFS 2019-Descriptional Complexity of Formal Systems*, Košice, Slovakia, July 17–19, 2019 [Lecture Notes in Computer Science, vol. 11612 Springer, 2019, pp. 260–271].

✉ Luca Prigioniero
prigioniero@di.unimi.it

Giovanni Pighizzini
pighizzini@di.unimi.it

[1] Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

Published online: 27 July 2022

devices have been recently considered. A double-exponential size increase when converting a nondeterministic constant-height pushdown automaton into an equivalent deterministic one, which cannot be avoided in the worst case, has been proven in [4]. A double-exponential size gap also holds for the conversion of deterministic and nondeterministic constant-height pushdown automata with a two-way input head into equivalent one-way devices [3]. Tight bounds for the size costs of Boolean operations on constant-height pushdown automata have been stated in [5].

A natural generative counterpart of constant-height pushdown automata are *nonself-embedding context-free grammars*, roughly context-free grammars without "true" recursion [7], which have been recently showed to be polynomially related in size to constant-height pushdown automata [10].

In this paper, we focus on pushdown automata with an unrestricted pushdown store, namely *classical pushdown automata*, that, however, are able to accept their inputs by making use only of a constant amount of pushdown store. More precisely, we say that a pushdown automaton $\mathcal{M}$ *accepts in constant height* h, for some given integer $h \geq 0$, if, for each word in the language accepted by $\mathcal{M}$, there exists at least one accepting computation in which the maximum height reached by the store is bounded by h. Notice that this does not prevent the existence of accepting or rejecting computations using an unbounded pushdown height. However, $\mathcal{M}$ can be converted into an equivalent constant-height pushdown automaton, which stops and rejects each time a computation tries to exceed the height limit h, and has a description whose size is a polynomial in both h and the size of the description of $\mathcal{M}$.

While studying these size relationships, we tried to understand *how large can h be with respect to the size of the description of* $\mathcal{M}$. We discovered that h can be arbitrarily large. Indeed, in the first part of the paper we prove that there is no recursive function bounding the maximal height reached by the pushdown store in a pushdown automaton accepting in constant height, with respect to the size of its description.

We also prove that it cannot be decided if a pushdown automaton accepts in constant height. We point out that this problem is different from the classical problem of deciding if a given context-free language is regular, which has been proven to be undecidable long time ago [2]. In fact, there exist pushdown automata that recognize regular languages using nonconstant height (an example is presented in the paper). Hence, while acceptance in constant height is sufficient for the regularity of the accepted language, it is not necessary.

In the second part of the paper, we restrict the attention to the case of pushdown automata with a one-letter input alphabet, namely *unary pushdown automata*. By studying the structure of the computations of these devices, we are able to prove that, in contrast to the general case, it can be decided whether or not they accept in constant height. Furthermore, we also prove that if a unary pushdown automaton $\mathcal{M}$ accepts in height h, constant with respect to the input length, then h is bounded by an exponential function in the size of $\mathcal{M}$. By presenting a suitable family of pushdown automata, we show that this bound cannot be reduced.

In the final part of the paper, we consider pushdown automata that accept using height which is not constant in the input length. Our aim is to investigate how the pushdown height grows. In particular, we want to know if there exists a minimum growth of the pushdown height, with respect to the length of the input, when it is not constant. The answer to this question is already known, and it derives from results on Turing machines: the height of the store should grow at least as a double logarithmic function [1]. This lower bound cannot be increased, because a matching upper bound has been recently obtained in [6], where a witness language defined over an alphabet of 6 letters is presented. Using standard arguments, such language can be encoded on a binary alphabet, without changing the use of the pushdown store. Hence, in the case of an input alphabet with at least two letters, there are languages

accepted by using a pushdown height which is double logarithmic with respect to the input length. When we restrict to a unary alphabet, the situation is different. In fact, as a consequence of the constructions presented in the second part of the paper, we are able to prove that if a unary pushdown automaton accepts using height, which is not constant with respect to the input length, then the height should grow at least as a logarithmic function. We also show that this logarithmic lower bound cannot be further increased, by presenting a unary pushdown automaton accepting every word using logarithmic pushdown height.

2 Preliminaries

We assume the reader to be familiar with the standard notions from formal language and automata theory, including the concepts of configurations and computations of recognizing devices, as presented in classical textbooks, e.g., [12]. As usual, the cardinality of a set S is denoted by $\#S$, the length of a string x is denoted by $|x|$, the empty string is denoted by ε.

We first recall the notion of pushdown automata and present the form for these devices that will be used in the paper. A *pushdown automaton* (PDA, for short) is a tuple $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_I, Z_0, q_F \rangle$ where Q is the finite *set of states*, Σ is the *input alphabet*, Γ is the *pushdown alphabet*, $q_I \in Q$ is the *initial state*, $Z_0 \in \Gamma$ is the *start symbol*, $q_F \in Q$ is the *final state*. We shall specify the transition function δ below, according to Items 3 and 4. Without loss of generality, we make the following assumptions about PDAs:

1. at the start of the computation the pushdown store contains only the start symbol Z_0, being at height 0, the input head is scanning the first input symbol, the finite control contains the initial state q_I;
2. the input is accepted if and only if the automaton reaches the final state q_F, the pushdown store contains only Z_0 and all the input has been scanned;
3. when the automaton reads an input symbol, it moves the head to the next symbol, and it does not make any change on the pushdown. Notice that this implies that the contents of the pushdown store can be changed only by ε-moves;
4. every push operation adds exactly one symbol on the pushdown.

The *transition function* δ of a PDA $\mathcal{M}$ in this form can be written as:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times (\{-, \mathtt{pop}\} \cup \{\mathtt{push}(A) | A \in \Gamma\})}.$$

In particular, for $q, p \in Q$, $A, B \in \Gamma$, $\sigma \in \Sigma$, $(p, -) \in \delta(q, \sigma, A)$ means that the PDA $\mathcal{M}$, in the state q, with A at the top of the pushdown, by consuming the input σ, can reach the state p without changing the pushdown contents; $(p, \mathtt{pop}) \in \delta(q, \varepsilon, A)$ ($(p, \mathtt{push}(B)) \in \delta(q, \varepsilon, A)$, $(p, -) \in \delta(q, \varepsilon, A)$, respectively) means that $\mathcal{M}$, in the state q, with A at the top of the pushdown, without reading any input symbol, can reach the state p by popping off the pushdown the symbol A from the top (by pushing the symbol B onto the top of the pushdown, without changing the pushdown, respectively).

As usual, a *configuration* of a PDA at a given instant represents its instantaneous description. According to [12], it records the internal state of the PDA, the portion of the input that has not been scanned yet, and the pushdown contents. Accepting configurations can be described as indicated above, according to Item 2. Notice that in any accepting computation the occurrence

of the start symbol Z_0 at the bottom of the pushdown is never removed, otherwise the next move would be undefined, so halting in a nonaccepting configuration. [1]

Now we present the main measure we consider in the paper, namely the *pushdown height*. The height of a PDA $\mathcal{M}$ in a given configuration is the number of symbols in the pushdown store *besides* the occurrence of the start symbol Z_0 at the bottom. Hence, in the initial and in the accepting configurations the height is 0. The *height of a computation* $\mathcal{C}$ is the maximum height reached in the configurations occurring in $\mathcal{C}$.

We say that $\mathcal{M}$ uses height $h(x)$ on an accepted input $x \in \Sigma^*$ if and only if $h(x)$ is the minimum pushdown height necessary to accept such a string, namely there exists a computation accepting x of height $h(x)$, and no computation accepting x of height smaller than $h(x)$. Moreover, if x is rejected, then $h(x) = 0$. To study pushdown height with respect to the input length, we consider the worst case among all possible inputs of the same length. Hence, for each integer $n \geq 0$, we define $h(n) = \max\{h(x) \mid x \in \Sigma^*, |x| = n\}$. When there is a constant H such that, for each n, $h(n)$ is bounded by H, we say that $\mathcal{M}$ *accepts in constant height*. Each PDA accepting in constant height can be easily transformed into an equivalent finite automaton. So the language accepted by it is regular.

In the following, by the *size of a* PDA we mean the length of its description. Notice that for each PDA in the above-defined form, over a fixed input alphabet Σ, the size is $O((\#Q)^2(\#\Gamma)^2)$, namely a polynomial in the cardinalities of the set of states and of the pushdown alphabet.

If we consider PDAs in different forms, as that given in [12] in which any push operation can replace the top of the pushdown by a string of symbols, to define the size we have to take into account also the number of symbols that can be pushed on the store in one single operation. However, PDAs in that form can be turned into the form we consider here with a polynomial increase in size and by preserving the property of being constant height. For a further discussion on this point, we address the reader to [4].

We now present some technical notions and results that will be useful in order to state our results. Let $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_I, Z_0, q_F \rangle$ be a fixed PDA.

A *surface pair* is defined by a state $q \in Q$ and a symbol $A \in \Gamma$, and it is denoted by $[qA]$. The surface pair in a given configuration is defined by the current state and the topmost pushdown symbol, namely the only part of the store which is relevant in order to decide the next move.

A *surface triple* is defined by two states $q, p \in Q$ and a symbol $A \in \Gamma$, and it is denoted by $[qAp]$. Surface triples are used to study parts of computations starting and ending at the same pushdown height and that do not go below that height in between. More precisely, a $[qAp]$-*computation* on a string $x \in \Sigma^*$ is a computation $\mathcal{C}$ which starts from the state q with A on the top of the pushdown at some height h and, after reading x from the input tape, ends in the state p with A on the top of the pushdown at the same height h without reaching pushdown height smaller than h in between. We also say that $\mathcal{C}$ *consumes* the string x. Notice that, at the beginning of $\mathcal{C}$, the input head is on the tape cell containing the leftmost symbol of x, while at the end it is one position to the right of the rightmost symbol of x (this includes also the case of inputs whose suffix is x, and the possibility of ε-transitions after reading the rightmost of x). We point out that, during $\mathcal{C}$, the symbol A at height h is never replaced. Hence, $\mathcal{C}$ does not depend on h and on the symbols stored in the pushdown below A. The *pushdown increment* during $\mathcal{C}$ is the difference between the pushdown height of $\mathcal{C}$ and the pushdown height at the beginning and at the end of $\mathcal{C}$. Notice that the surface pairs at the beginning and at the end of $\mathcal{C}$ are $[qA]$ and $[pA]$, respectively.

[1] Note that, according to the definition, a pushdown automaton can push the start symbol Z_0, so it can appear multiple times in the pushdown. However, in all our results, we never push or pop Z_0, so in our constructions and examples it only occurs at the bottom of the store.

We denote by $L_{[qAp]}$ the set of input strings consumed in all possible $[qAp]$-computations. We point out that the set of accepting computations of $\mathcal{M}$ coincides with the set of $[q_I Z_0 q_F]$-computations. Hence, $L_{[q_I Z_0 q_F]}$ is the language accepted by $\mathcal{M}$. Furthermore, for each surface triple $[qAp]$, if we modify $\mathcal{M}$ by using q, A, and p instead of the original initial state, original start pushdown symbol, and the original final state, respectively, we obtain a PDA accepting $L_{[qAp]}$ which, hence, is context-free.

A *horizontal loop* on a surface pair $[qA]$ is any $[qAq]$-computation consuming *at least one input symbol*. Notice that such computation starts and ends in the same state q. By considering a computation of 0 moves, we always have $\varepsilon \in L_{[qAq]}$. Hence, $[qA]$ *has a horizontal loop* when $L_{[qAq]}$ contains at least one more string, besides ε. We obtain that:

Lemma 1 *It is decidable if a surface pair* $[qA]$ *has a horizontal loop.*

Proof As seen above, $L_{[qAq]}$ is a context-free language. It is an easy observation that $L_{[qAq]} \setminus \{\varepsilon\}$ is context free as well. Thus, deciding if the surface pair $[qA]$ has a horizontal loop is equivalent to testing the emptiness of $L_{[qAq]} \setminus \{\varepsilon\}$, which is decidable for context-free languages [2]. □

If a $[qAp]$-computation $\mathcal{C}$ consists of three parts, namely it begins with a prefix $\mathcal{X}$, followed by a proper $[qAp]$-subcomputation $\mathcal{C}'$ using the *same* triple $[qAp]$, and ends by a suffix $\mathcal{Y}$, such that the middle part $\mathcal{C}'$ starts and ends with pushdown higher than at the beginning of $\mathcal{C}$, and then, the pair $(\mathcal{X}, \mathcal{Y})$ is called *vertical loop*. Note that, during the execution of $\mathcal{X}$, a nonempty string $A\alpha$ is saved on the top of the pushdown store[2] above the symbol A which was on the top at the beginning of $\mathcal{C}$, and this string is popped off during the execution of $\mathcal{Y}$.

A context-free grammar is a tuple $G = \langle V, \Sigma, P, S\rangle$, where V is the set of variables, Σ is the set of terminals, P is the set of productions of the form $A \to \beta$, where $A \in V$ and $\beta \in (V \cup \Sigma)^*$, and $S \in V$ is the start symbol. If all productions in P are of the form $A \to BC$ or $A \to a$, where A, B, and C are variables and a is a terminal, then G is in *Chomsky normal form*. Here, we will consider grammars in *binary normal form*, an extension of Chomsky normal form where also *unit productions* $A \to B$ and *ε-productions* $A \to \varepsilon$ are allowed.

If $A \to \beta$ is a production of P and α and γ are any strings in $(V \cup \Sigma)^*$, then the string $\alpha A \gamma$ derives in one step $\alpha\beta\gamma$, in symbols $\alpha A \gamma \Rightarrow \alpha\beta\gamma$. For $\eta, \iota \in (V \cup \Sigma)^*$, $k \geq 0$, we write $x \overset{k}{\Rightarrow} y$ if x derives y in k steps, i.e., $\exists x_0, x_1, \ldots, x_k \in (V \cup \Sigma)^*$ such that $x_0 = \eta$, $x_k = \iota$ and $x_{i-1} \Rightarrow x_i$, for $i = 1, \ldots, k$. The transitive closure of $\Rightarrow$ is denoted by $\overset{+}{\Rightarrow}$, while its reflexive and transitive closure is denoted by $\overset{\star}{\Rightarrow}$. If $\eta \overset{\star}{\Rightarrow} \iota$, $\eta, \iota \in (V \cup \Sigma)^*$, we say that η *derives* ι. For more details on standard notations, we refer the reader to classical textbooks (see, e.g., [12]).

It is well known that context-free languages defined over a one-letter alphabet, i.e., *unary* context-free languages, are regular [9]. The size cost of the conversion of unary context-free grammar and pushdown automata into equivalent nondeterministic and deterministic finite automata (NFAs and DFAs, respectively) has been investigated in [16]. In the paper, we will use the following small extension of [16, Thms. 4, 6]:

Lemma 2 *For each unary context-free grammar* $G = \langle V, \{a\}, P, S\rangle$ *in binary normal form, with* v *variables, there exist:*

– *an equivalent* NFA *with at most* $2^{2v-1} + 1$ *states, and*

[2] The symbol A being at the top.

– *an equivalent* DFA *with less than* 2^{v^2} *states.*

Proof By applying a standard construction (see, e.g., [12]), from G we can obtain a grammar G' in Chomsky normal form, having the same set of variables as G and generating the same language, with the possible exception of the empty word (if generated by G).

Using Theorems 4 and 6 in [16], we can convert G' into equivalent finite automata satisfying the bounds on the number of the states given in the statement of the lemma. By inspecting the proofs of those results, it can be observed that there are no transitions entering the initial states of the resulting automata. This allows to safely mark the initial states as accepting, in the case G generates the empty word, in order to make the resulting automata equivalent to the original grammar G. □

The following result, related to Diophantine equations, will be used in the paper:

Lemma 3 ([14, Lemma 2.6]) *Let* $n, z, i_0, i_1, \ldots, i_s$ *be integers with* $0 < i_j \leq n$, $j = 0, \ldots, s$, *and* $z \geq 0$. *If the equation* $i_0x_0 + i_1x_1 + \cdots + i_sx_s = z$ *has a solution in natural numbers, then it also has a solution in natural numbers satisfying* $i_1x_1 + \cdots + i_sx_s \leq n^2$.

3 Undecidability and nonrecursive bounds

In this section, we prove that the problem of whether a given PDA $\mathcal{M}$ accepts in constant height is not decidable. In addition, with respect to the size of $\mathcal{M}$ that does accept in constant height, neither the maximal height, which is reached by the pushdown store of $\mathcal{M}$, nor the number of states of the minimal finite automaton equivalent to $\mathcal{M}$ can be bounded by any recursive function.

These results are proven by using a technique introduced in [11], based on suitable encodings of single-tape Turing machine computations. Roughly, configurations of such a machine $\mathcal{T}$ with state set Q and alphabet Γ are denoted in a standard way as strings from $\Gamma^* Q \Gamma^*$. A computation consisting of m configurations $\alpha_1, \alpha_2, \ldots, \alpha_m$ is encoded as a string of blocks, separated by a delimiter $\$ \notin Q \cup \Gamma$, where the ith block is α_i when i is odd, and α_i^R when i is even (in the following, we use $\alpha_i^{(R)}$ to denote either α_i^R or α_i according to the parity of the index i).

Hence, the (encoding of a) *valid computation* of $\mathcal{T}$ on input w is a string $\mathcal{C} = \alpha_1\$\alpha_2^R\$\alpha_3\$\alpha_4^R\$\cdots\$\alpha_m^{(R)}$, for some integer $m \geq 1$ such that:

1. $\alpha_i \in \Gamma^* Q \Gamma^*$, i.e., α_i encodes a configuration of $\mathcal{T}$, $i = 1, \ldots, m$;
2. α_1 is the initial configuration on input w, encoded by the string $q_I w$, where q_I is the initial state of $\mathcal{T}$;
3. α_{i+1} is reachable in one step from α_i, $i = 1, \ldots, m-1$;
4. α_m is a halting configuration of $\mathcal{T}$, namely a configuration from which no move is possible.

A *partial valid computation* is defined in a similar way, by dropping Condition 4.

As proven in [11], the complement of the set of all valid computations of $\mathcal{T}$ is a context-free language.

Theorem 1 *It is undecidable whether a* PDA *accepts in constant height.*

Proof We give a reduction from the halting problem. Let $\mathcal{T}$ be a deterministic Turing machine. With an easy modification, we suppose that arbitrarily long computations use arbitrarily large amounts of tape (to this aim, it is sufficient to modify $\mathcal{T}$ by adding to the tape a track where the machine, between any two original consecutive moves, marks a tape cell not yet visited).

By adapting the techniques used in [11] to prove the above-mentioned result, we show that the complement of the language $partial(\mathcal{T}, w)$ of partial computations of $\mathcal{T}$ on a given input w, denoted $(partial(\mathcal{T}, w))^c$, is accepted by a PDA $\mathcal{M}_{\mathcal{T},w}$ in the following way. Given $\mathcal{D} = \beta_1\$\beta_2^R\$\cdots\$\beta_r^{(R)}$, with $\beta_i \in (Q \cup \Gamma)^*$, $i = 1, \ldots, r$, in order to decide whether $\mathcal{D} \in (partial(\mathcal{T}, w))^c$, $\mathcal{M}_{\mathcal{T},w}$ guesses which one among Conditions 1, 2 and 3 is not satisfied. For the first two conditions, the verification of the guess is done by only using the finite control. For the third condition, $\mathcal{M}_{\mathcal{T},w}$ nondeterministically selects one block $\beta_i^{(R)}$, $1 \leq i \leq r$, copies it on the pushdown store and then makes the verification. If $i < r$, this is done by scanning the $(i+1)$th block and by suitably comparing it with the block saved on the pushdown store. If $i = r$, then the verification fails immediately.

We remind the reader that the pushdown height used to accept any input string x is the minimum height of accepting computations on x. Hence, if $\mathcal{D}$ does not satisfy Condition 1 or Condition 2, then it is accepted with pushdown height 0; otherwise, the height is bounded by the length of the first block $\beta_i^{(R)}$ for which Condition 3 is not satisfied, i.e., the block corresponding to the largest i such that $\beta_j = \alpha_j$ for $j = 1, \ldots, i$, where $\alpha_1, \alpha_2, \ldots$ is the (possibly infinite) sequence of configurations in the computation of $\mathcal{T}$ on w.

If $\mathcal{T}$ halts on w in m steps, then the maximum height of the pushdown store used to accept strings in $(partial(\mathcal{T}, w))^c$ is equal to $|\alpha_m|$. Otherwise, for each arbitrarily large integer h, we can find an index $i > 0$ such that $|\alpha_i| > h$. To accept any string $\alpha_1\$\alpha_2^R\$\cdots\$\alpha_i^{(R)}\$\beta$, with $\beta \in \Gamma^* Q \Gamma^*$ and $\beta \neq \alpha_{i+1}^{(R)}$, $\mathcal{M}_{\mathcal{T},w}$ uses height $|\alpha_i| > h$.

This allows to conclude that $\mathcal{T}$ halts on input w if and only if $\mathcal{M}_{\mathcal{T},w}$ accepts in constant height. Hence, it cannot be decided whether a PDA accepts in constant height. □

In Theorem 8, we will present a PDA which recognizes a regular language but does not accept in constant height. Hence, the problem of deciding whether a PDA accepts in constant height is different from the regularity problem for context-free languages, namely the problem of deciding if a given context-free language is regular, which is also undecidable [2].

We point out that in the restricted case of deterministic context-free languages, namely languages accepted by deterministic pushdown automata, the regularity problem is decidable [18]. Even the property in Theorem 1 becomes decidable when we consider *deterministic* PDAs. Indeed, it is already decidable in the case of *unambiguous* PDAs [13].

Each PDA $\mathcal{M}$ accepting in height h can be converted into an equivalent PDA $\mathcal{M}'$ in which the height of each computation is bounded by h. This can be done by attaching a counter either to the pushdown symbols or to the states to keep track, in any configuration, of the current height, in order to stop and reject when a computation tries to exceed the height limit. By encoding the pushdown store of $\mathcal{M}'$ in a finite control, equivalent NFAs and DFAs with a number of states exponential and double exponential in h, respectively, are easily obtained. In the worst case, these bounds cannot be reduced [8]. We now show that, however, h cannot be bounded by any recursive function in the size of $\mathcal{M}$.

Theorem 2 *For any recursive function $f : \mathbb{N} \to \mathbb{N}$ and for infinitely many integers n there exists a* PDA *of size n accepting in constant height $H(n)$, where $H(n)$ cannot be bounded by $f(n)$.*[3]

Proof The argument is derived from [15, Prop. 7]. From among all single-tape deterministic Turing machines having n states and tape alphabet $\Gamma = \{1, \flat\}$ (a finite number of machines), let us take all those that, starting with the empty tape, halt after a finite number of steps (clearly, a finite number of machines again) and, from among them, let $\mathcal{BB}_n$ be one—called

[3] We point out that here $H(n)$ is a function of the size of the PDA and *not* of the input.

a *busy beaver*—that stops with the largest number of 1's, denoted as $\Sigma(n)$, written down on the tape. (There may exist more than one such machine; in that case, we can take the first one in some fixed enumeration of Turing machines.) It is known that $\Sigma(n)$ cannot be bounded by any recursive function [17]. Hence, also the maximal length of configurations occurring in such a computation cannot be bounded by any recursive function.

Let $\mathcal{C}_n$ be the encoding of the valid computation of $\mathcal{BB}_n$ on ε. By adapting the arguments used to prove Theorem 1, we can define a PDA $\mathcal{M}_n$ which accepts all the strings over $(Q_n \cup \Gamma \cup \{\$\})^*$ different from $\mathcal{C}_n$, using height bounded by the length of the longest configuration occurring in $\mathcal{C}_n$. Since n is fixed, $\mathcal{M}_n$ accepts in constant height. Moreover, it uses a constant number of states for testing that either one of Conditions 1, 2, and 4 does not hold for $\mathcal{C}_n$ to be a valid computation. For Condition 3, namely to check whether two configurations of $\mathcal{BB}_n$ are not reachable in one step, the machine has to compare the parts of configurations representing the cells of the tape different from the head position in the first configuration, which can be done using the pushdown store and a constant number of states, and the part (state and symbol) that is modified according to the transition function of $\mathcal{BB}_n$. Each transitions can be checked using a constant number of states. Because $\mathcal{BB}_n$ has n states and its working alphabet is fixed, it has $O(n)$ transitions. Hence, to check Condition 3, $\mathcal{M}_n$ uses $O(n)$ states. Summing up, the number of states of $\mathcal{M}_n$ is $O(n)$, its pushdown alphabet has cardinality $O(n)$. So, according to Sect. 2, $\mathcal{M}_n$ has size $O(n^4)$.

Furthermore, by suitably modifying $\mathcal{C}_n$ (with the same method we applied in the last part of the proof of Theorem 1 to a prefix of the string encoding the infinite computation of the machine $\mathcal{T}$ on input w), we can obtain a string that requires height equal to the maximal length of configurations occurring in $\mathcal{C}_n$ to be accepted by $\mathcal{M}_n$.

This allows to conclude that the pushdown height used by $\mathcal{M}_n$ cannot be bounded by any recursive function in the size of $\mathcal{M}_n$. □

The PDA $\mathcal{M}_n$ used to prove Theorem 2 accepts the complement of the singleton language $\{\mathcal{C}_n\}$. This implies that each equivalent deterministic automaton requires more than $|\mathcal{C}_n|$ states. Hence,

Corollary 1 *There is no recursive function bounding the size blowup from* PDA*s accepting in constant height to finite automata.*

4 Constant height decidability in the unary case

In Sect. 3, we proved that it cannot be decided if a PDA accepts in constant height. This section is devoted to showing that this property turns out to be decidable in the restricted case of PDAs with a one-letter input alphabet. We point out that it is well known that unary context-free languages are regular [9], so unary PDAs can always be converted into equivalent DFAs and therefore also into equivalent PDAs with constant height equal to zero. The problem considered here is whether the given PDA works or does not work with a pushdown of constant height. We first give an informal outline of the argument.

Any accepting computation on a sufficiently long input should contain horizontal or vertical loops. The use of vertical loops can lead to computations using unbounded height. However, we prove that if an accepting computation on an input a^ℓ visits a surface pair on which there exists a horizontal loop, then there is another accepting computation for the same input in which almost all occurrences of the vertical loops are replaced by occurrences of such a horizontal loop. The number of vertical loops that remain in the resulting computation

is bounded by a constant that only depends on the automaton. Hence, the height of such a computation is bounded by a constant. As a consequence, a^ℓ is accepted in constant height. This result is obtained by refining pumping arguments on grammars and the fact that, in the unary case, input symbols commute. In contrast, if no accepting computation on a long string a^ℓ visits any surface pair having a horizontal loop, vertical loops and an increasing of the pushdown height cannot be avoided. Hence, the given PDA works in constant height if and only if the cardinality of the language $L_v \setminus L_h$ is finite, where L_h (L_v, resp.) is the set of strings which are accepted by a computation visiting a (not visiting any, resp.) surface pair having a horizontal loop. Since the languages L_v and L_h are accepted by PDAs, so they are context free, and they are defined over a one-letter alphabet, by a well-known result proved in [9], they are regular. So the finiteness of their difference is decidable.

To obtain these results, we refine some of the arguments given in [16] to study the size costs of the transformations of unary context-free grammars and pushdown automata into equivalent finite automata.

4.1 Loops and grammars

In the following, we consider a grammar $G = \langle V, \Sigma, P, S\rangle$ in *binary normal form* and we denote by $v = \#V$ the number of its variables.

If T is a derivation tree whose root is labeled with a variable $A \in V$ and such that the labels of the leaves, from left to right, form a string $\gamma \in (V \cup \Sigma)^*$, then we write $T : A \stackrel{\star}{\Rightarrow} \gamma$. Furthermore, we indicate by $\nu(T)$ the set of variables occurring as labels of the nodes in T. As usual, the *height of a derivation tree* T is the maximum number of edges along the path from the root to a leaf in T.

A *gap tree* T from a variable $A \in V$, also called *A-gap tree*, is a tree corresponding to a nonempty derivation of the form $A \stackrel{+}{\Rightarrow} xAy$, with $x, y \in \Sigma^*$. When $x = y = \varepsilon$, the gap tree T is said to be *trivial*, otherwise, i.e., when it has at least one leaf labeled by a terminal, T is *nontrivial*.

Lemma 4 *If $A \stackrel{\star}{\Rightarrow} \gamma$, $A \in V$, $\gamma \in (V \cup \Sigma)^+$, then there exists a derivation tree $T : A \stackrel{\star}{\Rightarrow} \gamma$ of height at most $(|\gamma| + 1)v$.*

Proof Given a derivation tree $T : A \stackrel{\star}{\Rightarrow} \gamma$ of height $h > (|\gamma| + 1)v$, let $n_1, n_2, \ldots, n_h$ be the sequence of the internal nodes which are encountered on a longest path in T, moving from the leaf to the root. With each node n_k, we associate the pair (A_k, γ_k), where A_k is the variable labeling n_k and γ_k is the string generated by the subtree rooted at n_k.

Hence, for $k = 2, \ldots, h$, γ_{k-1} is a factor of γ_k and $\gamma_h = \gamma$. Considering that γ_1 could be ε, the number of possible different second components in these pairs is bounded by $|\gamma| + 1$.

Since $h > (|\gamma| + 1)v$, this implies that there is a sequence of $v + 1 > \#V$ indices $k, k + 1, \ldots, k + v$, with $1 \le k \le h - v$, such that $\gamma_k = \gamma_{k+1} = \ldots = \gamma_{k+v}$. Hence, $A_i = A_j$, for some i, j, with $k \le i < j \le k + v$, namely the tree $\hat{T}$ obtained by removing from the subtree of T rooted at n_j the subtree rooted at n_i, is a trivial A_i-gap tree. By removing $\hat{T}$ from T, i.e., by replacing the subtree rooted at n_j by the subtree rooted at n_i, we obtain a tree $T' : A \stackrel{\star}{\Rightarrow} \gamma$ with a smaller number of nodes than T.

We can iterate this process, up to obtain a tree for a derivation $A \stackrel{\star}{\Rightarrow} \gamma$ of height bounded by $(|\gamma| + 1)v$. □

Lemma 5 *Let $T : A \stackrel{\star}{\Rightarrow} \gamma$, $A \in V$, $\gamma \in (V \cup \Sigma)^+$, be a derivation tree.*

1. *Let k be the length of the longest path from the root to a leaf labeled by a terminal symbol, if any. Then, γ contains at most 2^{k-1} terminal symbols, and less than 2^{k-1} symbols when γ contains at least one variable.*
2. *If γ contains only terminal symbols, i.e., $\gamma \in \Sigma^*$, and the height of T is k, then $|\gamma| \leq 2^{k-1}$.*
3. *If $\gamma = xAy$, $xy \in \Sigma^+$, and T has a minimal number of nodes among all nontrivial A-gap trees, then $|xy| < 2^{2v-1}$.*

Proof The proof is given by adapting standard properties of derivation trees of grammars in Chomsky normal form (see, e.g., [12]).

1. The statement can be proven by induction on k. If $k = 1$, then the tree consists only of the root, labeled by A, with one son, labeled by $a \in \Sigma$, where $A \to a$ is a production of G. In this case, the statement is trivial. If $k > 1$ then, in any subtree of the root, the longest path to a leaf labeled by a terminal symbol has length at most $k-1$, so generating, by induction hypothesis, a string containing at most 2^{k-2} terminal symbols. Due to the form of the grammar, the root can have at most 2 subtrees. Hence, the number of terminal symbols in γ is bounded by 2^{k-1}. Furthermore, when γ contains one variable, one of the subtrees of the root derives a factor of γ containing such a variable. Hence, by induction, it generates a number of terminals which is strictly less than 2^{k-2}. As a consequence, the number of terminals in γ is less than 2^{k-1}.
2. Consequence of Item 1.
3. Let $n_1, n_2, \ldots, n_k$ be the sequence of the internal nodes on a longest path in T from a leaf labeled by a terminal symbol $a \in \Sigma$ to the root. With each node n_i, $i = 1, \ldots, n$, we associate a pair (A_i, b_i) where $A_i \in V$ and $b_i \in \{0, 1\}$ is 1 if and only if the factor of γ generated by the subtree rooted at n_i contains the variable A. Hence, the pair associated with n_1 is $(B, 0)$ for some $B \in V$ having the production $B \to a$, while the pair associated with the root n_k is $(A, 1)$.
   Suppose $k > 2v$. Then, there are two nodes n_i, n_j, with $1 \leq i < j \leq k$ with $(A_i, b_i) = (A_j, b_j)$. By replacing in T the subtree rooted at n_j by the subtree rooted at n_i, we obtain an A-gap tree T' which still generates at least one terminal symbol and has less nodes than T, which is a contradiction.
   Hence, in T, each path connecting the root and a leaf labeled by a terminal symbol should have length at most $2v$, which, according to Item 1, implies $|xy| < 2^{2v-1}$. □

From now on, let us suppose that G is unary, i.e., $\Sigma = \{a\}$. The following modified version of Lemma 2(ii) in [16] is derived from the arguments of the classical "pumping lemma" for context-free languages.

Lemma 6 *Let $T : S \stackrel{\star}{\Rightarrow} a^\ell$ be a derivation tree of G. If $\ell > 2^{v^2-1}$, then there exist three integers s, i, j, with $\ell = s + i + j$, $s \geq 0$, and $0 < i + j < 2^{v^2}$, a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$, a variable $A \in v(T)$, and an A-gap tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$, such that $v(T) = v(T_1) \supseteq v(T_2)$.*

Proof We use a combinatorial argument similar to that in the proofs of Lemmas 4 and 5 . Let $n_1, n_2, \ldots, n_e$ be the sequence of the internal nodes which are encountered on a longest path in T, moving from the leaf to the root. With each n_f, we associate the pair (A_f, α_f), where $A_f \in V$ is the variable labeling n_f and $\alpha_f \subseteq v(T)$ is the set of variables occurring in the subtree rooted at n_f, $f = 1, \ldots, e$. Hence, $\alpha_1 = \{A_1\}$, $\alpha_e = v(T)$, and $\alpha_{f-1} \subseteq \alpha_f$, $f = 2, \ldots, e$. Notice that we can have at most v different α_f's.

Since $\ell > 2^{v^2-1}$, from Lemma 5(2) we get $e > v^2$. Thus, there are more than v consecutive pairs (A_f, α_f) with the same second component and, so, the first components of two of them

should coincide. In other words, we can find two nodes n_x and n_y, $0 < x < y \leq v^2 + 1$, such that $(A_x, \alpha_x) = (A_y, \alpha_y)$ and $y - x \leq v$. By replacing in T the subtree rooted at n_y by the subtree rooted at n_x, we get a new tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$ with $s \leq \ell$. Let T_2 be the gap tree obtained from T by taking as root n_y and by deleting the subtree rooted at n_x. Then, $T_2 : A_x \stackrel{+}{\Rightarrow} a^i A_x a^j$, for some integers i, j with $s + i + j = \ell$. Furthermore, $\nu(T) = \nu(T_1) \supseteq \nu(T_2)$.

Since the root of T_2 is n_y, its height is at most $v^2 + 1$. By Lemma 5(1), this implies $i + j < 2^{v^2}$.

Finally, we observe that in case $i + j = 0$ and $s = \ell$, we can repeat the same argument after replacing T by T_1. Since the number of nodes in the "new" T is smaller than in the "old" one, by iterating this process, at some point we will finally obtain a tree T_1 producing a shorter string and a gap tree T_2 producing at least one terminal symbol. □

The following lemma will be crucial to obtain our main result. We prove that each long enough string a^ℓ can be derived by pumping a derivation tree of some short string by many occurrences of a *same* gap tree. Furthermore, such a gap tree can be arbitrarily chosen among "small" nontrivial A-gap trees, with A occurring in the derivation of a^ℓ.

Lemma 7 *For any derivation tree $T : S \stackrel{\star}{\Rightarrow} a^\ell$ and for any A-gap tree $T_A : A \stackrel{\star}{\Rightarrow} a^i A a^j$, with $0 < i + j < 2^{2v-1}$ and $A \in \nu(T)$, there exists a derivation tree $T' : S \stackrel{\star}{\Rightarrow} a^\ell$ which is obtained by pumping a tree $T_0 : S \stackrel{\star}{\Rightarrow} a^{\ell_0}$ such that $\nu(T_0) = \nu(T)$, $0 \leq \ell_0 \leq 2^{2v^2} - 3 \cdot 2^{v^2-1} + 1$, with $k \geq 0$ occurrences of T_A.*

Proof If $\ell \leq 2^{2v^2} - 3 \cdot 2^{v^2-1} + 1$, then we take $T_0 = T$, $\ell_0 = \ell$, and $k = 0$. Otherwise, we repeatedly apply Lemma 6 to "unpump" the tree T up to find a tree $T_r : S \stackrel{\star}{\Rightarrow} a^{\ell_r}$, with $\ell_r \leq 2^{v^2-1}$ and $\nu(T_r) = \nu(T)$.

Let $\{i_1, \ldots, i_s\} \subseteq \{1, \ldots, 2^{v^2} - 1\}$ be the set of numbers of terminals that are generated by the gap trees removed during this process. Hence, $\ell = \ell_r + i_1 x_1 + \cdots + i_s x_s$, where, for $t = 1, \ldots, s$, $x_t > 0$ is the number of gap trees generating i_t terminal symbols that have been removed to obtain T_r. Let $i_0 = i + j < 2^{2v-1} \leq 2^{v^2}$ be the number of terminals generated by the tree T_A. By Lemma 3 (applied with $z = \ell - \ell_r$ and $x_0 = 0$), we can find integers $x'_0, x'_1, \ldots, x'_s \geq 0$ in such a way that $\ell = \ell_r + i_0 x'_0 + i_1 x'_1 + \cdots + i_s x'_s$ and $i_1 x'_1 + \cdots + i_s x'_s \leq (2^{v^2} - 1)^2$. This means that we can pump the tree T_r with a suitable number of occurrences of some of the gap trees removed in the previous process, in order to get a tree $T_0 : S \stackrel{\star}{\Rightarrow} a^{\ell_0}$, with $\ell_0 = \ell_r + i_1 x'_1 + \cdots + i_s x'_s \leq 2^{v^2-1} + (2^{v^2} - 1)^2 = 2^{2v^2} - 3 \cdot 2^{v^2-1} + 1$ and $\nu(T_0) = \nu(T)$. Furthermore, by pumping T_0 with x'_0 occurrences of T_A, we finally get a tree $T' : S \stackrel{\star}{\Rightarrow} a^\ell$. □

4.2 Simulating vertical loops by a horizontal loop

From now on, let us consider a fixed PDA $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_I, Z_0, q_F \rangle$. We are going to define a context-free grammar $G = \langle V, \Sigma, P, S \rangle$, in binary normal form, which generates the same language accepted by $\mathcal{M}$. We give the same construction as in [16], which is a minor variation of that used in classical textbooks (see, e.g., [12]) to present the standard transformation of PDAs into CFGs. The grammar G is defined as follows:

– The set of variables V consists of all triples $[q A p]$, with $q, p \in Q$, $A \in \Gamma$.
The name of the variable $[q A p]$ immediately evokes $[q A p]$-computations. This reflects

a strict relationship; indeed, the grammar is defined in such a way that the set of strings generated by the variable $[qAp]$ coincides with the language $L_{[qAp]}$ of strings consumed in all possible $[qAp]$-computations. This property will be proven, after the definition of the grammar, in Lemma 8.

– The set of terminals coincides with the input alphabet Σ of $\mathcal{M}$.
– P contains the following productions:

1. $[qAp] \to [qAr][rAp]$, for $q, p, r \in Q$, $A \in \Gamma$;
2. $[qAp] \to [q'Bp']$, for $q, q', p, p' \in Q$, $A, B \in \Gamma$ such that $(q', \mathtt{push}(B)) \in \delta(q, \varepsilon, A)$ and $(p, \mathtt{pop}) \in \delta(p', \varepsilon, B)$;
3. $[qAp] \to \sigma$, for $q, p \in Q$, $\sigma \in \Sigma \cup \{\varepsilon\}$, $A \in \Gamma$ such that $(p, -) \in \delta(q, \sigma, A)$;
4. $[qAq] \to \varepsilon$, for $q \in Q$, $A \in \Gamma$.

– The start symbol S is the triple $[q_I Z_0 q_F]$.

We point out that the number of variables of G is $v = (\#Q)^2 \cdot \#\Gamma$. Furthermore, it is in binary normal form.

We are going to prove that G generates the same language accepted by $\mathcal{M}$. Since we are interested in the height of $\mathcal{M}$'s computations, we state such equivalence in a stronger form, which also considers the use of the pushdown.

In particular, we relate the pushdown increment to the *unit production height* which, for a derivation tree T of the above grammar G, is defined as the maximum number of edges corresponding to unit productions, i.e., productions of the form $A \to B$, with $A, B \in V$, in a path from the root to a leaf of T.

Lemma 8 *For any $x \in \Sigma^*$, $q, p \in Q$, $A \in \Gamma$, $h \in \mathbb{N}$, there exists a derivation tree $T : [qAp] \stackrel{\star}{\Rightarrow} x$ with unit production height h if and only if there exists a $[qAp]$-computation $\mathcal{C}$ on x with pushdown increment h.*

Proof Let $T : [qAp] \stackrel{k}{\Rightarrow} x$, for some $k > 0$, be a derivation tree with unit production height h. We prove by induction on k that there exists a $[qAp]$-computation $\mathcal{C}$ on x with pushdown increment h.

If $k = 1$, then the tree contains only the root and one leaf and it corresponds to the use of one of the productions of the form 3 or 4. So the statement is trivial.

If $k > 1$, then the production used at the root level of T is either of the form 1 or of the form 2.

In the first case, we have $[qAp] \to [qAr][rAp]$, for some $r \in Q$, the root of T has left subtree $T' : [qAr] \stackrel{k'}{\Rightarrow} x'$ and a right subtree $T'' : [rAp] \stackrel{k''}{\Rightarrow} x''$, for some $k', k'' > 0$ with $k' + k'' = k - 1$, $x', x'' \in \Sigma^*$, $x'x'' = x$. Then, the unit production height h of T is the maximum between the unit production heights h' and h'' of T' and T'', respectively, i.e., $h = \max\{h', h''\}$. According to the induction hypothesis, there exist a $[qAr]$-computation on x' and a $[rAp]$-computation on x'' with pushdown increment h' and h'', respectively. By concatenating these two computations, we obtain a $[qAp]$-computation on x in which the pushdown increment is $\max\{h', h''\}$, namely h.

In case the production applied to the root of T is $[qAp] \to [q'Bp']$, let $T' : [q'Bp'] \stackrel{k-1}{\Rightarrow} x$ be the subtree of T rooted at the only son of the root. Since in T at the top level a unit production is used, the unit production height on a path from the root of T' is $h - 1$. Also in this case, from the induction hypothesis we obtain a $[q'Bp']$-computation on x with pushdown increment $h - 1$. By adding to this computation the initial push and the final pop from which the production $[qAp] \to [q'Bp']$ is defined, we obtain a $[qAp]$-computation on x with pushdown increment h.

Conversely, let us consider a $[qAp]$-computation $\mathcal{C}$ on x with pushdown increment h. We proceed on the number k of steps of $\mathcal{C}$.

If $k = 1$, then the computation $\mathcal{C}$ does not make any pushdown increment and it can only correspond to a one-step derivation consisting of a production of the form 3 or 4. So the statement is trivial.

If $k > 1$, then we consider two cases, depending on whether or not at some configuration in $\mathcal{C}$, after the first and before the last configuration, the pushdown is at the same height than at the beginning and at the end of $\mathcal{C}$.

- If such configuration exists, then we split $\mathcal{C}$ at that configuration into a $[qAr]$-computation $\mathcal{C}'$ and a $[rAp]$-computation $\mathcal{C}''$, for some $r \in Q$, consuming some x', x'', with $x'x'' = x$ and with pushdown increment h', h'', respectively. Then, the pushdown increment in $\mathcal{C}$ is $\max\{h', h''\}$. Using the induction hypothesis, we find two trees T' and T'' corresponding to such computations, with unit production heights h' and h'', respectively. We can suitably combine T' and T'', using a production of form 1, in order to obtain a tree T which derives x and has height equal to $\max\{h', h''\}$.
- If such configuration does not exist, then the computation of $\mathcal{C}$ should start with a push of a symbol B which is removed in the last step. Let $\mathcal{C}'$ be $[q'Br']$-subcomputation of $\mathcal{C}$ which is obtained by removing the first and the last step. If h is the pushdown increment in $\mathcal{C}$, then the pushdown increment in $\mathcal{C}'$ is $h - 1$. Let $T' : [q'Br'] \stackrel{\star}{\Rightarrow} x$ be the tree corresponding to $\mathcal{C}'$, obtained according to the induction hypothesis. Its unit production height is $h - 1$. The tree T, which is obtained by taking T' as only subtree of a root with label $[qAr]$, derives x and has unit production height h. □

As a consequence of Lemma 8, we get:

Corollary 2 *For any integer $h \geq 0$, a string x is accepted by $\mathcal{M}$ using pushdown height h if and only if there is a derivation tree T of x in G with unit production height h.*

Combining Corollary 2 with Lemma 4, we get the following upper bound for the height of the pushdown store necessary to accept a string x:

Lemma 9 *If $x \in \Sigma^*$ is accepted by $\mathcal{M}$, then $h(x) \leq (|x| + 1)v$, where $v = (\#Q)^2 \cdot \#\Gamma$.*

Proof By contradiction, suppose that each computation of $\mathcal{M}$ accepting x uses pushdown height greater than $(|x|+1)v$. As a consequence of Corollary 2, each derivation tree of x in G has unit production height, and so height, greater than $(|x| + 1)v$, which is a contradiction to Lemma 4. □

Let us go back to the case of unary pushdown automata. Hence, from now on let $\mathcal{M} = \langle Q, \{a\}, \Gamma, \delta, q_I, Z_0, q_F\rangle$ be a fixed *unary* PDA. Using Lemma 8, we can reformulate Lemma 7 in terms of pushdown automata. Roughly, we can say that for each computation $\mathcal{C}$ accepting a "long" input, there is another computation accepting the same input, which is obtained by pumping a suitable computation $\mathcal{C}_0$, chosen from a finite set, with a repeated pattern which is arbitrarily selected from another finite set that depends on $\mathcal{C}_0$. We will use this property to replace, in any accepting computation $\mathcal{C}$, almost all the vertical loops with many occurrences of a horizontal loop, in the case a surface pair $[rB]$ having a horizontal loop occurs in $\mathcal{C}$. In this way, we will be able to obtain an accepting computation of bounded height on the same input.

Theorem 3 *Let $\mathcal{C}$ be an accepting computation on input a^ℓ which visits a surface pair $[rB]$ having a horizontal loop. Then, there exists an accepting computation on a^ℓ of height smaller than $2^{2v^2+\log_2 v}$, where $v = (\#Q)^2 \cdot \#\Gamma$.*

Proof Let G be the above-defined grammar, obtained from $\mathcal{M}$. First, we observe that if $\mathcal{C}$ visits the surface pair $[rB]$ then there exists a derivation tree $T : S \stackrel{\star}{\Rightarrow} a^\ell$ with $[rBr] \in \nu(T)$. In fact, one of the triples $[rBs]$ or $[sBr]$ for some $s \in Q$ should appear in the derivation tree corresponding to $\mathcal{C}$. Since G contains the productions $[rBs] \to [rBr][rBs]$, $[sBr] \to [sBr][rBr]$ and $[rBr] \to \varepsilon$, we can suitably modify the tree in order to introduce one occurrence of $[rBr]$, without changing the derived string.

Now we select a "small" $[rBr]$-gap tree $T_{[rBr]}$ deriving a nonempty string, i.e., $T_{[rBr]} : [rBr] \stackrel{\star}{\Rightarrow} a^i[rBr]a^j$, with $0 < i + j < 2^{2v-1}$. We prove that such a gap tree should exist. In fact, since $[rB]$ has a horizontal loop, the language $L_{[rBr]}$ should contain at least one nonempty string $w \in a^*$. Hence, $[rBr] \stackrel{\star}{\Rightarrow} w$. Furthermore, the tree corresponding to the derivation

$$[rBr] \stackrel{\star}{\Rightarrow} [rBr][rBr] \stackrel{\star}{\Rightarrow} w[rBr]$$

is a $[rBr]$-gap tree. Hence, from Lemma 5(3), it follows that there exists a $[rBr]$-gap tree $T_{[rBr]} : [rBr] \stackrel{\star}{\Rightarrow} a^i[rBr]a^j$, with $0 < i + j < 2^{2v-1}$.

According to Lemma 7, we can obtain another tree $T' : S \stackrel{\star}{\Rightarrow} a^\ell$ by pumping a tree $T_0 : S \stackrel{\star}{\Rightarrow} a^{\ell_0}$, such that $\nu(T_0) = \nu(T)$, $0 \le \ell_0 \le 2^{2v^2} - 3 \cdot 2^{v^2-1} + 1$, with $k \ge 0$ occurrences of $T_{[rBr]}$.

We observe that in the tree T', some of the k occurrences of $T_{[rBr]}$, say t, could be nested, possibly giving a pushdown height of the corresponding computation which linearly increases with k. To fix this problem, we modify T' as we now describe.

Let u be a node of T_0 labeled by $[rBr]$ and T_u be the subtree of T_0 rooted at u, such that T_0 is pumped starting from u with t nested occurrences of $T_{[rBr]}$, $1 < t \le k$. (The subtree rooted at u after the pumping is shown in Fig. 1, on the left.) We rearrange these t occurrences of $T_{[rBr]}$ in a sequence by inserting, starting from node u, a subtree corresponding to a derivation $[rBr] \stackrel{\star}{\Rightarrow} [rBr]^t$ obtained by using $t - 1$ times the production $[rBr] \to [rBr][rBr]$. To each leaf of this subtree, we append one occurrence of the $[rBr]$-gap tree $T_{[rBr]}$. Finally, to the leaf labeled $[rBr]$ of the first occurrence of $T_{[rBr]}$ we append the tree T_u, and to each of the remaining $t - 1$ leaves labeled $[rBr]$ we append one leaf labeled with the empty word (we remind the reader that $[rBr] \to \varepsilon$ is a production of G). The subtree rooted at u after the pumping and the one obtained after the rearrangement are shown in Fig. 1.

Let T'' be the tree obtained after this modification, which still generates a^ℓ. Using Corollary 2, we now estimate the height of the computation $\mathcal{C}''$ corresponding to T'', by calculating the unit production height of T'', which is bounded by the maximum number h_0 of edges corresponding to unit productions in any path in T_0 plus the maximum number h_1 of such edges in any path in $T_{[rBr]}$ which, in turn, are bounded by the height of T_0 and $T_{[rBr]}$, respectively. Using Lemma 4, we get $h_0 \le (\ell_0 + 1)v$ and $h_1 \le (i + j + 2)v$ (we remind the reader that the tree $T_{[rBr]}$ generates a string of length $i + j + 1$). Hence, the height of the pushdown is bounded by $h_0 + h_1 \le (\ell_0 + i + j + 3)v$. Considering the bounds on ℓ_0 and $i + j$, we obtain $\ell_0 + i + j + 3 < 2^{2v^2} - 3 \cdot 2^{v^2-1} + 2^{2v-1} + 4$.

For $v \ge 2$, it can be verified that $-3 \cdot 2^{v^2-1} + 2^{2v-1} < -4$. Hence, $h_0 + h_1 < 2^{2v^2} \cdot v = 2^{2v^2+\log_2 v}$.

In the case $v = 1$, the PDA $\mathcal{M}$ can have only one state q, which is both initial and final, and only one pushdown symbol Z_0. Since in the form we are considering for PDAs transitions consuming input symbols do not change the pushdown (cf. Sect. 2), the only possibility to read an input symbol is that of having the transition $(q, -) \in \delta(q, a, Z_0)$. If this is the case, then any string in a^* can be accepted by a computation which does not use the pushdown

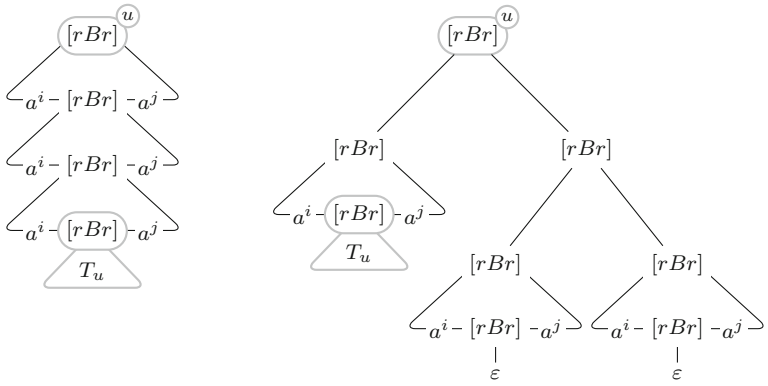

Fig. 1 On the left, the portion of the tree T_0 from the node u pumped by using t repetitions of the tree $T_{[rBr]}$. These occurrences are rearranged by using $t-1$ times the production $[rBr] \to [rBr][rBr]$, as shown on the right, in order to avoid a linear increase of the pushdown in the number of the repetitions of $T_{[rBr]}$. (In the figure $t = 3$)

store, i.e., of height 0. Otherwise, $\mathcal{M}$ accepts the empty language and so, by definition, it accepts in height 0. □

4.3 Vertical increase without horizontal loops

Now we evaluate the increase of the pushdown in computations that do not use horizontal loops, i.e., between any two repetitions of a same surface pair $[rB]$ at the same height either no input is consumed or there is at least one configuration with lower pushdown height.

Lemma 10 *Let $\mathcal{C}$ be a $[qAp]$-computation on a^ℓ with pushdown increment bounded by h and without horizontal loops. Then $\ell \leq (\#Q-1)^{h+1}$.*

Proof We give the proof by induction on h. Let h_0 be the pushdown height at the beginning and at the end of $\mathcal{C}$. We preliminary observe that since in $\mathcal{C}$ the pushdown height cannot be lower than h_0 and there are no horizontal loops, between any two repetitions of a same state at pushdown height h_0 no input symbols can be consumed, or else we would have a horizontal loop, a contradiction. Hence, we can remove from $\mathcal{C}$ the part between any two repetitions of such a state, to obtain a shorter $[qAp]$-computation on the same input having pushdown increment bounded by h. By iterating this process, we finally get $\mathcal{C}$ with at most $\#Q$ configurations at pushdown height h_0.

If $h = 0$, i.e., the pushdown height is never incremented, then $\mathcal{C}$ consists of at most $\#Q-1$ moves. Hence, $\ell \leq \#Q - 1 = (\#Q-1)^{h+1}$. Otherwise, we decompose $\mathcal{C}$ in $k < \#Q$ subcomputations $\mathcal{C}_1, \ldots, \mathcal{C}_k$, where, for $i = 1, \ldots, k$, $\mathcal{C}_i$ starts with a push of a symbol, which is popped off the pushdown only in the last move of $\mathcal{C}_i$. Let $\mathcal{C}'_i$ be the subcomputation obtained by removing from $\mathcal{C}_i$ the first and the last move and let a^{ℓ_i} be the input consumed during it. Then, the pushdown increment in $\mathcal{C}'_i$ is at most $h-1$. By induction hypothesis, this implies $\ell_i \leq (\#Q-1)^h$. Since push and pop moves do not consume input symbols, we get that $\ell \leq k(\#Q-1)^h \leq (\#Q-1)^{h+1}$. □

As a consequence of Lemma 10, the recognition of arbitrarily long strings without making use of horizontal loops requires unbounded pushdown height. This fact will be used later to derive a lower bound for such a pushdown height.

4.4 Decidability

Using the tools we developed so far, we are now able to prove the main result of this section:

Theorem 4 *Let $\mathcal{M}$ be a unary* PDA *with n states and m pushdown symbols. Then, $\mathcal{M}$ accepts in constant height if and only if it accepts in height smaller than $2^{18v^2+\log_2 v+\log_2 3} = 3v \cdot 2^{18v^2}$, where $v = n^2 m$.*

Proof Let L be the language accepted by $\mathcal{M}$. We also consider the following two languages L_h and L_v:

- L_h is the set of strings accepted by the computations of $\mathcal{M}$ which visit at least one surface pair having a horizontal loop.
- L_v is the set of strings accepted by the computations of $\mathcal{M}$ which visit only surface pairs that do not have horizontal loops.

Clearly, the language L accepted by $\mathcal{M}$ is the union of L_h and L_v.

According to Theorem 3, all strings in L_h are accepted in constant height. More precisely, from $\mathcal{M}$ we can build a unary PDA $\mathcal{M}_h$ which accepts L_h by simulating $\mathcal{M}$ and by accepting when the simulated computation is accepting and visits at least one surface pair having a horizontal loop, which can be decided according to Lemma 1.

To implement $\mathcal{M}_h$, we double the cardinality of the state set, in order to remember if some surface pair having a horizontal loop has been reached during the computation. That is, for each state q we create a copy q'. Thus, the simulation is straightforward but, *after* visiting a surface pair having a horizontal loop, $\mathcal{M}$ switches to q' instead of q. Hence, the final state of $\mathcal{M}_h$ is q'_F. From $\mathcal{M}_h$, we can obtain an equivalent grammar in binary normal form with $(2n)^2 m$ variables. However, in such a grammar, the triples $[q'Ap]$, where q and p are states of $\mathcal{M}$, cannot generate any string (in fact, once a pair having a horizontal loop is reached, the computation of $\mathcal{M}_h$ can only visit states in the copy of Q). This allows to reduce the number of variables to $3n^2 m = 3v$. According to Theorem 3, each string in L_h can be accepted using height smaller than $2^{2(3v)^2+\log_2(3v)} = 2^{18v^2+\log_2 v+\log_2 3}$.

If the set $L_v \setminus L_h$ is infinite, then it should contain arbitrarily long strings; by Lemma 10, an arbitrarily high pushdown is required to accept them.

Otherwise, when $L_v \setminus L_h$ is finite, $\mathcal{M}$ accepts in constant height, which is bounded by the maximum between the height used to accept strings in L_h and the height used to accept strings in $L_v \setminus L_h$. To estimate the latter amount, first we notice that L_v is accepted by a PDA $\mathcal{M}_v$, which can be obtained by just removing from $\mathcal{M}$ all the transitions defined from surface pairs $[rB]$ having horizontal loops. Hence, L_v is generated by a context-free grammar in binary normal form with $v = n^2 m$ variables. According to Lemma 2, from $\mathcal{M}_h$ and $\mathcal{M}_v$, we obtain equivalent DFAs with less than 2^{9v^2} and 2^{v^2} states, respectively. From them, using a standard product construction, we can obtain a DFA with less than 2^{10v^2} states accepting $L_v \setminus L_h$. Since such a language is finite, the length of each string in it is less than the number of states of such a DFA, i.e., it is bounded by 2^{10v^2}. By Lemma 9, this implies that each string in $L_v \setminus L_h$ is accepted using height bounded by $v2^{10v^2}$, which is lower than the bound we obtained for strings in L_h. By summarizing, we can conclude that if $\mathcal{M}$ accepts in constant height, then it accepts in height smaller than $2^{18v^2+\log_2 v+\log_2 3}$. □

Theorem 5 *It is decidable whether a unary* PDA *accepts in constant height.*

Proof As seen in the proof of Theorem 4, the language accepted by a PDA $\mathcal{M}$ is the union of the languages L_h (composed by the strings accepted by the computations of $\mathcal{M}$ which visit

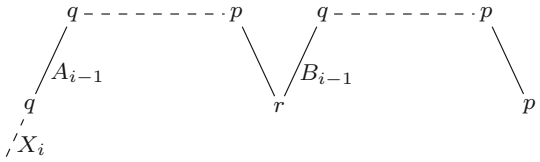

Fig. 2 The evolution of the pushdown store of $\mathcal{M}_k$ in a $[q\,X_i\,p]$-computation, where the symbol X_i which is initially on the top of the pushdown is either A_i, $i = 1, \ldots, k$, or B_i, $i = 1, \ldots, k-1$. The horizontal dashed lines should be replaced by a move reading one input symbol, when $i - 1 = 0$, and by the same pattern, in the other cases

at least one surface pair having a horizontal loop) and L_v (composed by the strings accepted by the computations of $\mathcal{M}$ which visit only surface pairs that do not have horizontal loops). Since $\mathcal{M}$ accepts in constant height if and only if $L_v \setminus L_h$ is finite, this problem reduces to deciding the finiteness of $L_v \setminus L_h$, which is computable because L_v and L_h are both unary context-free languages and hence regular. □

5 Size versus height in the unary case

The arguments used in Sect. 4 to prove that it is decidable whether a unary PDA accepts in constant height give an exponential upper bound for the maximum pushdown height, with respect to the size of a PDA working in constant height (see Theorem 4). In this section, we prove that such an exponential bound cannot be reduced.

To prove this result, we will make use of some modifications of the PDA described in the following example.

Example 1 Let us consider the language $L_k = \left\{a^{2^k}\right\}$, where $k > 0$ is a given integer. A deterministic PDA $\mathcal{A}_k$ for L_k might work as follows. The automaton can exploit its pushdown to implement the recursive function

$$f(i) = \begin{cases} 1 & \text{if } i = 0, \\ 2f(i-1) & \text{otherwise,} \end{cases}$$

in order to read $f(k) = 2^k$ input symbols. To this aim, it uses the state set $Q = \{q, r, p\}$, and the pushdown alphabet $\Gamma = \{A_0, A_1, \ldots, A_k, B_0, B_1, \ldots, B_{k-1}\}$.

One call to $f(i)$ is implemented by a $[q\,X_i\,p]$-computation, with $X_i \in \Gamma$. For $i = 0$, such a computation consists of one move which reads one input symbol (Transitions 1 or 2). Otherwise, the computation is split into two parts, both consuming 2^{i-1} input symbols, as depicted in Fig. 2:

- a $[q\,X_i\,r]$-computation that activates, by a recursive call, one $[q\,A_{i-1}\,p]$-computation (Transitions 3 or 4, and 7),
- a $[r\,X_i\,p]$-computation that activates, by a recursive call, one $[q\,B_{i-1}\,p]$-computation (Transitions 5 or 6, and 8).

In this way, a $[q\,A_k\,p]$-computation consumes the string a^{2^k}. Hence, to recognize L_k the automaton starts in the state q with A_k on the pushdown and accepts in the state p.

Formally, $\mathcal{A}_k = \langle Q, \{a\}, \Gamma, \delta, q, A_k, p\rangle$, where the transitions are:

1. $\delta(q, a, A_0) = (p, -)$;

2. $\delta(q, a, B_0) = (p, -)$;
3. $\delta(q, \varepsilon, A_i) = (q, \mathtt{push}(A_{i-1}))$, for $i = 1, \ldots, k$;
4. $\delta(q, \varepsilon, B_i) = (q, \mathtt{push}(A_{i-1}))$, for $i = 1, \ldots, k-1$;
5. $\delta(r, \varepsilon, A_i) = (q, \mathtt{push}(B_{i-1}))$, for $i = 1, \ldots, k$;
6. $\delta(r, \varepsilon, B_i) = (q, \mathtt{push}(B_{i-1}))$, for $i = 1, \ldots, k-1$;
7. $\delta(p, \varepsilon, A_i) = (r, \mathtt{pop})$, for $i = 0, \ldots, k-1$;
8. $\delta(p, \varepsilon, B_i) = (p, \mathtt{pop})$, for $i = 0, \ldots, k-1$.

We point out that the size of $\mathcal{A}_k$ is linear in the parameter k, while the minimum DFA accepting L_k has $2^k + 1$ states. □

We now present the main result of this section, by presenting a family of PDAs accepting in height which is constant in the input length but exponential with respect to the size of the machines:

Theorem 6 *For each integer $k > 0$, there exists a* PDA $\mathcal{M}_k$ *having a size linear in k and accepting in height which is constant with respect to the input length but exponential in k.*

Proof For each integer $k > 0$, let us consider two automata $\mathcal{A}'_k$ and $\mathcal{A}''_k$, accepting the languages $\left\{a^{2^k}\right\}^*$ and $\left\{a^{2^k+1}\right\}^*$, respectively, obtained by modifying the automaton $\mathcal{A}_k$ of Example 1 as follows:

- $\mathcal{A}'_k$ is obtained by adding to $\mathcal{A}_k$ the transition $\delta(p, \varepsilon, A_k) = (q, -)$ and by choosing q as final state. This allows $\mathcal{A}'_k$ to recognize $\left\{a^{2^k}\right\}^*$ with pushdown height k, using 3 states and a pushdown alphabet of size $2k + 1$. We point out that, from such a definition, each accepting computation of $\mathcal{A}'_k$ visits the surface pair $[q\, A_k]$ which has a horizontal loop.
- $\mathcal{A}''_k$, at the beginning of the computation, guesses how many repetitions of the word a^{2^k+1} are concatenated in the input word. This is done, in a preliminary phase, by pushing one occurrence of the symbol A_k on the store for each guessed repetition (Transitions 9 below). Then, for any such occurrence, $\mathcal{A}''_k$ makes the following operations:
  - it reads one a from the input (Transition 10),
  - it simulates one execution of $\mathcal{A}_k$, using Transitions 1–8 in Example 1,
  - it pops the symbol A_k off the pushdown (Transition 11).

  Formally, $\mathcal{A}''_k = \langle Q'', \{a\}, \Gamma \cup \{Z_0\}, \delta'', q_I, Z_0, s\rangle$, where $Q'' = Q \cup \{q_I, s\}$, and δ'' is a copy of δ with the addition of the following nondeterministic transitions:

  9. $\delta''(q_I, \varepsilon, X) = \{(q_I, \mathtt{push}(A_k)), (s, -)\}$, for $X \in \{Z_0, A_k\}$;
  10. $\delta''(s, a, A_k) = \{(q, -)\}$,
  11. $\delta''(p, \varepsilon, A_k) = \{(s, \mathtt{pop})\}$.

  Notice that $\mathcal{A}''_k$ has 5 states and $2k + 2$ pushdown symbols. Furthermore, the pushdown height used to accept the string $a^{\beta(2^k+1)}$ is $\beta + k$. So, $\mathcal{A}''_k$ does not accept in constant height.

It is easy to see that the automaton $\mathcal{M}_k$ obtained by concatenating the automata $\mathcal{A}'_k$ and $\mathcal{A}''_k$ using standard techniques (after renaming the states in such a way that the two sets of states are disjoint) recognizes the language

$$H_k = \{a^t \mid t = \alpha 2^k + \beta(2^k + 1),\ \alpha, \beta \geq 0\}$$

and has 8 states and a pushdown alphabet of $2k + 2$ symbols.

By construction, the first part of each accepting computation of $\mathcal{M}_k$ is an accepting computation of $\mathcal{A}'_k$ which, as above observed, visits a surface pair having a horizontal loop. Hence, from Theorem 3 it follows that $\mathcal{M}_k$ accepts in constant height, with respect to the input length.

We now prove that a height exponential in k is necessary.

Let us consider the string $a^t \in H_k$ obtained by choosing $\alpha = 0$ and $\beta = 2^k - 1$, namely $t = (2^k - 1)(2^k + 1) = 2^{2k} - 1$. We are going to prove that there is only one accepting computation on a^t.

To this aim, we observe that, due to the structure of $\mathcal{M}_k$, for each accepting computation on a^t, there should exist two integers $\alpha', \beta' \geq 0$, such that $t = 2^{2k} - 1 = \alpha' 2^k + \beta'(2^k + 1)$, from which $2^{2k} - 1 - \beta' = 2^k(\alpha' + \beta')$ and $2^{2k} - 2^k(\alpha' + \beta') = \beta' + 1$. So, 2^k should divide $\beta' + 1$. The only possible solution of $t = \alpha' 2^k + \beta'(2^k + 1)$ is obtained by taking $\alpha' = \alpha = 0$ and $\beta' = \beta = 2^k - 1$. In fact, this solution corresponds to the smallest $\beta' \geq 0$ such that 2^k divides $\beta' + 1$, while $\beta'(2^k + 1) > t$ for any larger multiple β' of 2^k. This allows to conclude that the only accepting computation on a^t is the one in which the simulation of $\mathcal{A}''_k$ uses height $\beta + k$, with $\beta = 2^k - 1$.

Hence, to accept a^t an exponential height, with respect to the size of $\mathcal{M}_k$, is necessary. □

6 An optimal lower bound for nonconstant height

In this section, we turn our attention to PDAs accepting in nonconstant height. First of all, we mention that each nondeterministic Turing machine, with a two-way read-only input tape, which accepts in $o(\log \log n)$ space, where n is the input length and the space is measured by considering the portion of an auxiliary work tape used during the least expensive computation, actually uses only a constant amount of space [1]. Since pushdown automata can be seen as a special case of this kind of machines, as a direct consequence, the height of the pushdown store in any PDA accepting in nonconstant height should be at least $\log \log n$, for infinitely many n's. Furthermore, this lower bound is optimal [6].

We show that in the unary case the optimal bound increases to a logarithmic function.

Let us start by proving the lower bound:

Theorem 7 *Let $\mathcal{M}$ be a unary* PDA *using height $h(n)$. Then, either $h(n)$ is bounded by a constant or there exists $c > 0$ such that $h(n) \geq c \log n$ infinitely often.*

Proof According to the proof of Theorem 4, if $h(n)$ is not constant, then there exist infinitely many strings in $L_v \setminus L_h$ that are accepted only by computations that use vertical loops and do not visit surface pairs having horizontal loops. We are going to prove that to accept all these strings a logarithmic pushdown height is necessary. To this aim, let us consider the PDA $\mathcal{M}_v$ accepting L_v, introduced in the proof of Theorem 4. This PDA is obtained from $\mathcal{M}$ by removing all transitions from surface pairs having horizontal loops. Hence, $\mathcal{M}_v$ uses the same set of states Q and the same pushdown alphabet Γ as $\mathcal{M}$, but accepts without using surface pairs having horizontal loops.

Let us fix an integer $\overline{n}$ such that $a^{\overline{n}} \in L_v \setminus L_h$. We first notice that by the construction of $\mathcal{M}_v$, the sets of computations of $\mathcal{M}$ and $\mathcal{M}_v$ on $a^{\overline{n}}$ coincide. Hence, $\mathcal{M}_v$ accepts $a^{\overline{n}}$ in the same height $h(\overline{n})$ as $\mathcal{M}$. Let us consider the PDA $\mathcal{M}_{h(\overline{n})}$ obtained by bounding the height of the pushdown of $\mathcal{M}_v$ to $h(\overline{n})$, which is a constant since $\overline{n}$ is fixed. In this way, the language accepted by $\mathcal{M}_{h(\overline{n})}$ is a subset of the language accepted by $\mathcal{M}_{h(\overline{n})}$. However, $\mathcal{M}_{h(\overline{n})}$ still

accepts $a^{\overline{n}}$. To obtain $\mathcal{M}_{h(\overline{n})}$, the pushdown alphabet of $\mathcal{M}_v$ is extended in order to keep track of the pushdown height, together with each symbol pushed on the pushdown. Hence, since the only symbol that appears at height 0 is Z_0, the cardinality of the pushdown alphabet of $\mathcal{M}_{h(\overline{n})}$ is bounded by $\#\Gamma \cdot h(\overline{n}) + 1$. According to the construction in Sect. 4.2, $\mathcal{M}_{h(\overline{n})}$ can be converted into an equivalent grammar in binary normal form with $(\#Q)^2 \cdot (\#\Gamma \cdot h(\overline{n}) + 1)$ variables, from which, using Lemma 2, we can obtain an NFA $\mathcal{N}_{h(\overline{n})}$ which is equivalent to $\mathcal{M}_{h(\overline{n})}$, and whose number of states is $2^{O(h(\overline{n}))}$.

Since $\mathcal{M}_{h(\overline{n})}$ has pushdown height bounded by $h(\overline{n})$, it cannot have vertical loops. Furthermore, since accepting computations of $\mathcal{M}_v$ do not use surface pairs with horizontal loops, also accepting computations of $\mathcal{M}_{h(\overline{n})}$ do not use horizontal loops. This allows to conclude that the language accepted by $\mathcal{M}_{h(\overline{n})}$ is finite. Thus, in the equivalent NFA $\mathcal{N}_{h(\overline{n})}$, the string $a^{\overline{n}}$ is accepted by a path without any repeated state. Hence, the number of states of $\mathcal{N}_{h(\overline{n})}$, which we already observed to be $2^{O(h(\overline{n}))}$, must be greater than $\overline{n}$.

To complete the proof, we finally notice that the previous argument can be applied to each string in $L_v \setminus L_h$. Since the cardinality of $L_v \setminus L_h$ is infinite, this allows to conclude that $2^{O(h(n))} > n$ infinitely often, thus implying the existence of a constant c such that $h(n) \geq c \log n$ for infinitely many integers n. □

In the next theorem, we prove a matching lower bound. The language accepted by the PDA we present is a^*. It should be clear that such a PDA is not the best machine for this language: instead of a trivial one-state finite automaton, we use an inefficient PDA which requires an unbounded pushdown store.

Theorem 8 *There exists a unary* PDA *accepting every word* a^ℓ, $\ell > 0$, *using pushdown height exactly* $\lfloor \log_2 \ell \rfloor + 1$ *and the empty word using height* 0.

Proof Consider the PDA $\mathcal{A} = \langle Q, \{a\}, \Gamma, \delta, q_I, Z_0, q_F \rangle$, where $Q = \{q_I, q_1, q_2, q_F\}$, $\Gamma = \{Z_0, 0, 1\}$, and the transition function δ is defined as follows:

1. $\delta(q_I, \varepsilon, X) = (q_F, -)$, for $X \in \Gamma$;
2. $\delta(q_I, \varepsilon, X) = (q_I, \mathtt{push}(0))$, for $X \in \Gamma$;
3. $\delta(q_F, \varepsilon, 0) = (q_1, \mathtt{pop})$;
4. $\delta(q_1, a, X) = (q_2, -)$, for $X \in \Gamma$;
5. $\delta(q_2, \varepsilon, X) = (q_I, \mathtt{push}(1))$, for $X \in \Gamma$;
6. $\delta(q_F, \varepsilon, 1) = (q_F, \mathtt{pop})$.

As a first observation, we point out that from the initial state q_I it is possible to reach the final state q_F with the same pushdown height by using a subroutine implementing a recursive strategy: either an ε-move is performed (Transitions 1), or two recursive calls of the same subroutine, with one read operation between them, are executed. This is done, as depicted in Fig. 3, by pushing 0 on the pushdown, while activating the first recursive call (Transition 2). When such a recursive call ends, 0 is popped off the pushdown (Transition 3), a symbol a is read from the input (Transition 4), and then, 1 is pushed while activating the second call (Transitions 5). Finally, when such a call ends, the symbol 1 is popped off the pushdown (Transitions 6).

Note that each string in a^* is accepted by $\mathcal{A}$.

In order to state how the length of a word and the height of the pushdown used for accepting it are related, let us calculate the *maximal length* $\ell(h)$ of strings consumed by $[q_I X q_F]$-computations with pushdown increment h, for $X \in \Gamma$. We point out that the moves made during such computations do not depend on the symbol X; hence, also $\ell(h)$ does not depend on X.

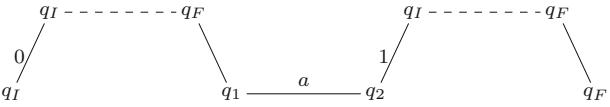

Fig. 3 The evolution of the pushdown store of $\mathcal{A}$ during the recursive subroutine leading from q_I to q_F, when recursive calls are made. The dashed lines should be replaced either by an ε-move or, recursively, by the same pattern

According to the recursive subroutine implemented by $\mathcal{A}$ (see also Fig. 3), we can write the following recurrence:

$$\ell(h) = \begin{cases} 0 & \text{if } h = 0, \\ 2\ell(h-1)+1 & \text{otherwise,} \end{cases}$$

which has solution $\ell(h) = 2^h - 1$. As a consequence, pushdown height h is sufficient to accept all strings of length up to $2^h - 1$. Furthermore, since $\ell(h-1) = 2^{h-1} - 1$ is the maximal length of strings accepted using height $h-1$, we conclude that pushdown height h is necessary and sufficient to accept all strings of length ℓ, with $2^{h-1} \leq \ell < 2^h$. Hence, for $\ell > 0$, the string a^ℓ is accepted using pushdown height exactly $\lfloor \log_2 \ell \rfloor + 1$. □

Funding Open access funding provided by Università degli Studi di Milano within the CRUI-CARE Agreement.

References

1. Alberts, M.: Space complexity of alternating Turing machines. In: Budach , L. (ed.) Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, 1985, Lecture Notes in Computer Science, vol. 199, pp. 1–7. Springer (1985). https://doi.org/10.1007/BFb0028785
2. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung **14**, 143–172 (1961)
3. Bednárová, Z., Geffert, V.: Two double-exponential gaps for automata with a limited pushdown. Inf. Comput. **253**, 381–398 (2017). https://doi.org/10.1016/j.ic.2016.06.005
4. 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
5. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Boolean language operations on nondeterministic automata with a pushdown of constant height. J. Comput. Syst. Sci. **90**, 99–114 (2017). https://doi.org/10.1016/j.jcss.2017.06.007
6. Bednárová, Z., Geffert, V., Reinhardt, K., Yakaryilmaz, A.: New results on the minimum amount of useful space. Int. J. Found. Comput. Sci. **27**(2), 259–282 (2016). https://doi.org/10.1142/S0129054116400098
7. Chomsky, N.: A note on phrase structure grammars. Inf. Control **2**(4), 393–395 (1959). https://doi.org/10.1016/S0019-9958(59)80017-6
8. 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
9. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. J. ACM **9**(3), 350–371 (1962). https://doi.org/10.1145/321127.321132

10. Guillon, B., Pighizzini, G., Prigioniero, L.: Non-self-embedding grammars, constant-height pushdown automata, and limited automata. Int. J. Found. Comput. Sci. **31**(8), 1133–1157 (2020). https://doi.org/10.1142/S0129054120420071
11. Hartmanis, J.: Context-free languages and Turing machine computations. In: Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics, vol. 19, pp. 42–51. American Mathematical Society (1967)
12. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, London (1979)
13. Malcher, A., Meckel, K., Mereghetti, C., Palano, B.: Descriptional complexity of pushdown store languages. J. Autom. Lang. Comb. **17**(2–4), 225–244 (2012). https://doi.org/10.25596/jalc-2012-225
14. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. **30**(6), 1976–1992 (2001). https://doi.org/10.1137/S009753979935431X
15. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: 12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971, pp. 188–191. IEEE Computer Society (1971). https://doi.org/10.1109/SWAT.1971.11
16. 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). https://doi.org/10.1006/jcss.2002.1855
17. Rado, T.: On non-computable functions. Bell Syst. Technol. J. **41**(3), 877–884 (1962). https://doi.org/10.1002/j.1538-7305.1962.tb00480.x
18. Stearns, R.E.: A regularity test for pushdown machines. Inf. Control **11**(3), 323–340 (1967). https://doi.org/10.1016/S0019-9958(67)90591-8