Information and Computation 306 (2025) 105329

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Pushdown and one-counter automata: Constant and non-constant memory usage ☆

Giovanni Pighizzini [a,*,1], Luca Prigioniero [b]

[a] Dipartimento di Informatica, Università degli Studi di Milano, Italy
[b] Department of Computer Science, Loughborough University, United Kingdom

ARTICLE INFO

Article history:
Received 26 February 2024
Received in revised form 30 June 2025
Accepted 4 July 2025
Available online 9 July 2025

Keywords:
Pushdown automata
Counter automata
Descriptional complexity

ABSTRACT

It cannot be decided whether a one-counter automaton accepts each string in its language using a counter whose value is bounded, with respect to the length of the input, by a constant. Furthermore, when the counter is bounded by a constant, its value cannot be limited by any recursive function in the size of the machine.
By taking into account the costs of all computations (strong measure) or of all accepting computations (accept measure) instead of those of the least expensive accepting computations (weak measure), the above-mentioned problem becomes decidable for both pushdown automata and one-counter automata, while the bounds for the pushdown height or the value of the counter, when non constant, are recursive in the size of the machine.
We also prove that, under the weak measure, if a one-counter automaton accepts with a counter that, with respect to the input length, is not bounded by any constants, then the counter grows at least as a logarithmic function. This is in contrast with the case of pushdown automata in which the bound is a double-logarithmic function. For the strong and accept measures these bounds are shown to be linear, for both pushdown and one-counter automata.

1. Introduction

The relative succinctness of devices describing a same class of languages is a classical topic in formal languages and automata theory and, for sure, it is the central topic in the field of descriptional complexity.

In this respect, many results have been obtained for devices describing the class of regular languages. Among them, besides several variants of finite automata (as one-way and two-way, deterministic and nondeterministic), *constant-height pushdown automata* have been considered (see, e.g., [1–4]). These devices are pushdown automata that can accept each word in their languages by making use of a constant (namely not depending on the input length) amount of pushdown store. This allows to simulate these devices by finite automata. However, for some languages, constant-height pushdown automata can be significantly smaller than nondeterministic finite automata [1].

In [5], we investigated the problem of deciding whether a pushdown automaton accepts in constant height. Namely, given a pushdown automaton $\mathcal{M}$, the question is to decide whether there exists a constant h such that any input string in

☆ Extended version of a paper presented at the *25th International Conference on Descriptional Complexity of Formal Systems (DCFS 2023)*, July 4–6, 2023, Potsdam, Germany [*Lectures Notes in Computer Science*, 13918, pp. 146–157, Springer, 2023].

* Corresponding author.
E-mail addresses: pighizzini@di.unimi.it (G. Pighizzini), l.prigioniero@lboro.ac.uk (L. Prigioniero).

[1] Member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

https://doi.org/10.1016/j.ic.2025.105329

the language accepted by $\mathcal{M}$ has an accepting computation in which the height of the pushdown store is at most h. We proved that this problem, in general, is undecidable. Furthermore, for pushdown automata accepting in constant height, it does not exist any recursive function bounding the height of the pushdown store with respect to the size of the description of the machine. We also proved that, in the restricted case of pushdown automata over a one-letter input alphabet, the above-mentioned problem is decidable. Furthermore, under the same restriction, in the case of acceptance in constant height, the height can be at most exponential in the size of the machine and, in the worst case, it cannot be reduced.

In this paper we investigate another restriction, by considering *one-counter automata*. These devices are pushdown automata in which the pushdown alphabet is restricted to one symbol, plus another symbol used only to indicate the bottom of the pushdown. Hence, for these devices the pushdown height can be seen as the value of a non-negative counter, which in one move can be incremented, decremented (if not negative), or left unchanged. It is also possible to test whether the counter contains zero. It is easy to observe that the class of languages accepted by these devices properly contains the class of regular languages and it is properly included in that of context-free languages. We prove that the above-mentioned problem remains undecidable for these devices, namely, given a one-counter automaton it cannot be decided if there exists a constant h such that any accepted string has an accepting computation in which the value of the counter is bounded by h. Furthermore, in the case of a bounded counter, its value cannot be upper limited by any recursive function of the size of the machine. So, we have a non-recursive trade-off.

We point out that all the above-mentioned results refer to the so-called weak measure, where the cost of the least expensive accepting computation is considered [6]. This is related to the idea of nondeterminism that, among all possible computations, allows to choose not only an accepting one, but also the *least expensive accepting computation*. However, it is quite natural to ask what happens if we consider the maximum cost among *all* computations (strong measure) or among *all accepting* computations (accept measure). We prove that for both these measures, in contrast with the weak one, the problem of deciding whether a pushdown automaton accepts in constant height is decidable. Hence also the corresponding problem for one-counter automata is computable. Furthermore, we are able to prove an upper bound for the height, with respect to the size of the machine, in the case of acceptance in constant height.

These results are obtained by using a technique which is partially inspired by [7] and reduces the investigation for these measures to the case of deterministic pushdown automata considered in [3]. Essentially, we extend the input alphabet with symbols encoding the transitions used during a computation in such a way that the original machine, modified to work on this extended alphabet, becomes deterministic, but makes the same use of the pushdown store as the original one.

We also study how much the height of the pushdown grows, with respect to the length of the input, when it is not constant. For the weak measure we already proved a double-logarithmic optimal lower bound [5]. In the case of one-counter automata here we increase such a lower bound to a logarithmic function, also showing that it can be reached. We further increase it to a linear function in the case of one-counter automata accepting unary languages. For the strong and accept measures, we prove that the lower bound, in both cases of pushdown and one-counter automata, is linear, regardless the cardinality of the input alphabet. Even these results are obtained by the above-mentioned technique that reduces the investigation to the deterministic case, and by making use of some results on deterministic pushdown automata from [8].

The paper is organized as follows. After presenting in Section 2 the fundamental notations and notions, together with the main computational models considered in the paper and the complexity measures for the height of the pushdown store and for the counter, in Section 3 we give some preliminary examples. Section 4 is devoted to proving our undecidability result and non-recursive trade-off for one-counter automata. In Section 5 we obtain optimal lower bounds for the value of the counter, when not constant, in the cases of binary and unary alphabets.

In the remaining part of the paper we enlarge our attention to pushdown automata. In Section 6 we introduce the *extended language* of a pushdown automaton and we study its fundamental properties. These properties are used to prove in Section 7 that, by replacing the weak measure by either the accept or the strong one, the property that has been proved undecidable in Section 4 for one-counter automata becomes decidable even for pushdown automata. The same happens for the non-recursive trade-off. Furthermore, under the accept or the strong measures, the lower bounds for the counter and for the height of the pushdown store are shown to be higher than under the weak measure. We present some final considerations in Section 8.

2. Preliminaries

We assume the reader familiar with the standard notions from formal languages and automata theory as presented in classical textbooks, e.g., [9,10]. As customary, given a set S we denote by $\#S$ its cardinality and by 2^S the powerset of S. The length of a string x is denoted by $|x|$ and the empty string by ε.

We now 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*. 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 scans the first input symbol, and 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 symbols have 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

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

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, or without changing the pushdown, respectively).

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 non-accepting configuration.

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 in addition to 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, by encoding the pushdown contents in the finite state control. So the language accepted by it is regular.

We point out that the above-given definition of acceptance in constant height considers the minimum height which is sufficient to accept inputs of length n. This measure is called weak. In contrast, by taking into account the costs of *all computations*, we obtain the strong measure. We can also give an intermediate measure, called accept, that considers the costs of *all accepting computations*. By summarizing, $\mathcal{M}$ works in strong (resp., accept) height $h(n)$ if any computation (resp., any accepting computation) on any input of length n uses at most height $h(n)$. For more details on these notions see, e.g., [6]. When not explicitly specified, the weak measure is assumed.

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 [10] 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 [2].

Given a pushdown automaton, a *configuration* is a triple (q, w, γ) describing the status of the machine in a given instant, where $q \in Q$ is the state in the finite control, $w \in \Sigma^*$ is the suffix of the input which has not been yet read, namely that starts on the head position, $\gamma \in \Gamma^*$ is the content of the pushdown store, written from bottom to top. We use the standard notations to denote reachability between configurations. Hence, $C' \vdash C''$ (resp., $C' \vdash^* C''$) denotes that from the configuration C' in one step (resp., in some steps), the configuration C'' can be reached.

An *internal configuration* is a pair describing the internal status of the machine, without the input. Hence, the internal configuration corresponding to (q, w, γ) is just the pair (q, γ).

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 on the cell to the right of the rightmost symbol 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. Notice that the surface pairs at the beginning and at the end of $\mathcal{C}$ are $[qA]$ and $[pA]$, respectively.

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. As observed in [5, Lemma 1], it is decidable if a surface pair $[qA]$ has a horizontal loop.

If a $[qAp]$-computation $\mathcal{C}$ 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 $\mathcal{C}$, then the pair $(\mathcal{X}, \mathcal{Y})$ is called *vertical loop*. Note that, during the execution of $\mathcal{X}$, a nonempty string αA 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 *one-counter automaton* (OCA, for short) is a PDA where the working alphabet Γ consists of the bottom symbol Z_0, which can occur only at the bottom of the pushdown store, plus another symbol Z, in such a way that the string on the pushdown store has always the form $Z_0 Z^h$, where $h \geq 0$ is the height. We also say that h is the *value of the counter*. All the notions and notations for pushdown automata can be used for one-counter automata. However, in configurations and internal configurations we can describe the pushdown content just by giving the value of the counter.

When we write that an OCA $\mathcal{A}$ accepts using a *constant counter*, we mean that there exists a constant $H \geq 0$ such that for each string x accepted by $\mathcal{A}$ there exists a computation accepting x in which the value of the counter is always bounded by H. We can extend this notion in an obvious way, to define acceptance using a logarithmic or a linear counter, with respect to the input length. Notice that it is enough to have one accepting computation satisfying the bound. Hence, we implicitly refer to the weak measure. When we want to take into account the value of the counter on all accepting computations or all computations, regardless acceptance, we will explicitly mention that we are using the accept or the strong measure, respectively.

3. Some preliminary examples

This section is devoted to presenting some examples of languages and of one-counter automata accepting them. The first purpose is to illustrate how one-counter automata work and to show the difference between weak, accept, and strong measures. Furthermore, these examples will be used in the next sections to state our results.

Let us consider the set

$$L = \{x_1\$x_2\$\cdots\$x_m\$ \mid m > 1,\ x_i \in \{0,1\}^* \text{ for } i = 1,\ldots,m, \text{and there exists } j, 1 \leq j < m, \text{ s.t. } x_j \neq x_{j+1}\}$$

of strings composed by binary (possibly empty) factors interleaved by special markers \$, in which there are at least two different subsequent factors. In the next result we prove that there exists an OCA accepting this language.

Lemma 1. *The language L is a one-counter language.*

Proof. We describe how an OCA $\mathcal{N}$ can accept L. The device $\mathcal{N}$ nondeterministically selects one block x_j, $1 \leq j \leq m$, by skipping, while moving the input head, the previous blocks. In this phase the counter is not used. Inside x_j, $\mathcal{N}$ has to guess a position for which x_j and x_{j+1} (if $j < m$) contain different symbols. To do that, $\mathcal{N}$ moves the head within x_j by incrementing the counter until reaching a position where it makes such a guess. At that moment $\mathcal{N}$ saves the input symbol σ, read by the head, in its finite control. Then, it moves the head to the right of the next \$ without changing the counter, and makes the verification. If $j = m$, then the verification fails immediately. If $j < m$, the device scans the initial part of the $(j+1)$-st block while decrementing the counter, and when the counter value reaches 0, it compares the scanned symbol with σ. If they are different, then $\mathcal{N}$ moves the head until reaching the end of the input and accepts. If, during the verification, the machine encounters a \$ delimiting the end of x_{j+1} while the value of the counter is greater than 0 (meaning that $|x_j| > |x_{j+1}|$), then the device decrements the counter until the value is 0, moves the head to the right until reaching the end of the input and accepts. It could also be possible that $|x_j| < |x_{j+1}|$ with x_j that is a proper prefix of x_{j+1}. To handle this case, $\mathcal{N}$ can guess, as position for which two subsequent blocks are different, the one of the delimiter \$ immediately to the right of x_j. The verification procedure remains the same. □

We briefly discuss how large can be the counter used by the OCA $\mathcal{N}$ described in the previous proof. We observe that in *each computation* of $\mathcal{N}$ the value of the counter is bounded by the length of the input. Furthermore, for each $k > 0$, in

[2] The symbol A being at the top.

the *only accepting computation* on input $0^k\$0^{k-1}1\$$, the value reached by the counter is k. This implies that, under all weak, accept, and strong measures, a counter linear in the length of the input is used.

Notice that the result in Lemma 1 is easily extensible to one-counter automata with no test for zero. In fact, after saving the position of σ within x_j in the counter and reaching the delimiter \$, the machine can locate the corresponding symbol of x_{j+1} by moving the head forward while decrementing the counter and nondeterministically guessing whether the counter value is equal to zero. Then the execution continues without modifying the counter until the end of the input is reached. The device accepts if and only if the value of the counter is zero. More precisely, if during the guess the machine tries to decrement the counter when it contains the value 0 (and so, after having passed the symbol to compare with σ), the computation fails. Otherwise, when the whole input is scanned and the value of the counter is greater than 0 (therefore, after having compared a symbol preceding the one to compare with σ), the machine does not accept.

Gabarro presented a language accepted using a logarithmic counter under the weak measure [8]. The language was derived by modifying a language in [11]. Here, we discuss a minor variation of it:

$$L_{bin} = \{x_1\$x_2\$\cdots\$x_m\$ \mid m \geq 1,\ x_i \in \{0,1\}^*,\ \text{for } i = 1,\ldots,m,\ \text{and } \exists j,\ 1 \leq j \leq m, \text{ s.t. } x_j \text{ is } \textit{not} \text{ the binary representation of } j\}.$$

Lemma 2. *The language L_{bin} is accepted by a one-counter automaton with a non-constant counter bounded by $O(\log n)$.*

Proof. First of all, we observe that a string $w = x_1\$x_2\$\cdots\$x_m\$$, with $x_i \in \{0,1\}^*$, $i = 1,\ldots,m$, belongs to L_{bin} if either $x_1 \neq 1$ or there exists an index j, $1 < j \leq m$, such that x_j does not represent the successor of x_{j-1} in the binary notation.

An OCA $\mathcal{N}_{bin}$ can accept L_{bin} by nondeterministically selecting a block x_j, $1 \leq j \leq m$. If $j = 1$ then, using the finite control, $\mathcal{N}_{bin}$ checks whether $x_1 \neq 1$. Otherwise, it checks that x_j does not represent the successor of x_{j-1}. We now explain how this can be done.

First suppose that x_{j-1} contains the symbol 0, namely $x_{j-1} = y01^k$ with $y \in \{0,1\}^*$ and $k \geq 0$. Then $\mathcal{N}_{bin}$ has to check that $x_j \neq y10^k$. To this aim, $\mathcal{N}_{bin}$ first can guess and verify, using the counter, that $|x_{j-1}| \neq |x_j|$. In this case the machine accepts. Otherwise, suppose $x_{j-1} = a_1\cdots a_s$ and $x_j = b_1\cdots b_s$, with $a_\ell, b_\ell \in \{0,1\}$, $\ell = 1,\ldots,s$. $\mathcal{N}_{bin}$ choses one position h, $1 \leq h \leq s$, guessing that b_h *does not correspond* to a_h in the representation of successor of x_{j-1}. The counter allows to locate the symbols in the same position h in both blocks. Exactly one of the following three cases are possible; the choice among them can be done using the finite state control, according to the structure of the string $z = a_h a_{h+1}\cdots a_s$. As the machine has to detect a mismatch between a_h and b_h, if the outcome of the test is positive, then it accepts:

- if $z \in 1^+$, then the machine tests whether $b_h = 1$,
- if $z \in 01^*$, then the machine tests whether $b_h = 0$,
- if $z \in z'01^*$ with $z' \neq \varepsilon$, then machine tests whether $b_h \neq a_h$.

With some minor modifications, the case $x_{j-1} = 1^k$, in which the machine has to verify that $x_j \neq 10^k$, can be handled.

We observe that if $w = x_1\$x_2\$\cdots\$x_m\$ \in L_{bin}$ then there exists an accepting computation which guesses the *smallest* j, $1 \leq j \leq m$, such that x_j is not the binary representation of j. If $j = 1$ then such a computation does not use the counter. Otherwise it uses a counter which is bounded by $|x_{j-1}|$ which, being x_{j-1} the binary representation of $j-1$, is logarithmic in j. Finally, since input length is $n = |w| \geq m \geq j$, we conclude that $\mathcal{N}_{bin}$ uses an $O(\log n)$ counter. □

By Lemma 2, under the weak measure a logarithmic counter is enough to accept the language L_{bin}. Furthermore, since L_{bin} is not regular, it cannot be accepted using a constant counter. In Section 5, we will prove that if a one-counter automaton uses a non-constant counter, then the counter grows at least as a logarithmic function of the input length.

We now are going to consider the other measures. First of all, we observe that, in each computation of the OCA $\mathcal{N}_{bin}$ described in the proof of Lemma 2, the value of the counter is bounded by the input length. Furthermore, for each $k > 0$, the string $0^k\$0\$$ belongs to the language L_{bin}. The OCA $\mathcal{N}_{bin}$ can accept it without using the counter, in the computation that guesses $j = 1$ and verifies that the first block is different than 1. However, $\mathcal{N}_{bin}$ could also guess $j = 2$ and, if $k > 1$, accept the same string by verifying that the lengths of the first and second blocks are different. In such accepting computation the value of the counter is k, namely it is linear in the input length. So, if we consider the cost of *all* accepting computations, namely the accept measure, the counter is linear. Since we observed that in each computation, the value of the counter is bounded by the length of the input, we can also conclude that even under the strong measure the counter is linear.

Remark 1. It can be observed that the machines $\mathcal{N}$ and $\mathcal{N}_{bin}$ in the previous proofs are finite-turn (1-turn) OCAs. Moreover, they are OCAs operating under the so-called *restriction* $C1$ of [12]: roughly, one-counters in which the machine is nondeterministic until the first pop move.

Finally, note that the technique shown in the proof of Lemma 1 can be adapted to accept some particular cases or modifications of L, for example the languages $\{ww \mid w \in \{0,1\}\}^c$ or $\{ww^R \mid w \in \{0,1\}\}^c$, where w^R denotes the reversal of w, which are known to be context free. Hence, these languages are one counter as well.

4. Undecidability and non-recursive bounds

In this section we show that the undecidability result proved in [5] for pushdown automata holds also if we restrict ourselves to one-counter automata. In particular, the problem of whether a given 1CA $\mathcal{M}$ accepts with the value of the counter bounded by a constant is not decidable. In addition, with respect to the size of $\mathcal{M}$ that does accept with a constant counter, neither the maximal value of the counter of $\mathcal{M}$ nor the number of states of the minimum finite automaton equivalent to $\mathcal{M}$ can be bounded by any recursive function.

These results are proven by refining a technique introduced by Hartmanis based on the fact that the complement of the language composed by suitable encodings of single-tape Turing machine computations is context free [13]. The language we define is a modification of the one used by Hartmanis. Roughly, configurations of a single-tape Turing machine $\mathcal{T}$ with state set Q and tape alphabet Γ are denoted in the standard way as strings in $\Gamma^* Q \Gamma^*$. A computation is encoded as a string $\alpha_1\$\alpha_2\$\ldots\$\alpha_m\$$, where the blocks α_i, $i = 1, \ldots, m$, are encodings of configurations of $\mathcal{T}$ and $\$ \notin Q \cup \Gamma$ is a delimiter. Hence, the (encoding of a) *valid computation* of $\mathcal{T}$ on input w is a string $\alpha_1\$\alpha_2\$\cdots\$\alpha_m\$$, 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_m is a halting configuration of $\mathcal{T}$, namely a configuration from which no move is possible;
4. α_{j+1} is reachable in one step from α_j, $j = 1, \ldots, m-1$.

Following Hartmanis' idea, to prove our undecidability result, we show that the set of all the strings not representing valid computations of $\mathcal{T}$ on the input word w, denoted $(valid(\mathcal{T}, w))^c$, is a one-counter language. This will allow us to prove the main result of this section.

Lemma 3. *Let $\mathcal{T}$ be a Turing machine and w be a fixed string. Then the set $(valid(\mathcal{T}, w))^c$ is a one-counter language.*

Proof. The language $(valid(\mathcal{T}, w))^c$ is accepted by an 1CA $\mathcal{M}_{\mathcal{T},w}$ using the following strategy. Given $\mathcal{D} = \beta_1\$\beta_2\$\cdots\$\beta_m\$$, with $\beta_j \in (Q \cup \Gamma)^*$, $j = 1, \ldots, m$, in order to decide whether $\mathcal{D} \in (valid(\mathcal{T}, w))^c$, $\mathcal{M}_{\mathcal{T},w}$ guesses which one among Conditions 1, 2, 3, and 4 is not satisfied. For the first three conditions, the verification of the guess is done by only using the finite control. For the last condition, $\mathcal{M}_{\mathcal{T},w}$ can use the technique shown in the proof of Lemma 1. In this case, instead of detecting two consecutive different blocks, the machine checks whether the configuration of $\mathcal{D}$ encoded in the $(j+1)$-st block is not the successor, according to the transition function of $\mathcal{T}$, of the j-th one, namely the one encoded by the block β_j, $1 \leq j < m$, that is nondeterministically chosen by the device at the beginning of the computation. □

We are now ready to prove the main result of this section.

Theorem 1. *It is undecidable whether an* 1CA *accepts with bounded counter.*

Proof. We give a reduction from the blank-tape halting problem, namely the problem of deciding whether a Turing machines halts on an empty tape. 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 end, it is sufficient to modify $\mathcal{T}$ by adding to the tape a track where the machine, between every pair of consecutive moves of the original device, marks a tape cell not yet visited).

It is possible to define an 1CA $\mathcal{M}_{\mathcal{T},\varepsilon}$ accepting the language $(valid(\mathcal{T}, \varepsilon))^c$, with the following property: Given $\mathcal{D} = \beta_1\$\beta_2\$\cdots\$\beta_m\$$, with $\beta_j \in (Q \cup \Gamma)^*$, $j = 1, \ldots, m$, if $\mathcal{D}$ does not satisfy Conditions 1, 2, or 3, then $\mathcal{M}_{\mathcal{T},\varepsilon}$ has an accepting computation that does not use the counter (so the value remains 0), otherwise $\mathcal{M}_{\mathcal{T},\varepsilon}$ has an accepting computation in which the counter is bounded by the length of the first block β_j for which Condition 4 is not satisfied, i.e., the block corresponding to the largest j such that $\beta_k = \alpha_k$ for $k = 1, \ldots, j$, where $\alpha_1, \alpha_2, \ldots$ is the (possibly infinite) sequence of configurations in the computation of $\mathcal{T}$ on ε. Hence a counter bounded by $|\alpha_k|$ is sufficient to accept such a string.

If $\mathcal{T}$ halts on ε in m steps, then the maximum value of the counter sufficient to accept strings in $(valid(\mathcal{T}, \varepsilon))^c$ is equal to $|\alpha_m|$. Otherwise, for each arbitrarily large integer h, we can find an index $j > 0$ such that $|\alpha_j| > h$. To accept a string $\alpha_1\$\alpha_2\$\cdots\$\alpha_j\$\beta\$$, with $\beta \in \Gamma^* Q \Gamma^*$ and β differs from α_{j+1} in the rightmost symbol (of α_{j+1}), it would require a counter equal to $|\alpha_j| > h$.

This allows to conclude that $\mathcal{T}$ halts on input ε if and only if $\mathcal{M}_{\mathcal{T},\varepsilon}$ accepts with constant counter. Hence, it cannot be decided whether an 1CA accepts with the counter bounded by any constant. □

Observe that, also in this case, the result is easily extensible to one-counter automata with no test for zero.

In the general case of pushdown automata, 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 every 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 [1]. As a corollary, in the restricted case of one-counter automata, these bounds are polynomial and exponential in h, respectively, because only a counter from 0 to h is attached to the states. Using an argument derived from [14, Prop. 7], we can show that, however, the value 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 one-counter automaton of size n accepting with counter bounded by a value $H(n)$, which is constant with respect to the length of the input, but exceeds $f(n)$.*

Proof. The argument is derived from [14, Prop. 7]. Let us consider any halting single-tape deterministic Turing machine $\mathcal{T}_n$ with n states. Let $\mathcal{C}_n$ be the encoding of the computation of $\mathcal{T}_n$ on ε.

By adapting the arguments used to prove Theorem 1, we can define a one-counter automaton $\mathcal{M}_n$ which accepts $(valid(\mathcal{T}_n, \varepsilon))^c$ using a counter bounded by the length of the longest configuration occurring in $\mathcal{C}_n$. Since n is fixed, $\mathcal{M}_n$ accepts with constant counter. Moreover, it uses a constant number of states for testing that either one of Conditions 1, 2, and 3 does not hold for $\mathcal{C}_n$ to be a valid computation. For Condition 4, namely to check whether two configurations of $\mathcal{T}_n$ are not reachable in one step, the machine has to compare the parts of configurations representing the cells of the tape different than the head position in the first configuration, which can be done using the counter and a number of states that does not depend on the input, and the part (state and symbol) that is modified according to the transition function of $\mathcal{T}_n$. Each transition can be checked using a constant number of states. As $\mathcal{T}_n$ has n states and its working alphabet is fixed, it has $O(n)$ transitions. Hence, to check Condition 4, $\mathcal{M}_n$ uses $O(n)$ states. So, $\mathcal{M}_n$ has size $O(n^2)$.

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 ε), we can obtain a string that requires a counter equal to the maximal length of configurations occurring in $\mathcal{C}_n$ to be accepted by $\mathcal{M}_n$.

Hence, since the maximal length of configurations occurring in $\mathcal{C}_n$ cannot be bounded by any recursive function in n, this allows to conclude that the counter used by $\mathcal{M}_n$ cannot be bounded by any recursive function in the size of $\mathcal{M}_n$. □

We point out that the question in Theorem 1 is decidable in the cases of *deterministic* and of *unambiguous* PDAs, as well as in the case of PDAs with a *unary input alphabet* [5]. Hence, in all these cases it is decidable for one-counter automata.

5. Counter lower bound for one-counter automata

From the results concerning the minimal space requirements for Turing machines accepting nonregular languages, it turns out that if a PDA does not work in constant height, then the height must grow, with respect to the input length, at least as a double-logarithmic function [15]. Furthermore, there exists a language accepted by a PDA using such a height [16]. Here, we prove that, in the case of one-counter automata, the optimal lower bound is higher. In fact, we show that if a one-counter automaton accepts a language L using an unbounded counter, then the value of the counter must grow, with respect to the input length, at least as a logarithmic function. Furthermore, this bound is optimal for input alphabets of at least two symbols. In the case of a unary input alphabet, we will show that the bound is linear.

In order to prove these results we introduce some auxiliary notations and we state some preliminary lemmas.

Given a string $u \in \Sigma^*$ and an integer $h \geq 0$, let us denote by $Q_h(u)$ the set of *internal configurations* that are reachable after reading u, by a sequence of moves where the value of the counter is *at most* h, namely:

$$Q_h(u) = \{(q, t) \mid t \leq h \text{ and } (q_I, u, 0) \vdash_h^\star (q, \varepsilon, t)\},$$

where $\vdash_h^\star$ indicates that in the sequence of moves under consideration the value of the counter cannot exceed h.

Lemma 4. *If $Q_h(u) = Q_h(v)$ then u and v are indistinguishable for computations with counter bounded by h, namely for each $z \in \Sigma^*$ there exists a computation with counter bounded by h accepting uz if and only if there exists a computation with counter bounded by h accepting vz.*

Proof. Consider a computation accepting uz in which the counter is bounded by h. Let $(q, t) \in Q_h(u)$ be the internal configuration reached after reading u in such a computation. Since $Q_h(u) = Q_h(v)$, from the initial configuration, reading v, it is possible to reach (q, t) and then to read z while making exactly the same moves as in the accepting computation on uz. This gives an accepting computation on vz in which the value of the counter is bounded by h. The converse implication is proved in the same way, by switching u and v. □

In the following, for each integer $k \geq 0$, let us denote by $L[k]$ the set of strings that are accepted by computations of $\mathcal{M}$ with counter bounded by k, but that cannot be accepted by any computation in which the value of the counter never reaches k, i.e.,

$$L[k] = \{w \in \Sigma^* \mid (q_I, w, 0) \vdash_k^\star (q_F, \varepsilon, 0)\} \setminus \{w \in \Sigma^* \mid (q_I, w, 0) \vdash_{k-1}^\star (q_F, \varepsilon, 0)\}.$$

Lemma 5. *If $L[k]$ is not empty then it contains at least one string of length less than $\#Q \cdot (k+1) \cdot 2^{\#Q \cdot k}$.*

Proof. Suppose $L[k] \neq \emptyset$ and consider a string $x \in L[k]$. We are going to prove that if $|x| \geq \#Q \cdot (k+1) \cdot 2^{\#Q \cdot k}$ then $L[k]$ also contains a string x' shorter than x.

Let $x = a_1 a_2 \cdots a_n$, with $n = |x|$ and $a_i \in \Sigma$, $i = 1, \ldots, n$. Consider a computation $\mathcal{C}$ accepting x in which the counter is bounded by k:

$$(q_I, x, 0) = (p_0, a_1 a_2 \cdots a_n, 0) \vdash_k^\star \cdots \vdash_k^\star (p_i, a_{i+1} \cdots a_n, t_i) \vdash_k^\star \cdots \vdash_k^\star (p_n, \varepsilon, t_n) = (q_F, \varepsilon, 0)\,,$$

where (p_i, t_i) is an internal configuration reached after reading $a_1 a_2 \cdots a_i$, $i = 1, \ldots, n$, with $p_0 = q_I$, $p_n = q_F$, $t_0 = t_n = 0$. For $i = 0, \ldots, n$, let us consider the triple $\mathcal{T}_i = \langle p_i, t_i, Q_{k-1}(a_1 a_2 \cdots a_i)\rangle$. The number of possible different such triples is bounded by $\#Q \cdot (k+1) \cdot 2^{\#Q \cdot k}$. This implies that there are two integers i, j, with $0 \leq i < j \leq n$, such that $\mathcal{T}_i = \mathcal{T}_j$. From $(p_i, t_i) = (p_j, t_j)$ it follows that the string $x' = a_1 \cdots a_i a_{j+1} \cdots a_n$ is accepted by a computation with the counter bounded by k. Suppose now that x' is accepted by a computation in which the value of the counter is smaller than k, namely bounded by $k-1$. From $Q_{k-1}(a_1 a_2 \cdots a_i) = Q_{k-1}(a_1 a_2 \cdots a_j)$ and Lemma 4, with $h = k-1$, $u = a_1 a_2 \cdots a_i$, $v = a_1 a_2 \cdots a_j$, and $z = a_{j+1} \cdots a_n$, it follows that x is accepted by a computation in which the value of the counter is at most $k-1$, which is a contradiction. Hence, x' cannot be accepted by any computation in which the value of the counter is smaller than k. This allows to conclude that $x' \in L[k]$. □

We are now able to prove the main result of this section:

Theorem 3. *Let $\mathcal{M}$ be a one-counter automaton using a counter bounded by $h(n)$. Then either $h(n) \in O(1)$, or $h(n) \notin o(\log n)$.*

Proof. Given $k > 0$ such that $L[k] \neq \emptyset$, by Lemma 5 the shortest string $x \in L[k]$ has length $n < \#Q \cdot (k+1) \cdot 2^{\#Q \cdot k}$ that, *ad abundantiam*, gives $n < 2^{\#Q \cdot k} \cdot 2^{\#Q \cdot k} = 2^{2 \cdot \#Q \cdot k}$.

Since x cannot be accepted using a counter lower than k, this implies that $h(n) \geq k > \frac{1}{2 \cdot \#Q} \log_2 n$.

We complete the proof by observing that if $h(n) \notin O(1)$ then there are infinitely many k such that $L[k] \neq \emptyset$. Hence, by applying the previous argument to any such k, we can conclude that there are infinitely many n such that $h(n) \geq \frac{1}{2 \cdot \#Q} \log_2 n$, i.e., $h(n) \notin o(\log n)$. □

We notice that the language L_{bin} presented in Section 3 and studied in Lemma 2 matches the logarithmic lower bound proved in Theorem 3. This language is defined over a 3-letter alphabet that, with a simple encoding, can be reduced to two letters. It is natural to ask whether the logarithmic bound can be reached with an 1DCA accepting a language defined over an alphabet of only one letter. We now give a negative answer to this question, by proving that, in the case of a unary input alphabet, the lower bound in Theorem 3 can be increased to a linear function.

Theorem 4. *Let $\mathcal{M}$ be a one-counter automaton using a counter bounded by $h(n)$. If the input alphabet of $\mathcal{M}$ has cardinality 1, then either $h(n)$ is bounded by a constant or there exists $c > 0$ such that $h(n) \geq c \cdot n$ infinitely often.*

Proof. Let us start by giving an outline of the argument used in [5] to prove that it is decidable whether or not a *unary* PDA $\mathcal{P}$ accepts in constant height. The starting observation is that, when the height used in an accepting computation exceeds the number of possible surface triples, some *vertical loops* occurred. However, it could exist another accepting computation that uses a smaller height (we remind the reader that, for each accepted input, we are considering the minimum height used by an accepting computation). The rough idea is to see whether it is possible to replace the vertical loops occurring in an accepting computation by *horizontal loops*, in order to find another accepting computation for the same input whose height is bounded by a constant. In particular, it has been shown that if an input w has an accepting computation that visits a configuration having a horizontal loop, then w also has an accepting computation accepting w in which the height of the pushdown is bounded by a constant (that depends on the cardinalities of the state set and working alphabet). Hence the PDA $\mathcal{P}$ does not accept in constant height if and only if there are infinitely many strings for which all accepting computations do not visit any configuration having horizontal loops.

So, the problem can be decided as follows. From $\mathcal{P}$ we can (constructively) find two languages L_h and L_v, whose union $L = L_h \cup L_v$ is the language accepted by $\mathcal{P}$. The set L_h contains the strings accepted visiting at least one configuration having a horizontal loop, while L_v contains all the strings that are accepted by at least one computation that does not visit any configuration with a horizontal loop. Thus, $\mathcal{P}$ accepts in constant height if and only if the difference $L_v \setminus L_h$ is finite. Indeed, the height increasing in $\mathcal{P}$ is due to accepting computations on strings in $L_v \setminus L_h$. Notice that the language L_v is accepted by a PDA $\mathcal{P}_v$ that is obtained by removing from $\mathcal{P}$ some transitions (exactly all transitions that exit configurations having horizontal loops). Hence, each accepting computation of $\mathcal{P}_v$ is also an accepting computation of $\mathcal{P}$ and each string in $L_v \setminus L_h$ is accepted using the same height in both $\mathcal{P}$ and $\mathcal{P}_v$. As a consequence, $\mathcal{P}$ accepts in constant height if and only if each string in $L_v \setminus L_h$ is accepted in constant height by $\mathcal{P}_v$.

Obviously, the same analysis applies to unary one-counter automata. Hence, to study the growth of the counter, we can consider the strings in $L_v \setminus L_h$ and the one-counter automaton $\mathcal{M}_v$ accepting L_v obtained according to [5], as outlined

above. We point out that if $\mathcal{M}$ accepts using a counter bounded by $h(n)$, then also $\mathcal{M}_v$ accepts using a counter bounded by $h(n)$.

Let us fix an integer $\overline{n}$ such that $a^{\overline{n}} \in L_v \setminus L_h$. Since the sets of computations of $\mathcal{M}$ and of $\mathcal{M}_v$ on $a^{\overline{n}}$ coincide, $\mathcal{M}_v$ accepts $a^{\overline{n}}$ in the same height $h(\overline{n})$ as $\mathcal{M}$. Let us consider the NFA with ε-transitions $\mathcal{N}_{h(\overline{n})}$, which directly simulates the computations of $\mathcal{M}_v$ with counter bounded by $h(\overline{n})$. This automaton is obtained by encoding the counter in the finite control. More formally, the states of $\mathcal{N}_{h(\overline{n})}$ are pairs (q,t), where q is a state of $\mathcal{M}$, t is an integer with $0 \leq t \leq h(\overline{n})$, the initial and the final state are respectively $(q_I,0)$ and $(q_F,0)$, and the transition function $\delta_{\mathcal{N}_{h(\overline{n})}}$ is defined in such a way that $(q',t') \in \delta_{\mathcal{N}_{h(\overline{n})}}((q,t),\sigma)$ if and only if $(q,\sigma,t) \vdash (q',\varepsilon,t')$ in $\mathcal{M}_v$, for all states q,q', $\sigma \in \{a,\varepsilon\}$, $0 \leq t,t' \leq h(\overline{n})$.

The language accepted by $\mathcal{N}_{h(\overline{n})}$ is a subset of the language accepted by $\mathcal{M}_v$, which includes $a^{\overline{n}}$.

Suppose that $\mathcal{N}_{h(\overline{n})}$ contains a loop from a state (q,t) consuming an input string $x \neq \varepsilon$, i.e., a sequence of states $(q_0,t_0),\ldots,(q_m,t_m) \in Q \times \{0,\ldots h(\overline{n})\}$, $m > 0$, such that $(q_0,t_0) = (q_m,t_m) = (q,t)$, $\delta_{\mathcal{N}_{h(\overline{n})}}((q_{i-1},t_{i-1}),\sigma_i) = (q_i,t_i)$, $\sigma_i \in \{a,\varepsilon\}$, $i = 1,\ldots,m$, with $x = \sigma_1\sigma_2\cdots\sigma_m$.

If $t_j \geq t$ for each $j = 0,\ldots,m-1$, then such a loop simulates an horizontal loop on the surface pair $[q_0A_0]$, in the one-counter automaton $\mathcal{M}_v$. This is not possible because horizontal loops have been removed. Hence, it must exist at least one j, $0 < j < m$, such that $t_j < t$. Among all such j's choose one with minimal t_j, i.e., $t_j \leq t_i$ for $i = 0,\ldots,m$. It is not difficult to observe that the sequence $(q_i,t_i),\ldots,(q_m,t_m),(q_1,t_1),\ldots,(q_i,t_i)$ simulates an horizontal loop on the surface pair $[q_jA_j]$ that consumes the string $\sigma_{i+1}\cdots\sigma_m\sigma_1\cdots\sigma_i$ which, being the alphabet unary, coincides with x. Since $x \neq \varepsilon$, this is a contradiction because horizontal loops have been removed.

Hence, all the loops in the automaton $\mathcal{N}_{h(\overline{n})}$ consist only of ε-moves, so they are useless with respect to acceptance, and each string can be accepted by a path *without any repeated state*. Since $a^{\overline{n}}$ is accepted by $\mathcal{N}_{h(\overline{n})}$, we get that the number of states of the automaton, which is $\#Q \cdot (h(\overline{n})+1)$, must be greater than $\overline{n}$, i.e., $h(\overline{n}) > \frac{\overline{n}}{\#Q} - 1$.

We finally notice that the previous argument can be applied to each string in $L_v \setminus L_h$. This set is infinite, when $h(n)$ is not bounded by a constant. This allows us to conclude that $h(n) > \frac{n}{\#Q} - 1$ infinitely often. By taking $c = \frac{1}{2\#Q}$, we obtain $h(n) \geq c \cdot n$ for infinitely many integers n. □

We finally observe that the lower bound given in Theorem 4 cannot be increased. In fact the unary language $(aa)^*$ consisting of all strings of even length can be accepted by an OCA that in a first phase scans the input while incrementing the counter and at some point, with a nondeterministic choice, enters a second phase in which it continues to scan the input, while decrementing the counter. The machine accepts if, after scanning all the input, the counter contains 0. The value of the counter in the only accepting computation on an input of even length n is $n/2$, hence it is linear in n.

6. The extended language of a pushdown automaton

All the results presented in [5] for PDAs and in the previous sections for one-counter automata refer to the weak measure where, for each accepted word, the cost of a least expensive accepting computation is considered. In the next sections of the paper we will inspect the strong and accept measures, in which the costs of all computations and of all accepting computations, respectively, are taken into account. We will prove that, with respect to these measures, the situation is completely different: first, it becomes decidable whether or not a PDA uses constant height. Furthermore, for PDAs not using constant height, the lower bound for the height increases to a linear function in the input length. Such a bound can be reached even in the case of one-counter automata.

To obtain these results, we will make a reduction to the deterministic case. Roughly, for any given PDA $\mathcal{M}$, we consider a *deterministic* PDA $\mathcal{M}_{\text{ext}}$, with the same structure and the same internal configurations as $\mathcal{M}$, which recognizes a language, called the *extended language* of $\mathcal{M}$, whose strings describe accepting computations of $\mathcal{M}$. Each string in such a language is obtained by padding a string $w \in L(\mathcal{M})$ with some symbols encoding the transitions used in a computation accepting w. In this way, we obtain a bijection between the set of accepting computations of $\mathcal{M}$ and the set of strings in the extended language. Furthermore, as we will show, the extended language is *deterministic* context free and it is regular if and only if $\mathcal{M}$ accepts in constant height. Thus, we can apply to such machines the known results for the deterministic case, recovering from them the properties of the original machine $\mathcal{M}$.

We mention that the idea of padding the input of a PDA with some encoding of the transitions used by the machine during the computations was firstly presented in [7].

Let us start by introducing some notations:

- Given a finite alphabet Γ, let us consider the set $\Gamma^{-1} = \{A^{-1} \mid A \in \Gamma\}$ and define the *Dyck language* over Γ, denoted as D_Γ, as the set of sequences of balanced brackets over $\Gamma \cup \Gamma^{-1}$, where each $A \in \Gamma$ represents an opening bracket and A^{-1} is the corresponding closing bracket.
  If A^{-1} is interpreted as the *right inverse* of A, then in each string from $(\Gamma \cup \Gamma^{-1})^*$ we can replace a factor AA^{-1}, $A \in \Gamma$, with the empty word, obtaining a shorter string. Hence, starting from a string $x \in (\Gamma \cup \Gamma^{-1})^*$, we can iteratively apply the previous process until obtaining a string that cannot be further reduced, namely that does not contain any factor AA^{-1}. It is possible to prove that such a string, denoted as $\mu(x)$, is unique (see [9, Prop. 10.4.1]). In the following $\mu(x)$ will be called the *reduced word* obtained from x.

Observe that, for each $x \in (\Gamma \cup \Gamma^{-1})^*$, $\mu(x) = \varepsilon$ if and only if $x \in D_\Gamma$. Furthermore, for each prefix x' of some word in D_Γ, $\mu(x') \in \Gamma^*$ is the sequence of opening brackets in x' that need to be closed in each x'', with $x'x'' \in D_\Gamma$.

- Given two alphabets Σ, Δ, with $\Sigma \subseteq \Delta$, for each $x \in \Delta^*$ let us denote by $\pi_\Sigma(x)$ the *projection* of x on Σ, i.e., the string obtained by removing from x all the symbols not in Σ. This notion can be extended to languages by defining $\pi_\Sigma(L) = \{\pi_\Sigma(x) \mid x \in L\}$, for $L \subseteq \Delta^*$.

The following lemma states that each subset of a Dyck language containing strings with an arbitrarily large level of nesting cannot be regular.

Lemma 6. *Let Γ be an alphabet and $D \subseteq D_\Gamma$ be a subset of the Dyck language over Γ. If for each integer $h > 0$ there exists $z \in D$ such that z has a prefix z' with $\mu(z') \in \Gamma^h$, then D is not regular.*

Proof. By contradiction, suppose that D is accepted by a DFA A with N states. Let $z \in D$ be a string having a prefix z' with $\mu(z') \in \Gamma^N$. Suppose that z' is the shortest prefix of z with such a property and let z'' be the corresponding suffix, i.e., $z = z'z''$.

We decompose z' as $z' = z_1 z_2 \cdots z_N$ where, for $i = 1, \ldots, N-1$, $z_1 z_2 \cdots z_i$ is the longest prefix of z' such that $\mu(z_1 z_2 \cdots z_i) \in \Gamma^i$. Intuitively, $z_1 \cdots z_i$ contains exactly i opening brackets whose matching closing brackets are located in the suffix z''. Observe that $\mu(z_i) \in \Gamma$, for $i = 1, \ldots, N$.

Let p_i be the state reached by A after reading $z_1 \cdots z_i$. Then $p_i = p_j$ for some $0 \le i < j \le N$. So the string $\tilde{z} = z_1 \cdots z_i (z_{i+1} \cdots z_j)^2 z_{i+1} \cdots z_N z''$ obtained by pumping z with the factor $z_{i+1} \cdots z_j$ is also accepted by A. However, since the factor inserted in z to obtain $\tilde{z}$ verifies $\mu(z_{i+1} \cdots z_j) \in \Gamma^{j-i}$, the string $\tilde{z}$ contains $j - i > 0$ more opening than closing brackets. Hence, $\tilde{z} \notin D_\Gamma$. This is a contradiction. □

From now on let us fix a PDA $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_I, Z_0, q_F \rangle$.

Let us consider the *global alphabet* $\Delta = \Sigma \cup \{\mathcal{E}\} \cup \Gamma \cup \Gamma^{-1}$, where $\mathcal{E} \notin \Sigma \cup \Gamma$ is a new symbol that will be used to encode transitions on the empty word that do not change the pushdown contents.

We are going to consider strings in $(\Delta \cdot Q)^+$. The rough idea is to use a string $x = a_0 p_0 a_1 p_1 \cdots a_m p_m$, $m > 0$, $a_i \in \Delta$, $p_i \in Q$, $i = 0, \ldots, m$, to represent a computation path which starts with $a_0 \in \Gamma$ on the top of the pushdown in the state p_0 and makes m moves, where the ith move is encoded by the factor $p_{i-1} a_i p_i$, $i = 1, \ldots, m$. For $a_i \in \Sigma \cup \{\mathcal{E}\}$, the move does not change the pushdown store and reads $a_i \in \Sigma$ from the tape, or does not read any input symbol, if $a_i = \mathcal{E}$; for $a_i \in \Gamma$, the move is a push of a_i on the pushdown; for $a_i \in \Gamma^{-1}$, the move pops off the pushdown A, where $a_i = A^{-1}$. In all the cases, the move changes the state from p_{i-1} to p_i.

The moves also depend on the symbol on the top of the pushdown. Such a symbol and the entire pushdown content can be recovered from the string x, provided that it effectively represents a computation path. To this aim we associate with x a string $\mathtt{store}(x) \in (\Gamma \cup \Gamma^{-1})^*$, a symbol $\mathtt{top}(x) \in \Gamma \cup \Gamma^{-1}$, and a string $\mathtt{input}(x) \in \Sigma^*$, defined as follows:

- $\mathtt{store}(x) = \mu(\pi_{\Gamma \cup \Gamma^{-1}}(x))$, intuitively it represents the pushdown content after the moves specified in x (with the topmost symbol to the right),
- $\mathtt{top}(x)$ is the rightmost symbol of $\mathtt{store}(x)$, if $\mathtt{store}(x)$ is not empty, otherwise it is undefined; it represents the top of the pushdown after the moves specified by x,
- $\mathtt{input}(x) = \pi_\Sigma(x)$.

Let $x = a_0 p_0 a_1 p_1 \cdots a_m p_m \in (\Delta \cdot Q)^+$, $m > 0$. For $i = 0, \ldots, m$, let us denote by x_i the prefix of length $2i + 2$ of x, namely $x_i = a_0 p_0 a_1 p_1 \cdots a_i p_i$.

We say that x is *well formed* when:

- For $i = 0, \ldots, m$, $\mathtt{store}(x_i) \in \Gamma^*$ and, if $i < m$, $\mathtt{store}(x_i) \neq \varepsilon$;
- For $i = 1, \ldots, m$, let $A = \mathtt{top}(x_{i-1})$, then:
  - if $a_i \in \Sigma$ then $(p_i, -) \in \delta(p_{i-1}, a_i, A)$;
  - if $a_i = \mathcal{E}$ then $(p_i, -) \in \delta(p_{i-1}, \varepsilon, A)$;
  - if $a_i \in \Gamma$ then $(p_i, \mathtt{push}(a_i)) \in \delta(p_{i-1}, \varepsilon, A)$;
  - if $a_i \in \Gamma^{-1}$ then $(p_i, \mathtt{pop}) \in \delta(p_{i-1}, \varepsilon, A)$ and $a_i = A^{-1}$.

Notice that the first condition implies $a_0 \in \Gamma$. Intuitively, if x is well formed then it represents a computation path that starts with the pushdown store containing a string γ plus the symbol a_0 on the top, which is never removed except possibly in the last step, consumes the string $\mathtt{input}(x)$ from the input tape, and ends with the pushdown store containing the original string γ in the bottom part plus the string $\mathtt{store}(x)$. Such a computation path does not depend on γ.

More formally, we prove the following:

Lemma 7. *A string $x = a_0p_0a_1p_1\cdots a_mp_m \in (\Delta\cdot Q)^+$, $m \geq 0$, is well formed if and only if there is a computation path*

$$(p_0, w_0, \gamma_0)\vdash(p_1, w_1, \gamma_1)\vdash\cdots\vdash(p_m, w_m, \gamma_m)$$

such that $\gamma_i = \mathtt{store}(a_0p_0\cdots a_ip_i)$ and $w_i = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_mp_m)$, for $i = 0, \ldots, m$.

Proof. First of all, we observe that, by definition, if x is well formed then $a_0 \in \Gamma$. Furthermore, if γ_0 and w_m satisfy the statement of the lemma, then $\gamma_0 = a_0$ and $w_m = \varepsilon$. We proceed by induction on m.

Basis ($m = 0$).

In this case $x = a_0p_0$ and the computation path consists only of the configuration $(p_0, \varepsilon, \gamma_0)$, where $\gamma_0 = a_0 = \mathtt{store}(a_0p_0)$, $\mathtt{input}(a_0p_0) = \varepsilon$.

Induction (from $m-1$ to m, for $m > 0$).

Given $x = a_0p_0\cdots a_mp_m$, consider $x' = a_0p_0\cdots a_{m-1}p_{m-1}$. Applying the induction hypothesis to $x' = a_0p_0\cdots a_{m-1}p_{m-1}$, we get a computation path

$$(p_0, w_0, \gamma_0)\vdash(p_1, w_1, \gamma_1)\vdash\cdots\vdash(p_{m-1}, w_{m-1}, \gamma_{m-1}) \tag{1}$$

such that $\gamma_i = \mathtt{store}(a_0p_0\cdots a_ip_i)$ and $w_i = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_{m-1}p_{m-1})$, for $i = 0, \ldots, m-1$. Let $A = \mathtt{top}(x')$. The following cases are possible:

- $a_m \in \Sigma$ *and* $(p_m, -) \in \delta(p_{m-1}, a_m, A)$.
  Appending a_m to the second components of configurations in the path (1), we obtain the path
  $$(p_0, w_0a_m, \gamma_0)\vdash(p_1, w_1a_m, \gamma_1)\vdash\cdots\vdash(p_{m-1}, w_{m-1}a_m, \gamma_{m-1})\,.$$
  By applying the transition $(p_m, -) \in \delta(p_{m-1}, a_m, A)$ to the last configuration, the machine reaches the configuration (p_m, w_m, γ_m), with $\gamma_m = \gamma_{m-1} = \mathtt{store}(a_0p_0\cdots a_{m-1}p_{m-1}) = \mathtt{store}(a_0p_0\cdots a_{m-1}p_{m-1}a_mp_m)$.
  Furthermore, $w_ia_m = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_{m-1}p_{m-1})a_m = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_{m-1}p_{m-1}a_mp_m)$, for $i = 0, \ldots, m$.
- $a_m = \mathcal{E}$ *and* $(p_m, -) \in \delta(p_{m-1}, \varepsilon, A)$.
  Similar to the previous case, using ε instead of a_m.
- $a_m \in \Gamma$ *and* $(p_m, \mathtt{push}(a_m)) \in \delta(p_{m-1}, \varepsilon, A)$.
  It is enough to extend the path (1) with the move $(p_{m-1}, w_{m-1}, \gamma_{m-1})\vdash(p_m, w_{m-1}, \gamma_{m-1}a_m)$ and to take $\gamma_m = \gamma_{m-1}a_m$, $w_m = w_{m-1} = \varepsilon$.
- $a_m \in \Gamma^{-1}$ *and* $(p_m, \mathtt{pop}) \in \delta(p_{m-1}, \varepsilon, A)$ *with* $a_m = A^{-1}$.
  Since $\gamma_{m-1} = \mathtt{store}(x')$, then $\gamma_{m-1} = \gamma' a_m$, for some γ'. It is enough to extend the path (1) with the move $(p_{m-1}, w_{m-1}, \gamma' a_m)\vdash(p_m, w_{m-1}, \gamma')$ and to take $\gamma_m = \gamma'$, $w_m = w_{m-1} = \varepsilon$.

Conversely, let us consider a computation

$$(p_0, w_0, \gamma_0)\vdash(p_1, w_1, \gamma_1)\vdash\cdots\vdash(p_{m-1}, w_{m-1}, \gamma_{m-1})\vdash(p_m, w_m, \gamma_m) \tag{2}$$

such that $\gamma_i = \mathtt{store}(a_0p_0\cdots a_ip_i)$ and $w_i = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_mp_m)$, for $i = 0, \ldots, m$. Let $A \in \Gamma$ be the rightmost symbol of γ_{m-1}, namely the pushdown top in the configuration $(p_{m-1}, w_{m-1}, \gamma_{m-1})$. Again we consider four cases, depending on the last move used in the computation:

- *The last move corresponds to the transition* $(p_m, -) \in \delta(p_{m-1}, a, A)$, *with* $a \in \Sigma$.
  In this case $w_i = w_i'a$, for some $w_i' \in \Sigma^*$, $i = 1, \ldots, m-1$. Therefore, by considering the computation path $(p_0, w_0', \gamma_0)\vdash\cdots\vdash(p_{m-1}, w_{m-1}', \gamma_{m-1})$, using the induction hypothesis we get that there exists a well-formed string $x' = a_0p_0\cdots a_{m-1}p_{m-1}$ such that $\gamma_i = \mathtt{store}(a_0p_0\cdots a_ip_i)$ and $w_i' = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_{m-1}p_{m-1})$, from which we can obtain $w_i = w_i'a = \mathtt{input}(a_{i+1}p_{i+1}\cdots a_{m-1}p_{m-1}ap_m)$, for $i = 0, \ldots, m-1$. Hence, $(p_0, w_0, \gamma_0)\vdash\cdots\vdash(p_{m-1}, w_{m-1}, \gamma_{m-1})$, with $w_{m-1} = a$. Furthermore, by taking $w_m = \varepsilon$ and $\gamma_m = \gamma_{m-1}$, we get that $(p_{m-1}, w_{m-1}, \gamma_{m-1})\vdash(p_m, w_m, \gamma_m)$. So, the well-formed string we are looking for is $x'ap_m$.
- *The last move corresponds to the transition* $(p_m, -) \in \delta(p_{m-1}, \varepsilon, A)$.
  Similar to the previous case, by taking $a = \mathcal{E}$.
- *The last move corresponds to the transition* $(p_m, \mathtt{push}(B)) \in \delta(p_{m-1}, \varepsilon, A)$.
  By induction hypothesis, from $(p_0, w_0, \gamma_0)\vdash\cdots\vdash(p_{m-1}, w_{m-1}, \gamma_{m-1})$ we obtain $x' = a_0p_0\cdots a_{m-1}p_{m-1}$. Furthermore $\gamma_m = \gamma_{m-1}B$. It is not difficult to verify that the well-formed string corresponding to the computation path (2) is $x'Bp_m$.
- *The last move corresponds to the transition* $(p_m, \mathtt{pop}) \in \delta(p_{m-1}, \varepsilon, A)$.
  Again, using the induction hypothesis we obtain $x' = a_0p_0\cdots a_{m-1}p_{m-1}$. The well-formed string corresponding to the computation path (2) is $x'A^{-1}p_m$. □

At this point we are able to define the *extended language* of $\mathcal{M}$ as the set of well-formed strings encoding accepting computations of $\mathcal{M}$, namely:

$$EL(\mathcal{M}) = \{x = a_0 p_0 a_1 p_1 \cdots a_m p_m \mid x \text{ is well formed, } a_0 = Z_0,\ p_0 = q_I,\ p_m = q_F, \text{ and } \mu(x) = Z_0\}$$

Lemma 8. *For each $w \in \Sigma^*$, there is a bijection between the set of accepting computations of $\mathcal{M}$ on w and the set of strings $x \in EL(\mathcal{M})$ such that $\mathtt{input}(x) = w$. Furthermore, $\mu(x) = Z_0$ for each $x \in EL(\mathcal{M})$.*

Proof. Consequence of Lemma 7 and of the fact that in the accepting configuration the pushdown store contains only the bottom symbol Z_0 which, as we observed, is never removed during accepting computations. □

Lemma 9. *For every* PDA *$\mathcal{M}$ there exists a* DPDA *$\mathcal{M}_{\text{ext}}$ accepting the language $EL(\mathcal{M})$ such that for any integer $h \geq 0$, $\mathcal{M}_{\text{ext}}$ accepts a string $x \in \Delta^*$ using height h if and only if $\mathcal{M}$ has an accepting computation on $\mathtt{input}(x)$ using height h.*

Proof. The rough idea is to define $\mathcal{M}_{\text{ext}}$ in such a way that each accepting computation of $\mathcal{M}_{\text{ext}}$ on input $x \in (\Delta \cup Q)^*$ visits exactly the same internal configurations as an accepting computation of $\mathcal{M}$ on $\mathtt{input}(x)$. To do that, in each internal configuration with state q and pushdown store containing γ, $\mathcal{M}_{\text{ext}}$ reads the next two input symbols a, p. If $a \in \Delta$, $p \in Q$, and the move of $\mathcal{M}$ which corresponds to a can be applied in the current configuration leading to the internal configuration (p, γ') in $\mathcal{M}$, for some $\gamma' \in \Gamma^*$, then even in $\mathcal{M}_{\text{ext}}$ the same configuration is reached, otherwise the computation stops.

Formally $\mathcal{M}_{\text{ext}} = \langle Q', \Delta \cup Q, \Gamma, \delta', q_I, Z_0, q_F \rangle$, where $Q' = Q \cup (Q \times \Delta) \cup (Q \times \Delta \times Q)$ and δ' is defined as follows, for $q, p \in Q$, $a \in \Delta$, $A, B \in \Gamma$:

1. $\delta'(q, a, A) = \{([q, a], -)\}$
2. $\delta'([q, a], p, A) = \{([q, a, p], -)\}$
3. $\delta'([q, a, p], \varepsilon, A) = \begin{cases} \{(p, -)\}, & \text{if } a \in \Sigma \text{ and } (p, -) \in \delta(q, a, A); \\ \{(p, -)\}, & \text{if } a = \mathcal{E} \text{ and } (p, -) \in \delta(q, \varepsilon, A); \\ \{(p, \mathtt{push}(B))\}, & \text{if } a = B \text{ and } (p, \mathtt{push}(B)) \in \delta(q, \varepsilon, A); \\ \{(p, \mathtt{pop})\}, & \text{if } a = A^{-1} \text{ and } (p, \mathtt{pop}) \in \delta(q, \varepsilon, A). \end{cases}$
4. In all the remaining cases no transition is defined, i.e., the value of δ' is the empty set.

Transitions 1 and 2, starting from a state q, allow to read a factor $ap \in \Delta \cdot Q$ from the tape and to save the triple $[q, a, p]$ in the control. After that, Transition 3 can be used, which allows to simulate moves "compatible" with the saved triple, finally keeping in the control the state p that will be used, if the computation is not finished, as the starting point of a new triple, obtained starting with a transition of the form 1 in which p replaces q.

The reader can easily observe that $\mathcal{M}_{\text{ext}}$ is deterministic. Furthermore, if $x = a_0 p_0 \cdots a_m p_m$ is a well-formed string with $a_0 = Z_0$ and $p_0 = q_I$ then, according to the above-given construction and to Lemma 7, $\mathcal{M}_{\text{ext}}$ from the initial configuration, after reading x (and performing the transition 3 of the triple $[p_{m-1}, a_m, p_m]$) reaches the internal configuration (q, γ), with $q = p_m$, if and only if there exists a computation of $\mathcal{M}$ that, from the initial configuration, after reading $\mathtt{input}(x)$, reaches the configuration (q, γ). Hence, the two computations use the same height. □

From Lemma 9 we obtain:

Theorem 5. *The language $EL(\mathcal{M})$ is a deterministic context-free language. Furthermore, it is regular if and only if there exists a constant h such that each accepting computation of $\mathcal{M}$ uses at most height h.*

Proof. Since $EL(\mathcal{M})$ is accepted by the deterministic PDA $\mathcal{M}_{\text{ext}}$ described in the proof of Lemma 9, it is a deterministic context-free language. We now study its regularity.

Suppose that each accepting computation of $\mathcal{M}$ uses at most height h, for some constant h. Then also each accepting computation of the DPDA $\mathcal{M}_{\text{ext}}$ obtained in Lemma 9 uses height at most h. Hence, $EL(\mathcal{M})$ is accepted in constant height and, so, it is regular.

Conversely, suppose that $EL(\mathcal{M})$ is regular. Then, from the closure properties of regular languages, it follows that also the language $Y = \pi_{\Gamma \cup \Gamma^{-1}}(EL(\mathcal{M}))$ and the language D obtained by removing from each string in Y the first symbol Z_0 are regular. Furthermore $D \subseteq D_\Gamma$. By Lemma 6 this implies that there is a constant h such that for each string $z \in D$ and for each prefix z' of z, $\mu(z') \in \Gamma^i$ for some $i \leq h$. Let $x \in EL(\mathcal{M})$ be a string verifying $Z_0 z = \mathtt{store}(x) = \pi_{\Gamma \cup \Gamma^{-1}}(x)$. Then for each prefix x' of x we have $Z_0 z' = \mathtt{store}(x')$ for a prefix z' of z. This allows to conclude that the height of the pushdown store in all accepting computations is bounded by a constant. □

The tools we have developed in this section, mainly Lemma 9 and Theorem 5, will be extensively used to obtain the results presented in Section 7.

7. Decidability and lower bounds

In [5] it has been proved that it is not decidable whether a PDA accepts in constant height. That result is proved for the weak measure, namely by considering the cost, on each accepted input, of the least expensive computation. As a consequence of Lemma 9, here we prove that the question becomes decidable if we take into account the amount of pushdown store used in *all* accepting computations.

Theorem 6. *It is decidable whether a* PDA *works in constant height under the accept measure.*

Proof. With each PDA $\mathcal{M}$, we associate the *deterministic* PDA $\mathcal{M}_{\text{ext}}$ described in Lemma 9. Hence, for any fixed constant h, *each* accepting computation of $\mathcal{M}$ uses at most height h if and only if, on each accepted input, $\mathcal{M}_{\text{ext}}$ uses height at most h. The result follows from the decidability of acceptance in constant height for deterministic PDAs [3]. □

We can also prove the following:

Theorem 7. *Let $\mathcal{M}$ be a* PDA*, with set of states Q and pushdown alphabet Γ, such that each accepting computation of $\mathcal{M}$ uses at most height h, for some constant h. Then h is less than $(\#Q)^2 \cdot \#\Gamma$.*

Proof. Roughly, if the height in an accepting computation exceeds the number of *surface triples*, then a *vertical loop* occurs in it. So the computation can be pumped in order to obtain other accepting computations using an arbitrarily large amount of pushdown store. (Similar ideas have been used in [3].)

More formally, given an accepting computation $\mathcal{C}$ using height $h \geq (\#Q)^2 \cdot \#\Gamma$, let us divide it in $\mathcal{C}'$ and $\mathcal{C}''$, where $\mathcal{C}'$ is the part of $\mathcal{C}$ from the initial configuration to a chosen configuration C having height h, and $\mathcal{C}''$ is the remaining part of $\mathcal{C}$, namely from C to the last configuration of $\mathcal{C}$. For $i = 0, \ldots, h$, let us consider the last configuration C'_i of $\mathcal{C}'$ having height i and the first configuration C''_i of $\mathcal{C}''$ having height i. Since there are no configurations between C'_i and $\mathcal{C}''$ with height lower than i, we obtain a surface triple $[q_i A_i p_i]$ where q_i and p_i are the states in $\mathcal{C}'$ and $\mathcal{C}''$, respectively, and A_i is the symbol on the top of the pushdown in such configurations.[3]

Since $h \geq (\#Q)^2 \cdot \#\Gamma$, there are i, j, $0 \leq i < j \leq h$, such that the triples $[q_i A_i p_i]$ and $[q_j A_j p_j]$ are equal. For each $k > 0$, if we pump $\mathcal{C}'$ with k occurrences of the path from C'_i to C'_j and $\mathcal{C}''$ with k occurrences of the path from C''_j to C''_i, the resulting computation is still accepting and uses height $h + (j - i) \cdot k > h$. This allows us to conclude that if an accepting computation uses height at least $(\#Q)^2 \cdot \#\Gamma$, then the PDA $\mathcal{M}$ has accepting computations using an arbitrarily large amount of pushdown store. □

We point out that, in contrast to Theorem 7, in the case of the weak measure there is no recursive function which gives an upper limit to the height of a PDA accepting in constant height with respect to the size of the PDA [5].

We continue the analysis of the accept case, by proving that if under this measure the height is not constant with respect to the length of the input, then it must grow at least linearly. As already mentioned, the optimal lower bound in the weak case is double logarithmic if the input alphabet contains at least two symbols, while it is logarithmic for unary input alphabets.

Theorem 8. *Let $\mathcal{M}$ be a* PDA*. Let us denote by $h_{\mathcal{M},acc}(n)$ the maximum height used in accepting computations on inputs of length n. Then either $h_{\mathcal{M},acc}(n) \in O(1)$, or $h_{\mathcal{M},acc}(n) \notin o(n)$.*

Proof. If $h_{\mathcal{M},acc}(n) \notin O(1)$, then by Theorem 5 the language $EL(\mathcal{M})$ is deterministic context free but it is not regular. By Theorem 1 in [8], to accept it a linear height is necessary. Hence the PDA $\mathcal{M}_{\text{ext}}$ accepting $EL(\mathcal{M})$ uses height $h'(n') \notin o(n')$, where h' is the height used by the DPDA $\mathcal{M}_{\text{ext}}$, and n' is the length of the input of $\mathcal{M}_{\text{ext}}$; i.e., there exists $c > 0$ such that $h'(x) > c \cdot |x|$ for infinitely many x. Considering the costs of *all* accepting computations of $\mathcal{M}$ on $\texttt{input}(x)$, we obtain $h'(x) \leq h_{\mathcal{M},acc}(\texttt{input}(x))$. This implies $h_{\mathcal{M},acc}(\texttt{input}(x)) > c \cdot |x| \geq c \cdot |\texttt{input}(x)|$. Hence $h_{\mathcal{M},acc}(n) > c \cdot n$ for infinitely many n, thus implying $h_{\mathcal{M},acc}(n) \notin o(n)$. □

We now consider the strong measure, which is defined by taking into account the cost of *each* computation. To extend the results in Theorems 6, 7, and 8 to this case, we could try to make each computation accepting. However, a PDA can have computations that stop before reading all the input and computations that enter infinite loops, which will not be considered using this approach.

To overcome this problem, from any given PDA $\mathcal{M}$ we build another PDA $\widetilde{\mathcal{M}}$ that, in each configuration, can nondeterministically enter a special state q_e, where it consumes input and pushdown symbols, finally reaching the accepting

[3] We remind the reader that we do not count the bottom symbol Z_0 in the height, i.e., height 0 means that the pushdown store contains only the bottom symbol.

Table 1
Bounds for the height of the pushdown store and for the value of the counter, if non constant.

	strong $\#\Sigma \geq 1$	accept $\#\Sigma \geq 1$	weak $\#\Sigma > 1$	weak $\#\Sigma = 1$
Pushdown Automata	n (Th. 9)	n (Th. 8)	$\log\log n$ ([15])	$\log n$ ([5])
One-Counter Automata	n (Th. 9)	n (Th. 8)	$\log n$ (Th. 3)	n (Th. 4)

configuration. In this way, each configuration reachable in $\mathcal{M}$ on input w belongs to some accepting computation of $\widetilde{\mathcal{M}}$ on w that does not use any extra amount of pushdown store.

Formally, the set of states of $\widetilde{\mathcal{M}}$ is obtained by adding to the set of states of $\mathcal{M}$ the new state q_e. The PDA $\widetilde{\mathcal{M}}$ has the same transitions as $\widetilde{\mathcal{M}}$, plus one transition from each state q that, without reading any input symbol and without changing the pushdown store, enters the state q_e. In the state q_e, $\widetilde{\mathcal{M}}$ can always remove the symbol on the top of the pushdown, remaining in the same state and it can also read a symbol from the tape, without changing the pushdown store and the state. Furthermore, q_e is the final state of $\widetilde{\mathcal{M}}$. In this way, for each prefix $\mathcal{C}_p$ of each computation $\mathcal{C}$ of $\mathcal{M}$, $\widetilde{\mathcal{M}}$ has an accepting computation $\widetilde{\mathcal{C}}$ consisting of the same moves as $\mathcal{C}_p$ followed by a transitions entering the state q_e and by a sequence of configurations in which the state is never changed. Furthermore, in these configurations the height of the pushdown store never increases. Hence, the height used in $\widetilde{\mathcal{C}}$ is the same as the one used in $\mathcal{C}_p$.

This argument allows us to conclude that $\mathcal{M}$ on input w can reach a configuration using height h if and only if $\widetilde{\mathcal{M}}$ in some accepting computation on input w visits a configuration using height h. So $h_{\mathcal{M},strong}(n) = h_{\widetilde{\mathcal{M}},acc}(n)$, where $h_{\mathcal{M},strong}(n)$ denotes the maximal height used by $\mathcal{M}$ on inputs of length n. As a consequence, using Theorems 6 to 8, we obtain the following results:

Theorem 9. *Let $\mathcal{M}$ be a* PDA.

- *It is decidable whether $\mathcal{M}$ works in constant height under the strong measure.*
- *If every accepting computation of $\mathcal{M}$ uses at most height h, for some constant h, then h is upper bounded by n^2m, where n and m are the cardinalities of state set and pushdown alphabet of $\mathcal{M}$, respectively.*
- *Let $h_{\mathcal{M},strong}(n)$ be the maximum height of the pushdown store used in the computations on inputs of length n. Then either $h_{\mathcal{M},strong}(n) \in O(1)$, or $h_{\mathcal{M},strong}(n) \notin o(n)$.*

8. Conclusion

Table 1 summarizes the lower bounds for the height of the pushdown store and for the value of the counter, with respect to the length of the input, when they are non constant. For each of these bounds it is possible to show a language witnessing that it can be reached.

We could ask if it is also possible to give *upper bounds* for the height of the pushdown store with respect to the length of the input. It is known that each context-free language can be accepted by a pushdown automaton using height that is at most linear in the length of the input (consider, e.g., the standard conversion from context-free grammars in Greibach normal form into equivalent pushdown automata). However, we could ask, for each *fixed* PDA, if such an upper bound there exists. In general, the answer is negative, because due to ε-transitions some PDAs can have even accepting computations using an arbitrarily large height. However, we can prove that for each halting computation $\mathcal{C}$ there always exists a *reduced computation* $\mathcal{C}_r$ using at most linear height, where in this context "reduced" means that $\mathcal{C}_r$ can be obtained from $\mathcal{C}$ by removing some redundant sequences of ε-transitions, while all the moves in $\mathcal{C}$ on "real" input symbols are unchanged.

Theorem 10. *Given a* PDA *$\mathcal{M}$, for each halting computation $\mathcal{C}$ there exists a computation $\mathcal{C}_r$ that uses at most linear height with respect to the length of the input and it is accepting if and only if $\mathcal{C}$ is accepting. In particular, $\mathcal{C}_r$ can be obtained by removing from $\mathcal{C}$ some sequences of ε-transitions, while performing exactly the same transitions as $\mathcal{C}$ on symbols from the input alphabet.*

Proof. The idea is to show that if a halting computation uses more than linear height, then it can be "unpumped" by removing some sequences of ε-transitions, in order to reduce the height and to obtain a shorter computation, on the *same input*, using exactly the same transitions on "real" input symbols.

First, let us prove the result in the case of accepting computations.

The technical argument is similar to the one used to prove Theorem 7, with the main difference that here we have to locate vertical loops consisting of ε-transitions only.

Consider an accepting computation $\mathcal{C}$ on an input w of length n in which the maximal height reached by the pushdown store is h. We are going to show how to get a shorter accepting computation of the same input, by removing some ε-transitions, when $h > (n+1) \cdot (\#Q)^2 \cdot \#\Gamma$.

As in the proof of Theorem 7, let us divide $\mathcal{C}$ in $\mathcal{C}'$ and $\mathcal{C}''$, where $\mathcal{C}'$ is the part of $\mathcal{C}$ from the initial configuration to a chosen configuration C having height h, and $\mathcal{C}''$ is the remaining part of $\mathcal{C}$, from the configuration C to the last configuration

of $\mathcal{C}$. For $i = 0, \ldots, h$, let us consider the last configuration C'_i of $\mathcal{C}'$ having height i, the first configuration C''_i of $\mathcal{C}''$ having height i, and the surface triple $[q_i A_i p_i]$ corresponding to them.

We can factorize the sequence α of triples $[q_0 A_0 p_0], [q_1 A_1 p_1], \ldots, [q_h A_h p_h]$ as $\alpha_0 \alpha_1 \cdots \alpha_k$ in such a way that two consecutive triples $[q_{i-1} A_{i-1} p_{i-1}]$ and $[q_i A_i p_i]$ are in the same factor if and only if C'_i is reached from C'_{i-1} *using only* ε-transitions and C''_{i-1} is reached from C''_i *using only* ε-transitions, i.e., without reading any input symbol. Since the input length is n and assuming, without loss of generality, that each α_j is not empty, we have $k \leq n$. Furthermore, if each factor α_j consisted of at most $(\#Q)^2 \cdot \#\Gamma$ triples, we would obtain $h \leq (n+1) \cdot (\#Q)^2 \cdot \#\Gamma$, which is a contradiction. Hence, there is a factor consisting of a sequence of triples $[q_s A_s p_s], [q_{s+1} A_{s+1} p_{s+1}], \ldots, [q_{s+t} A_{s+t} p_{s+t}]$ such that $t \geq (\#Q)^2 \cdot \#\Gamma$ and only ε-transitions are used in the parts of $\mathcal{C}$ from C'_s to C'_{s+t} and from C''_{s+t} to C''_s. This implies the existence of two indices i, j, $s \leq i < j \leq s+t$, such that the triples $[q_i A_i p_i]$ and $[q_j A_j p_j]$ are equal.

By removing from $\mathcal{C}$ the parts from C'_i to C'_j and from C''_j to C''_i, a shorter computation on the same input, using exactly the same transitions as $\mathcal{C}$ on "real" input symbols, is obtained. Furthermore, such a computation starts and ends exactly in the same configurations as the original computation $\mathcal{C}$. Hence, since we supposed $\mathcal{C}$ accepting, also the computation so obtained is accepting, but it is shorter than $\mathcal{C}$. In the case it still uses height greater than $(n+1) \cdot (\#Q)^2 \cdot \#\Gamma$, we can iterate the same argument until finally obtaining an accepting computation, for the same input, which uses the same transitions on "real" input symbols as the original one and has height at most $(n+1) \cdot (\#Q)^2 \cdot \#\Gamma$.

Now, suppose that $\mathcal{C}$ is halting and not accepting. First of all, we observe that, if in the last configuration of $\mathcal{C}$ the pushdown store is empty, then the previous argument also works. More in general, the argument also works if in the above-considered part $\mathcal{C}''$ there is a configuration in which the height of the pushdown is 0, i.e., the store contains only the bottom symbol Z_0. Indeed this allows to define all the triples $[q_i A_i p_i]$, $i = 0, \ldots, h$. If this is not the case, then let us consider the minimal height $j > 0$ reached by the pushdown store in the part $\mathcal{C}''$. For $i = j, \ldots, h$, we still consider the above-defined triples $[q_i A_i p_i]$, while for $i = 0, \ldots, j-1$ we consider triples $[q_i A_i -]$. We point out that, in the last case, since height i is not reached in $\mathcal{C}''$, we do not have a state defining the third component. The symbol '$-$' replaces it and can be interpreted as a dummy state that could be introduced if we want to extend the computation $\mathcal{C}$ with additional moves to make the pushdown store empty. We can easily adapt the previous combinatorial argument, used in the case of accepting computations, to the sequence of triples so obtained, in order to obtain the reduced computation. □

We finally mention that all the results we presented for one-counter automata also hold for one-counter automata *without zero test*, namely those that, when the counter is 0, can perform exactly the same transitions, as when the counter is greater than zero, except those decreasing the counter.

CRediT authorship contribution statement

Giovanni Pighizzini: Writing – review & editing, Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization. **Luca Prigioniero:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] V. Geffert, C. Mereghetti, B. Palano, More concise representation of regular languages by automata and regular expressions, Inf. Comput. 208 (4) (2010) 385–394.
[2] Z. Bednárová, V. Geffert, C. Mereghetti, B. Palano, Removing nondeterminism in constant height pushdown automata, Inf. Comput. 237 (2014) 257–267.
[3] A. Malcher, K. Meckel, C. Mereghetti, B. Palano, Descriptional complexity of pushdown store languages, J. Autom. Lang. Comb. 17 (2–4) (2012) 225–244.
[4] B. Guillon, G. Pighizzini, L. Prigioniero, Non-self-embedding grammars, constant-height pushdown automata, and limited automata, Int. J. Found. Comput. Sci. 31 (8) (2020) 1133–1157.
[5] G. Pighizzini, L. Prigioniero, Pushdown automata and constant height: decidability and bounds, Acta Inform. 60 (2) (2023) 123–144.
[6] A. Bertoni, C. Mereghetti, G. Pighizzini, Strong optimal lower bounds for Turing machines that accept nonregular languages, in: J. Wiedermann, P. Hájek (Eds.), Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Prague, Czech Republic, August 28 - September 1, 1995, Proceedings, in: Lecture Notes in Computer Science, vol. 969, Springer, 1995, pp. 309–318.
[7] K. Salomaa, D. Wood, S. Yu, Pumping and pushdown machines, RAIRO Theor. Inform. Appl. 28 (3–4) (1994) 221–232.
[8] J. Gabarró, Pushdown space complexity and related full-AFLs, in: M. Fontet, K. Mehlhorn (Eds.), STACS 84, Symposium of Theoretical Aspects of Computer Science, Paris, France, 11-13 April, 1984, Proceedings, in: Lecture Notes in Computer Science, vol. 166, Springer, 1984, pp. 250–259.
[9] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.
[10] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.
[11] P.M. Lewis II, R.E. Stearns, J. Hartmanis, Memory bounds for recognition of context-free and context-sensitive languages, in: 6th Annual Symposium on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, USA, October 6-8, 1965, IEEE Computer Society, 1965, pp. 191–202.
[12] O.H. Ibarra, Restricted one-counter machines with undecidable universe problems, Math. Syst. Theory 13 (1979) 181–186.
[13] J. Hartmanis, Context-free languages and Turing machine computations, in: Mathematical Aspects of Computer Science, in: Proceedings of Symposia in Applied Mathematics, vol. 19, American Mathematical Society, 1967, pp. 42–51.

[14] A.R. Meyer, M.J. Fischer, 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, IEEE Computer Society, 1971, pp. 188–191.
[15] M. Alberts, Space complexity of alternating Turing machines, in: L. Budach (Ed.), Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9–13, 1985, in: Lecture Notes in Computer Science, vol. 199, Springer, 1985, pp. 1–7.
[16] Z. Bednárová, V. Geffert, K. Reinhardt, A. Yakaryilmaz, New results on the minimum amount of useful space, Int. J. Found. Comput. Sci. 27 (2) (2016) 259–282.