Pushdown Automata and Constant Height: Decidability and Bounds

Giovanni Pighizzini, Luca Prigioniero

Pushdown Automata and Constant Height: Decidability and Bounds

Giovanni Pighizzini and Luca Prigioniero

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

Abstract. It cannot be decided whether a pushdown automaton accepts using constant pushdown height, with respect to the input length, or not. Furthermore, in the case of acceptance in constant height, the height cannot be bounded by any recursive function in the size of the description of the machine. In contrast, in the restricted case of pushdown automata over a one-letter input alphabet, i.e., unary pushdown automata, the above property becomes decidable. Moreover, if the height is bounded by a constant in the input length, then it is at most exponential with respect to the size of the description of the pushdown automaton. This bound cannot be reduced. 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 only regular languages. Despite their limited computational power, constant-height pushdown automata are interesting since they allow more succinct representations of regular languages than finite automata [5]. A natural generative counterpart of these devices are *non-self-embedding context-free grammars*, roughly context-free grammars without "true" recursion [4], which have been recently showed to be polynomially related in size to constant-height pushdown automata [7].

In this paper, we focus on standard pushdown automata, namely with an unrestricted pushdown store, that, however, are able to accept their inputs by making use only of a constant amount of the pushdown store. More precisely, we say that a pushdown automaton $\mathcal{M}$ *accepts in constant height* h, for some given h, if for each word in the language accepted by $\mathcal{M}$ there exists 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.

It is a simple observation that a pushdown automaton $\mathcal{M}$ accepting in constant height h can be converted into an equivalent constant-height pushdown automaton: in any configuration it is enough to keep track of the current height in order to stop and reject when a computation tries to exceed the height limit. The description of the resulting constant-height pushdown automaton has size polynomial in h and in 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.

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 can be 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 recently obtained in [3]. As a consequence of the constructions presented in the second part of the paper, we are able to prove that in the unary case this lower bound is logarithmic. We also show that it cannot be further increased.

For brevity reasons, many of the proofs are only outlined in this version of the paper.

2 Preliminaries

We assume the reader familiar with the standard notions from formal language and automata theory as presented in classical textbooks, e.g., [9]. 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 $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ where Q is the finite *set of states*, Σ is the *input alphabet*, Γ is the *pushdown alphabet*, $q_0 \in Q$ is the *initial state*, $Z_0 \in \Gamma$ is the *start symbol*, $F \subseteq Q$ is the set of *final states*. 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; this symbol is never pushed on or popped off the stack;
2. the input is accepted if and only if the automaton reaches a final state, the pushdown store contains only Z_0 and all the input has been scanned;
3. if the automaton moves the input head, then no operations are performed on the stack;
4. every `push` adds exactly one symbol on the stack.

Note that the transition function δ of a PDA $\mathcal{M}$ 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 stack, by consuming the input σ, can reach the state p without changing the stack 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 stack, without reading any input symbol, can reach the state p by popping off the stack the symbol A on the top (by pushing the symbol B on the top of the stack, without changing the stack, respectively).

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 start symbol Z_0. Hence, in the initial and in the accepting configurations the height is 0. The height in 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 x, namely, there exists a computation accepting x using pushdown height $h(x)$, and no computations accepting x using height less than $h(x)$. Moreover, if x is rejected then $h(x) = 0$. To study pushdown height with respect to input lengths, we consider the worst case among all possible inputs of the same length. Hence, 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 polynomial in the cardinalities of the set of states and of the pushdown alphabet.[1]

[1] In some papers PDAs are presented in different forms. As pointed out in [2], it is possible to turn the definition of PDAs into these equivalent forms, with a polynomial increase in size and by preserving the property of being constant height.

We now present some technical notions that will be useful in order to state our results. Let $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ 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 stack symbol, namely the only part of the stack 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 an input 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 the input head on the tape cell containing the leftmost symbol of x, and ends in the state p with A on the top of the pushdown at the same height h and the input head on the cell to the right of the cell containing the rightmost symbol of x, without reaching pushdown height less than h in between. We also say that $\mathcal{C}$ *consumes* the input x. We point out that, during $\mathcal{C}$, the symbol A at height h is never replaced and $\mathcal{C}$ does not depend on h and on the symbols in the pushdown store at height less than h. The *stack increment* during $\mathcal{C}$ is the difference between the maximum stack height in $\mathcal{C}$ and the stack 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 $[pA]$.

We denote by $L_{[qAp]}$ the set of input strings consumed in all possible $[qAp]$-computations. By suitably modifying $\mathcal{M}$, we can obtain a PDA accepting $L_{[qAp]}$ which, hence, is context free.

An *horizontal loop* on a surface pair $[qA]$ is any $[qAq]$-computation consuming *at least one input symbol*. 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.

If a $[qAp]$-computation $\mathcal{C}$ contains a proper $[qAp]$-subcomputation $\mathcal{C}'$, for the *same* triple $[qAp]$, which starts with stack higher than at the beginning of $\mathcal{C}$, then the pair $(\mathcal{X}, \mathcal{Y})$ where $\mathcal{X}$ is the prefix of $\mathcal{C}$ ending in the first configuration of $\mathcal{C}'$, and $\mathcal{Y}$ is the suffix of $\mathcal{C}$ starting from the last configuration of $\mathcal{C}'$, is called *vertical loop*. Notice that at the end of $\mathcal{X}$ a nonempty string $A\alpha$ is on the pushdown above the occurrence of A which was on the top at the beginning of $\mathcal{C}$, and such a string is popped off during $\mathcal{Y}$.

It is well known that context-free languages defined over a one-letter alphabet are regular [6]. The size costs of the conversions of unary PDAs and context-free grammars into equivalent finite automata have been studied in [13].

3 Undecidability and Non-Recursive Bounds

In this section we prove that it cannot be decided whether a PDA accepts in constant height or not. Furthermore, the trade-off between the sizes of PDAs accepting in constant height and the maximal heights that are reached by their pushdown stores is not recursive.

These results are proved by using a technique introduced in [8], based on suitable encodings of single-tape Turing machine computations. Roughly, configurations of a such 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, 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 encodes the initial configuration on input w;
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 $\mathcal{T}$.

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

As proved in [8], 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.*[2]

Proof. (outline) 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.

Given an input w, let $\alpha_1, \alpha_2, \ldots$ be the (possibly infinite) sequence of configurations in the computation of $\mathcal{T}$ on w. By adapting the techniques used in [8] to prove the above mentioned result, we describe a PDA $\mathcal{M}_{\mathcal{T},w}$ accepting the complement of the language $partial(\mathcal{T}, w)$ of partial computations of $\mathcal{T}$ on w.

Given an input $\mathcal{D} = \beta_1\$\beta_2^R\$\cdots\$\beta_r^{(R)}$, with $\beta_i \in (Q \cup \Gamma)^*$, $i = 1, \ldots, r$, $\mathcal{M}_{\mathcal{T},w}$ guesses which one among Conditions 1, 2 and 3 is not satisfied. For the first two conditions, the finite control is sufficient. 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 checks the condition by scanning the $(i+1)$th block, if any, while suitably comparing it with the block just saved on the store. (If $i = r$ then the verification fails.) Suppose that $\mathcal{D}$ satisfies Conditions 1 and 2, but not Condition 3. Then, there is a computation which accepts $\mathcal{D}$ using pushdown height equal to the length of the first block $\beta_i^{(R)}$ for which the condition is not satisfied, i.e., the block corresponding to the largest i such that $\beta_j = \alpha_j$ for $j = 1, \ldots, i$. Since the pushdown height used to accept a string x is defined as the *minimum* pushdown height in accepting computations on x, we conclude that the pushdown height used to accept $\mathcal{D}$ is bounded by $|\alpha_i|$. So, if $\mathcal{T}$ halts on w, then the maximum amount of the pushdown store used to accept a string in $(partial(\mathcal{T}, w))^c$ is bounded by the length of the largest configuration reached by $\mathcal{T}$ on w. Otherwise, for each integer h, any string $\alpha_1\$\alpha_2^R\$\cdots\alpha_i\$\beta^R$, where i is odd, $|\alpha_i| \geq h$, $\beta \in \Gamma^*Q\Gamma^*$, and $\beta \neq \alpha_{i+1}$, requires height at least h to be accepted.

Hence, $\mathcal{T}$ halts on w if and only if $\mathcal{M}_{\mathcal{T},w}$ accepts in constant height. □

[2] We point out that for *unambiguous* PDAs, the property is decidable [10].

As already observed in the introduction, any PDA $\mathcal{M}$ accepting in constant height h can be converted into an equivalent constant-height PDA. From such a machine, 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 [5]. 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. (outline) The argument is derived from [12, Prop. 7]. For $n > 0$, let $\mathcal{BB}_n$ be a *busy beaver* with a set of n states Q_n and tape alphabet $\Gamma = \{1, \b{b}\}$, namely a single-tape deterministic Turing machine that, starting with an empty tape, ends the computation with a string on the tape in which the number of 1s, denoted as $\Sigma(n)$, is maximum. It is known that $\Sigma(n)$ cannot be bounded by any recursive function [14]. 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 $\mathcal{M}_n$, whose description has a size polynomial in n, which accepts all the strings over $(Q_n \cup \Gamma \cup \{\$\})^*$ different from $\mathcal{C}_n$, and such that each string different from $\mathcal{C}_n$ is accepted 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. Furthermore, by suitably modifying $\mathcal{C}_n$, 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 $\mathcal{M}_n$. □

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 Section 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 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 horizontal

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

loop. The number of vertical loops which remain in the resulting computation is bounded by a constant. As a consequence, the amount of pushdown store sufficient to accept a^ℓ is also bounded by a constant. In contrast, when all accepting computations on a long string a^ℓ do not visit any surface pair having a horizontal loop, vertical loops and an increasing of the stack cannot be avoided. Hence, the given PDA works in constant height if and only if the cardinality of $L_v \setminus L_h$ 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 we are considering a unary alphabet, languages L_v and L_h are regular. So the finiteness of their difference is decidable. To obtain these results, we refine some of the arguments given in [13] 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

Let $G = \langle V, \Sigma, P, S\rangle$ be a context-free grammar in *binary normal form*, an extension of Chomsky normal form where, besides productions $A \to BC$ and $A \to a$, also *unit productions* $A \to B$ and *ε-productions* are allowed. Let $v = \#V$ be the number of variables of G.

If T is a parse 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 $\alpha \in (V \cup \Sigma)^*$, then we write $T : A \stackrel{\star}{\Rightarrow} \alpha$. 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 from the root to a leaf of T. A *gap tree* 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 $x, y \in \Sigma^*$.

Let us suppose that G is unary, i.e., $\Sigma = \{a\}$. The following lemma will be crucial to obtain the main result of this section. It states that each long enough string a^ℓ in the language generated by G 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 the A-gap trees generating "short" nonempty strings, with A occurring in the derivation of a^ℓ.

Lemma 1. *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 $0 < i + j \leq 2^{v^2} - 2^{v^2 - v}$ 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 $\nu(T_0) = \nu(T)$, $0 \leq \ell_0 \leq 2^{v^2-1} + (2^{v^2} - 2^{v^2-v})^2$, with $k \geq 0$ occurrences of T_A.*

Proof. (outline) First, the tree T is "un-pumped" by removing several gap trees, up to find a tree $T_r : S \stackrel{\star}{\Rightarrow} a^{\ell_r}$, with $\ell_r \leq 2^{v^2-1}$ and $\nu(T_r) = \nu(T)$. The tree T_r is then "re-pumped" to get T_0, by using a number bounded by a constant of occurrences of the removed gap trees. The tree T', which generates the same string as the original T, can be obtained by pumping T_0 with a suitable number of occurrences of T_A. The possibility of doing these transformations, which finally

produce a different tree for the same string, derives from a result related to Diophantine equations [11, Lemma 2.6] and from the fact that in the unary case terminal symbols commute. □

4.2 Simulating vertical loops by a horizontal loop

Let us consider a fixed (not necessarily unary) PDA $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$. We define the grammar $G = \langle V, \Sigma, P, S\rangle$, where the elements of V are triples $[qAp]$, with $q, p \in Q$, $A \in \Gamma$, plus the start symbol S (hence $v = \#V = 1 + (\#Q)^2 \cdot \#\Gamma$), and 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 $(q', push(B)) \in \delta(q, \varepsilon, A)$ and $(p, pop) \in \delta(p', \varepsilon, B)$;
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 $q \in Q$, $A \in \Gamma$;
5. $S \to [q_0 Z_0 q]$, for $q \in F$.

Applying standard techniques, we can prove that G generates the language accepted by $\mathcal{M}$. Since we are interested in the amount of stack used by $\mathcal{M}$, we state such equivalence in a stronger form, which also considers the use of the stack in the computations. In particular, we relate the stack 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 in a path from the root to a leaf in T. As a consequence of a technical lemma, which is omitted, we obtain:

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.*

Let us restrict to the unary case. Using Corollary 2, we can reformulate Lemma 1 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 an 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 on the same input using a bounded amount of pushdown storage.

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 another accepting computation on a^ℓ which uses pushdown height at most $2^{O(v^2)}$.*

Proof. (outline) 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 $[rBr] \in \nu(G)$. 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 $[rBr]$-gap tree $T_{[rBr]}$ deriving a "short" non empty string, i.e., $T_{[rBr]} : [rBr] \stackrel{\star}{\Rightarrow} a^i[rBr]a^j$, with $0 < i+j \leq 2^{v^2} - 2^{v^2-v}$. According to Lemma 1, 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 $\nu(T_0) = \nu(T)$, $0 \leq \ell_0 \leq 2^{v^2-1} + (2^{v^2} - 2^{v^2-v})^2$, with $k \geq 0$ occurrences of $T_{[rBr]}$.

We observe that in the tree T' the k occurrences of $T_{[rBr]}$ could be nested, possibly giving a stack height in 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 $t > 1$ nested occurrences of $T_{[rBr]}$. 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.

Let T'' be the tree obtained after this modification, which still generates a^ℓ. Using Corollary 2 we now estimate the amount of pushdown store used in the computation $\mathcal{C}''$ corresponding to T''. The unit production height of T'' is bounded by the maximum number h_0 of such edges in a path in T_0 plus the maximum number h_1 of such edges in a path in $T_{[rBr]}$ which, in turn, are bounded by the height of T_0 and $T_{[rBr]}$, respectively. We can prove that $h_0 + h_1$ is $2^{O(v^2)}$. According to Corollary 2, this allows us to conclude that a^ℓ is accepted by a computation which uses pushdown height $2^{O(v^2)}$. □

4.3 Decidability

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 bounded by $2^{O(v^2)}$, where $v = n^2m + 1$.*

Proof. (outline) Let L be the language accepted by $\mathcal{M}$. We also consider the following two languages L_h and L_v, whose union gives L:

- 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.

According to Theorem 3, all strings in L_h are accepted in constant height $2^{O(v^2)}$.

If the set $L_v \setminus L_h$ is infinite, then it should contain arbitrarily long strings. It can be verified that an arbitrarily high stack is required to accept them.

Otherwise, $\mathcal{M}$ accepts in constant height. In this case, we evaluate the height of the stack used to accept the strings in $L_v \setminus L_h$. We can modify $\mathcal{M}$ to obtain PDAs $\mathcal{M}_v$ and $\mathcal{M}_h$ accepting languages L_v and L_h, respectively. According to Corollary 2 in [13], these automata can be converted into equivalent DFAs with $2^{O(v^2)}$ states. Hence $L_v \setminus L_h$, which is finite, is accepted by a DFA with less than $2^{O(v^2)}$ states. So, the longest string in $L_v \setminus L_h$ has length at most $2^{O(v^2)}$. This implies that each string in $L_v \setminus L_h$ is accepted by $\mathcal{M}$ using height $2^{O(v^2)}$. □

Corollary 3. *It is decidable whether a unary* PDA *accepts in constant height.*

5 Size versus Height in the Unary Case

The arguments used in Section 4 to prove that it is decidable whether a unary PDA $\mathcal{A}$ accepts in constant height, give an exponential upper bound for the maximum stack height, with respect to the size of $\mathcal{A}$. We can prove that such an exponential bound cannot be reduced:

Theorem 5. *For each integer $k > 0$ there exists a* PDA *$\mathcal{M}_k$ having a size polynomial in k and accepting in height which is constant with respect to the input length but exponential in k.*

Proof. (outline) For each integer $k > 0$, let us consider the language $H_k = \{a^t \mid t = \alpha 2^k + \beta(2^k + 1),\ \alpha, \beta \geq 0\}$. We can define a PDA $\mathcal{M}_k$ of size $O(k)$ which accepts each string in H_k by computations consisting of two parts. In the first part, a horizontal loop which consumes 2^k input symbols is repeated an arbitrary number $\alpha \geq 0$ of times. This uses constant height $O(k)$. In the second part, a vertical loop consuming 2^k+1 input symbols occurs $\beta \geq 0$ times, using height $\beta + k - 1$. According to Theorem 3, each accepting computation can be replaced by an equivalent accepting computation in which the number of occurrences of the vertical loop is bounded by a constant. Hence $\mathcal{M}_k$ accepts in constant height.

However, an height exponential in k is necessary. In fact, let $a^t \in H_k$ be the string obtained by choosing $\alpha = 0$ and $\beta = 2^k - 1$, namely, $t = (2^k - 1)(2^k + 1)$. We can prove that the only solution of the equation $t = \alpha' 2^k + \beta'(2^k + 1)$, with integers $\alpha', \beta' \geq 0$ is $\alpha' = \alpha = 0$ and $\beta' = \beta = 2^k - 1$. This allows to conclude that the only accepting computation on a^t is the one which uses height $\beta + k - 1 \geq 2^k$. Hence, to accept a^t an exponential height, with respect the size of $\mathcal{M}_k$, is necessary. □

6 An Optimal Lower Bound for Non-Constant Height

In this section we turn our attention to PDAs accepting in non-constant height. It is known that in this case the height of the pushdown store should grow at least

as the function $\log\log n$, with respect to the input length n [1]. Furthermore, this lower bound is optimal [3]. We show that in the unary case the optimal bound increases to a logarithmic function.

Let us start by proving the lower bound:

Theorem 6. *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. (outline) 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. Let $\mathcal{M}_v$ be the PDA accepting L_v and, for $a^n \in L_v \setminus L_h$, $\mathcal{M}_{h(n)}$ be the PDA obtained by bounding the height of the pushdown of $\mathcal{M}_v$ to $h(n)$.

Using Corollary 2 in [13], we can prove that there exists an NFA $\mathcal{N}_{h(n)}$ equivalent to $\mathcal{M}_{h(n)}$ whose number of states is bounded by $2^{d \cdot h(n)+1} + 1$, where $d = 2 \cdot (\#Q)^2 \cdot \#\Gamma$.

Since $\mathcal{M}_{h(n)}$ has stack height bounded by $h(n)$, it cannot have vertical loops. Furthermore, accepting computations of $\mathcal{M}_v$ do not use surface pairs with horizontal loops. Hence, the language accepted by $\mathcal{M}_{h(n)}$ is finite. Thus, in $\mathcal{N}_{h(n)}$ the string a^n is accepted by a path without any repeated state. This implies that $\mathcal{N}_{h(n)}$ must have more than n states.

Given any constant c, if $h(n) < c \log n$, then the number of states of $\mathcal{N}_{h(n)}$ would be $2^{d \cdot h(n)+1} + 1 < 2^{d \cdot c \log n + 1} + 1 = 2n^{d \cdot c} + 1$. For c sufficiently small, e.g., $c < 1/(2d)$, we get that the number of states of $\mathcal{N}_{h(n)}$ is less than n, provided that n is not too small. This gives a contradiction. Hence, it must exist a constant c such that $h(n) \geq c \log n$ infinitely often. □

We now prove a matching upper bound:

Theorem 7. *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. (outline) 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 δ defined as follows, for $X \in \Gamma$:

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

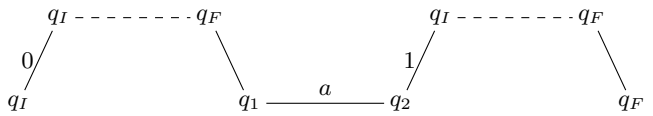

Fig. 1. The evolution of the pushdown store of $\mathcal{A}$ during the recursive subroutine leading from q_I to q_F, when recursive calls are made. The dashed lines should be replaced either by an ε-move or, recursively, by the same pattern.

From the initial state q_I, the PDA $\mathcal{A}$ can reach the final state q_F with the same pushdown height either with an ε-move (Transition 1) or by using a computation path making two recursive calls and consuming one input symbol as depicted in Figure 1. Each string in a^* is accepted by $\mathcal{A}$. Furthermore, pushdown height h is necessary and sufficient to accept all strings of length ℓ, with $2^{h-1} \leq \ell < 2^h$. □

References

1. Alberts, M.: Space complexity of alternating Turing machines. In: Budach, L. (ed.) Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985. Lecture Notes in Computer Science, vol. 199, pp. 1–7. Springer (1985). https://doi.org/10.1007/BFb0028785
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. 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
4. Chomsky, N.: A note on phrase structure grammars. Information and Control **2**(4), 393–395 (1959). https://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. 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
7. Guillon, B., Pighizzini, G., Prigioniero, L.: Non-self-embedding grammars, constant-height pushdown automata, and limited automata. In: Câmpeanu, C. (ed.) CIAA 2018, Proceedings. Lecture Notes in Computer Science, vol. 10977, pp. 186–197. Springer (2018). https://doi.org/10.1007/978-3-319-94812-6_16
8. 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)
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
10. Malcher, A., Meckel, K., Mereghetti, C., Palano, B.: Descriptional complexity of pushdown store languages. Journal of Automata, Languages and Combinatorics **17**(2-4), 225–244 (2012). https://doi.org/10.25596/jalc-2012-225
11. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. **30**(6), 1976–1992 (2001). https://doi.org/10.1137/S009753979935431X
12. 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
13. 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
14. Rado, T.: On non-computable functions. Bell System Technical Journal **41**(3), 877–884 (1962). https://doi.org/10.1002/j.1538-7305.1962.tb00480.x