Minimal Useful Size of Counters for (Real-Time) Multicounter Automata

Viliam Geffert and Zuzana Bednárová^(✉)

Department of Computer Science, P. J. Šafárik University,
Jesenná 5, 04154 Košice, Slovakia
{viliam.geffert,zuzana.bednarova}@upjs.sk

Abstract. We show that, for nondeterministic and alternating machines with weak space bounds, the minimal space that is required for accepting a nonregular language by real-time or one-way multicounter automata is $(\log n)^{\varepsilon}$. The same space is required for two-way multicounter automata, independent of whether they are deterministic, nondeterministic, or alternating, and of whether they work with strong or weak space bounds. On the other hand, for deterministic, nondeterministic, and alternating machines with strong space bounds, and also for deterministic machines with weak space bounds, we show that the minimal space required for accepting a nonregular language by real-time or one-way multicounter automata is n^{ε}. All these bounds hold both for unary and general nonregular languages. Here ε represents an arbitrarily small—but fixed—real positive constant; the "space" refers to the values stored in the counters, rather than to the lengths of their binary representation.

Keywords: Space complexity · Pushdown automata
Counter automata · Real-time automata

1 Introduction and Preliminaries

The minimal amount of necessary resources is one of the fundamental research directions in complexity theory. By the space hierarchy theorem [14], we know that with a small increase in space $s(n)$ we can solve new problems that could not be solved before: if a function $s_2(n)$ grows faster than $s_1(n)$, then there exists languages that can be accepted with space bound $s_2(n)$ but not with space bound $s_1(n)$. (For more details and an advanced version, we refer the reader to [8].) This holds both for *(i) strong space bound s(n)* that refers to the space used by any computation path, on all inputs of length n, and for *(ii) weak space bound s(n)* that refers to the minimal space that is required for an accepting computation path, for all accepted inputs of length n. (Some other variants of space complexity have also been considered in the literature, see [11,17,19].)

However, there is a gap between $s_2(n) = \log\log n$ and $s_1(n) = 0$: each language accepted with space below $\log\log n$ is necessarily regular, and hence

Supported under contracts VEGA 1/0056/18 and APVV-15-0091.

accepted with no requirements on the worktape space.[1] For this reason, a worktape the size of which is bounded by $o(\log\log n)$ is not useful. This result was gradually improved, beginning with two-way deterministic Turing machines with strong space bounds and ending by an argument for two-way alternating machines with weak space bounds [1,14–16]. The first nonregular language accepted by a two-way deterministic machine using this minimal useful space, i.e., with strong space bound $O(\log\log n)$, appeared already in [14].

Moreover, unary languages need a special attention, since they may require resources that are different from those for languages built over general (or binary) alphabets [3,4,11,17]. This is because a recognizer, already having too little space to remember an input head position, must also cope with the lack of any structure along the input. The first *unary* nonregular language accepted deterministically with strong space bound $O(\log\log n)$ was presented in [2]. The two-way alternating machines using $O(\log\log n)$ space can actually be quite strong, e.g., they can recognize some unary languages the binary coded versions of which are PSPACE-complete [10].

It turns out that the minimal space bounds for *one-way* machines accepting nonregular languages are different, namely [1,14]: $\log n$ for deterministic, nondeterministic, and alternating machines with strong space bounds, and also for deterministic machines with weak space bounds, but $\log\log n$ for nondeterministic and alternating machines with weak space bounds.

Also in the one-way case we do have unary nonregular witness languages matching these lower bounds: the language $\{1^p : p \text{ is a prime}\}$ can be accepted by a one-way deterministic Turing machine with strong space bound $O(\log n)$ [19, Sect. 3.1] and the language $\mathcal{L}$ introduced by (2) in Sect. 2, by a one-way nondeterministic Turing machine with weak space bound $O(\log\log n)$ [4] (see [17]).

The minimal useful space resources for one-way machines do not change even if we require a computation in *real-time*, by machines that move the input head forward in each computation step. Clearly, all lower bounds presented above for one-way machines must also hold for real-time machines. Second, these bounds cannot be raised up: the unary nonregular witness language accepted by a real-time deterministic (hence, also nondeterministic or alternating) Turing machine with strong space bound $O(\log n)$ appeared in [20], the unary nonregular language accepted by a real-time nondeterministic (hence, also alternating) machine with weak space bound $O(\log\log n)$ in [3].

Taking into account that the above bounds cannot be decreased, a natural question arises, namely, if we cannot improve the results by the use of even simpler computational models. For example, a machine may use a simpler kind of memory, like a pushdown store or a finite number of counters.

Several results should be mentioned. First [3], some nonregular languages can be accepted by two-way deterministic pushdown automata with strong space bound $O(\log\log n)$, and also by real-time nondeterministic pushdown automata with weak space bound $O(\log\log n)$. Thus, using a pushdown store instead of a worktape does not increase the minimal useful space. Since each unary

[1] Throughout the paper, $\log x$ denotes the *binary* logarithm of x.

context-free language is regular [13], only unary regular languages are accepted by one-way nondeterministic pushdown automata, but their alternating counterparts can simulate any alternating machine that uses linear space [5].

Consider now one-way machines using one counter instead of a pushdown store. We have a real-time alternating automaton recognizing a unary nonregular language by the use of one counter with weak space bound $O(\log n)$ [3], but only unary regular languages are accepted by one-way nondeterministic machines using one counter, by argument that all unary context-free languages are regular.

The primary computational model studied in this paper is a real-time automaton recognizing a unary language by the use of a finite number of counters, but several results easily extended to more powerful models.

First, we present a unary nonregular language that can be accepted by a real-time nondeterministic automaton using four counters with weak space bound $O(\log n)$. Then, by increasing the number of counters—but keeping the real-time processing of the input—we reduce the weak space bound for this language to $O((\log n)^{\varepsilon})$, where ε represents an arbitrarily small, but fixed, real positive constant. Next, we show that this upper bound cannot be decreased: with weak space bound $(\log n)^{o(1)}$, even two-way alternating (hence, also real-time nondeterministic) multicounter automata can recognize only regular languages. This gives an answer to a question stated in [20]—namely, we have shown that real-time nondeterministic and two-way alternating multicounter automata need the *same* amount of useful space (i.e., the alternation does not help here, even if a two-way head is available). This clearly carries over to all models with computational power in between.

For completeness, we also show that $O((\log n)^{\varepsilon})$ is the minimal useful space for recognizing unary nonregular languages by two-way multicounter automata, independent of whether they are deterministic, nondeterministic, or alternating, and of whether they work with strong or weak space bounds.

Finally, we present a unary nonregular language accepted by a real-time deterministic automaton using two counters with strong space bound $O(n^{1/2})$. By increasing the number of counters while keeping the real-time processing, we reduce the strong space bound for this language to $O(n^{\varepsilon})$. Also here the achieved bound cannot be decreased to $n^{o(1)}$, neither for one-way alternating (hence, neither for deterministic or nondeterministic) multicounter machines with strong space bounds, nor for one-way deterministic multicounter machines with weak space bounds. This improves [20, Theorem 11] that presents, for each $j > 1$, a nonregular language recognized by a real-time deterministic automaton using j counters with strong space bound $O(n^{1/j})$. This result did not give a *single* witness language accepted with space $O(n^{\varepsilon})$ at all but a sequence of languages with decreasing space bounds and the size of input alphabets growing in j.

The bounds obtained in this paper are summarized in Table 2.

We assume the reader is familiar with the standard models of finite state automata and pushdown automata (see, e.g., [12,15,19]).

A *nondeterministic multicounter automaton* is a nondeterministic machine equipped with a finite state control, a read-only input tape, and, for some $\ell \geq 0$, with ℓ *counters*, containing initially zeros. The set of operations with a counter $\mathcal{C}$ consists of testing its contents for zero ($\mathcal{C} \overset{?}{=} 0$), increasing by one ($\mathcal{C} := \mathcal{C} + 1$), decreasing by one ($\mathcal{C} := \mathcal{C} - 1$), and no change ($\mathcal{C} := \mathcal{C}$). The action of the multicounter machine depends on the current state, the symbol currently scanned by the input head, and which of the counters contain/do not contain zeros. In one step, the machine changes its state, moves its head at most one position along the input tape (forward, backward, or no move), and updates its counters, independently (increase, decrease, or no change). The computation is aborted, if the machine tries to decrease a counter containing zero.[2] Under the space used by counter we mean the value stored in the counter.

A multicounter automaton is *one-way*, if it never moves its input head backward, otherwise it is *two-way.* A two-way machine has its input enclosed in between two endmarkers. A *real-time* machine is a restricted one-way variant in which the input head moves forward in each computation step.

As usual, a given computation path is *accepting*, if it starts in the initial state and halts in any accepting state. For one-way machines, acceptance requires halting in an accepting state after reading the entire input.

A *deterministic* machine can be obtained from nondeterministic version by claiming that there is allowed at most one possible transition at a time. An *alternating* machine is obtained from nondeterministic machine by partitioning the state set into the sets of existential and universal states, disjointedly. In the existential states, the machine chooses one from among possible executable steps, but in the universal states it follows all possible branches in parallel. (For more details, see, e.g., [1,5,7,9,19].)

2 Weak Space Bounds

Here we shall present the minimal space that is required for accepting a nonregular language by multicounter real-time automata with weak bounds on space.

We begin by constructing a real-time nondeterministic machine accepting a unary nonregular language by the use of four counters, working with weak space bound $O(\log n)$. To this aim, consider the following function:

$$f(n) = \text{the smallest positive integer not dividing } n.$$

To give an idea how the function $f(n)$ develops, Table 1 shows some values. It is well-known [2,6,19] that $f(n)$ can be bounded by $O(\log n)$. For our purposes, we shall need a more exact upper bound, derived in [3, Lemma 6]:

$$f(n) < 2\cdot \log n\,, \quad \text{for each } n \geq 3. \tag{1}$$

[2] Equivalently, the counter may be viewed as a special case of the pushdown store, the contents of which are always in the form $\vdash X^h$, where $\vdash$ denotes the bottom-of-pushdown-store-endmarker.

Table 1. Some values of function $f(n)$.

n	0	1	2	3	4	5	6	$\cdots$	12	$\cdots$	60	$\cdots$	420	$\cdots$	840	$\cdots$
$f(n)$	$+\infty$	2	3	2	3	2	4	<5	5	<7	7	<8	8	<9	9	$\cdots$

Next, consider the following unary language:

$$\mathcal{L} = \{1^n : \ f(n) \text{ is not equal to a power of } 2\}. \tag{2}$$

For the special case of $n = 0$, we have $f(n) = +\infty$ and $1^0 \in \mathcal{L}$. Historically, the *complement* of $\mathcal{L}$ was the first known *unary* nonregular language accepted with only $O(\log\log n)$ space, by a two-way deterministic Turing machine with strong space bound [2]. Later, in [4] (see also [17]), it was shown that the language $\mathcal{L}$ (but not its complement) can be accepted by a *one-way* nondeterministic Turing machine, with the same—but weak—space bound $O(\log\log n)$. Quite recently [3], this was improved by showing that $\mathcal{L}$ can also be accepted by a *real-time* nondeterministic machine, still keeping the weak space bound $O(\log\log n)$. Both in [4] and in [3], the machines for $\mathcal{L}$ are based on the observation that, for each $n > 0$,

1. $1^n \in \mathcal{L}$ if and only if there exist two positive integers k and i satisfying $2^i < k < 2^{i+1}$, such that $n \bmod k \neq 0$ and $n \bmod 2^i = 0$.
2. Moreover, if $1^n \in \mathcal{L}$ and $n \geq 3$, the membership can be certified by taking $k = f(n)$ and $2^i = 2^{\lfloor \log k \rfloor}$. This gives $2^i < k < 2 \cdot \log n$, by (1).

The conditions in (1) guarantee that $1^n \in \mathcal{L}$, the item (2) gives an upper bound on the values 2^i and k that certify the membership in $\mathcal{L}$. These properties allow us to construct a machine for a minor modification of $\mathcal{L}$:

Theorem 1. *There exists a unary nonregular language—namely, $\mathcal{L}' = \mathcal{L} \cdot \{1\}$, for $\mathcal{L}$ introduced by (2)—accepted by a real-time nondeterministic automaton using four counters with weak space bound $O(\log n)$.*

Proof. We first present a nondeterministic machine $\mathcal{A}$ for $\mathcal{L}$ accepting in a nonstandard way, using a set of accepting configurations rather than the set of accepting states. Then we shall replace $\mathcal{A}$ by $\mathcal{A}'$ for $\mathcal{L}'$ that accepts by classic halting in an accepting state.

At the very beginning on the given input 1^n, our nondeterministic machine $\mathcal{A}$ for the language $\mathcal{L}$ nondeterministically chooses between $n \leq 6$ and $n \geq 7$.

Short inputs with $n \leq 6$ are solved without using the counters at all.

Consider now the case of $n \geq 7$. The recognition is based on the facts presented by the items (1) and (2) above. That is, the machine $\mathcal{A}$ nondeterministically guesses an integer $k > 0$ different from a 8 power of two, takes $2^i = 2^{\lfloor \log k \rfloor}$, which ensures that $2^i < k < 2^{i+1}$, and checks whether the conditions $n \bmod k \neq 0$ and $n \bmod 2^i = 0$ are satisfied. To compute the length of the input modulo k, the machine uses two counters, denoted here by $\mathcal{C}_1$ and $\mathcal{C}_2$. Similarly, to compute n modulo 2^i, we use another two counters, $\mathcal{C}_3$ and $\mathcal{C}_4$.

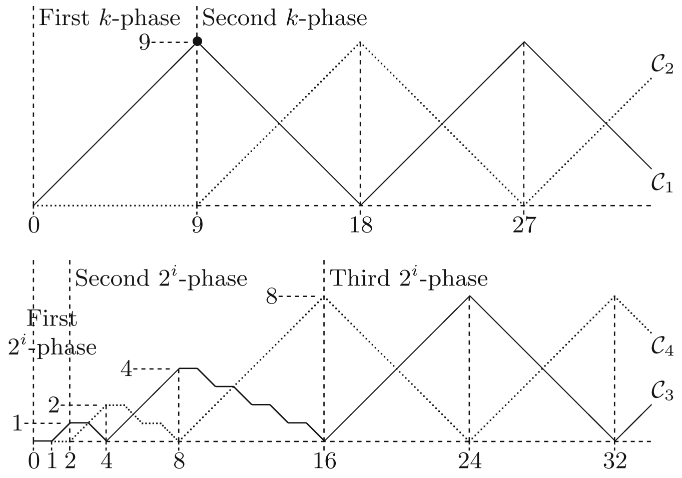


Fig. 1. The counters in the course of a computation that guesses $k = 9$, with $2^i = 8$.

Starting with counters that are all empty the main problem is the initial assignment of values k and 2^i to the counters $\mathcal{C}_1$ and $\mathcal{C}_3$, respectively, in such a way that $2^i < k < 2^{i+1}$. Recall that $\mathcal{A}$ should be a *real-time* machine and should move its input head forward *in each computation step*. To make this tricky procedure easier to follow, we shall describe manipulation with $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3, \mathcal{C}_4$ separately. The reader is referred to Fig. 1.

First, let us describe the work with the counters $\mathcal{C}_1$ and $\mathcal{C}_2$ which are used to guess $k > 0$ and to compute the length of the input modulo k. With respect to k, the computation is divided into two phases (see also Fig. 1, top):

First k-phase: Moving k symbols to the right along the input, $\mathcal{A}$ guesses an integer $k > 0$ and saves this value in $\mathcal{C}_1$. More precisely:

– In each step, reading one symbol from the input, $\mathcal{A}$ increases the counter $\mathcal{C}_1$; the counter $\mathcal{C}_2$ does not change. This is repeated in a loop, the moment of leaving this loop is chosen nondeterministically.

That is, in each step, $\mathcal{A}$ nondeterministically chooses between carrying on and leaving the first k-phase.[3] Clearly, at the moment when $\mathcal{A}$ decides to leave this phase, $\mathcal{C}_1 = k$ (the number of input symbols read so far) and $\mathcal{C}_2 = 0$.

Second k-phase: From this moment on, the counters $\mathcal{C}_1$ and $\mathcal{C}_2$ are used to compute the length of the input modulo k. This is quite straightforward, $\mathcal{A}$ switches between the following two sweep modes:

– *Odd sweep*: reading one input symbol, $\mathcal{A}$ decreases $\mathcal{C}_1$ and increases $\mathcal{C}_2$. When $\mathcal{C}_1$ becomes empty, $\mathcal{A}$ switches to even-sweep mode below.

[3] An important detail is that leaving this loop is disabled, whenever $\mathcal{C}_3 = 0$ or $\mathcal{C}_4 = 0$. This ensures that we cannot choose k be equal to a power of two—to be described later.

– *Even sweep*: reading one symbol from the input, $\mathcal{A}$ decreases $\mathcal{C}_2$ and increases $\mathcal{C}_1$. When $\mathcal{C}_2 = 0$, $\mathcal{A}$ switches to odd-sweep mode.

Thus, we keep $\mathcal{C}_1 + \mathcal{C}_2 = k$, which ensures counting the length of the input 1^n modulo k. Clearly, when the end of the input is reached, we have $\mathcal{C}_1 \neq 0 \wedge \mathcal{C}_2 \neq 0$ if and only if $n \bmod k \neq 0$.

Second, consider the work with the counters $\mathcal{C}_3$ and $\mathcal{C}_4$. These two counters are used to compute the length of the input modulo 2^i, for $2^i = 2^{\lfloor \log k \rfloor}$. Counting modulo 2^i runs in parallel with counting modulo k, described above. Among others, this means that the two procedures manipulating with $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3, \mathcal{C}_4$ share the same input head, moving this head one position to the right in each computation step. With respect to 2^i, the computation is divided into three phases (see Fig. 1, bottom).

First 2^i*-phase*: After this initialization, we set $\mathcal{C}_3 := 1$ and $\mathcal{C}_4 := 0$:

– In the first two steps, reading the first two symbols from the input, $\mathcal{A}$ increases $\mathcal{C}_3$ from 0 to 1; the counter $\mathcal{C}_4$ does not change.

Second 2^i*-phase*: This involves the standard use of the counters $\mathcal{C}_3$ and $\mathcal{C}_4$ to multiply, repeatedly, the actual value by the factor of two. This is done by switching between the following two sweep modes:

– *Odd sweep*: reading two input symbols, $\mathcal{A}$ decreases $\mathcal{C}_3$ by 1 and increases $\mathcal{C}_4$ by 2. When $\mathcal{C}_3$ becomes empty, $\mathcal{A}$ switches to even-sweep mode below.
– *Even sweep*: reading two symbols in two steps, $\mathcal{A}$ decreases $\mathcal{C}_4$ by 1 and increases $\mathcal{C}_3$ by 2. When $\mathcal{C}_4 = 0$, $\mathcal{A}$ switches to odd-sweep mode.

Thus, starting with $\mathcal{C}_3 + \mathcal{C}_4 = 1$ after the first 2^i-phase and iterating this way i sweeps, for some $i > 0$, we get $\mathcal{C}_3 + \mathcal{C}_4 = 2^i$, with either $\mathcal{C}_3 = 0$ or $\mathcal{C}_4 = 0$, depending on parity of i.

The whole process prepares to terminate when the procedure manipulating the counters $\mathcal{C}_1$ and $\mathcal{C}_2$ (described above, running in parallel) nondeterministically decides to switch from the first k-phase to the second k-phase. This means that the final value k has been fixed at this moment. When this happens, the current sweep of the second 2^i-phase (no matter whether it is odd or even) is fixed as the last one, which is kept in the finite state control. The current sweep of the second 2^i-phase is still going to be completed, that is, we carry it on until we get $\mathcal{C}_3 = 0$ or $\mathcal{C}_4 = 0$, depending on parity of i. After that, $\mathcal{A}$ switches from the second 2^i-phase to the third 2^i-phase—to be described below. More precisely, depending on parity of i, we switch either from the odd sweep of the second 2^i-phase to the even sweep of the third 2^i-phase or, vice versa, from the even sweep of the second 2^i-phase to the odd sweep of the third 2^i-phase.

Let us calculate how many symbols are read from the input in the course of the first two 2^i-phases. Let i be the total number of sweeps iterated in the second 2^i-phase—including the one in which the final value k has been fixed, i.e., during which $\mathcal{A}$ switches nondeterministically from the first k-phase to the second k-phase. It is easy to see that the second 2^i-phase reads exactly $\sum_{j=1}^{i-1} 2^j =$

$2^i - 2$ symbols in the first $i-1$ sweeps and exactly $\sum_{j=1}^{i} 2^j = 2^{i+1} - 2$ symbols in the first i sweeps. Since, by Footnote 3, the machine $\mathcal{A}$ cannot switch from the first k-phase to the second k-phase whenever $\mathcal{C}_3 = 0$ or $\mathcal{C}_4 = 0$, the final value k must be fixed *in the middle* of the i-th sweep of the second 2^i-phase. Taking into account that exactly 2 symbols were read during the first 2^i-phase, we get:

$$2^i < k < 2^{i+1}. \tag{3}$$

This implies that the input tape segment traversed in the course of the first two 2^i-phases is of length 2^{i+1}. Using (3), it is also easy to see that $82^i = 2^{\lfloor \log k \rfloor}$.

Third 2^i*-phase*: From this moment on, the counters $\mathcal{C}_3$ and $\mathcal{C}_4$ are used to compute the length of the input modulo 2^i. Since we have traversed 2^{i+1} input symbols in the first two 2^i-phases, which is an integer multiple of 2^i, and the second 2^i-phase ends with $\mathcal{C}_3 + \mathcal{C}_4 = 2^i$, with either $\mathcal{C}_3 = 0$ or $\mathcal{C}_4 = 0$ (depending on parity of i), counting modulo 2^i can be implemented in the same way as in the second k-phase, using $\mathcal{C}_3, \mathcal{C}_4$ instead of $\mathcal{C}_1, \mathcal{C}_2$:

– *Odd sweep*: reading one symbol from the input, $\mathcal{A}$ decreases $\mathcal{C}_3$ and increases $\mathcal{C}_4$. When $\mathcal{C}_3 = 0$, $\mathcal{A}$ switches to even-sweep mode below.
– *Even sweep*: reading one symbol from the input, $\mathcal{A}$ decreases $\mathcal{C}_4$ and increases $\mathcal{C}_3$. When $\mathcal{C}_4 = 0$, $\mathcal{A}$ switches to odd-sweep mode.

Thus, at the end of the input, $\mathcal{C}_3 = 0 \vee \mathcal{C}_4 = 0$ if and only if $n \bmod 2^i = 0$.

Finally, by definition, $\mathcal{A}$ accepts if it reaches the end of the input

– in the second k-phase with $\mathcal{C}_1 \neq 0 \wedge \mathcal{C}_2 \neq 0$ and, at the same time, in the third 2^i-phase with $\mathcal{C}_3 = 0 \vee \mathcal{C}_4 = 0$.

The above nondeterministic machine $\mathcal{A}$ accepts an input 1^n if and only if $1^n \in \mathcal{L}$. However, $\mathcal{A}$ accepts in a somewhat nonstandard way, by getting to an accepting configuration (situation) rather than to an accepting state. Now we replace $\mathcal{A}$ by a new machine $\mathcal{A}'$ that simulates $\mathcal{A}$ but,

– each time $\mathcal{A}$ gets to an accepting situation, i.e., each time $\mathcal{A}$ is in the second k-phase with $\mathcal{C}_1 \neq 0 \wedge \mathcal{C}_2 \neq 0$ and, at the same time, in the third 2^i-phase with $\mathcal{C}_3 = 0 \vee \mathcal{C}_4 = 0$, the machine $\mathcal{A}'$ nondeterministically decides whether to carry on the simulation or to stop by reading one more symbol from the input and switching to its unique final state. The same applies to the special path handling short inputs of length $n \leq 6$.

This changes $\mathcal{A}$ for $\mathcal{L}$ to $\mathcal{A}'$ that accepts $\mathcal{L}' = \mathcal{L} \cdot \{1\}$ by classic halting in a unique accepting state at the end of the input. $\square$

As the next step, we shall show that the space used in Theorem 1 can be decreased to $O((\log n)^{\varepsilon})$, where ε represents an arbitrarily small, but fixed, real positive constant, but using more than four counters. To this aim, let us show that we can save a substantial amount of space by simulating one counter with the help of two counters, preserving the real-time processing of the input.

Theorem 2. *Each automaton $\mathcal{A}$ using a counter $\mathcal{C}$ the space of which is bounded by $s(n)$—not excluding that $\mathcal{A}$ may also utilize some other computational resources—can be replaced by an equivalent automaton $\mathcal{A}'$ utilizing the same computational resources as $\mathcal{A}$ but, instead of the counter $\mathcal{C}$, it uses two counters $\mathcal{C}_1$ and $\mathcal{C}_2$, both of them with space bound $s'(n) \leq O(s(n)^{1/2})$. Moreover, if $\mathcal{A}$ is a real-time machine, so is $\mathcal{A}'$.*

Proof. A counter $\mathcal{C}$ with space bound $s(n)$ can be simulated by two counters $\mathcal{C}_1, \mathcal{C}_2$ with space bound $O(s(n)^{1/2})$, based on the one-to-one correspondence between $\mathbb{N}$, the set of natural numbers, and $\mathbb{N} \times \mathbb{N}$. (See, e.g., [19, Theorem 3.2.3].) The main problem to be fixed here is making such simulation real-time: each single-step operation with $\mathcal{C}$ should be simulated in one step, since the new machine $\mathcal{A}'$ is forced to move its input head forward in each computation step.

The current value in the counter $\mathcal{C}$ of the original machine $\mathcal{A}$ will be represented by three quantities, namely, by two values stored in the counters $\mathcal{C}_1, \mathcal{C}_2$, and by a sweep mode $s \in \{\mathsf{even}, \mathsf{odd}\}$, kept in the finite state control. This doubles the number of finite control states of the original machine. Initially, $\mathcal{A}'$ starts in even sweep mode and both counters are empty,[4] that is, $s = \mathsf{even}$, $\mathcal{C}_1 = 0$, and $\mathcal{C}_2 = 0$. All other computational resources of the original machine $\mathcal{A}$—including its current final control state and its head along the input tape—are manipulated in a straightforward way.

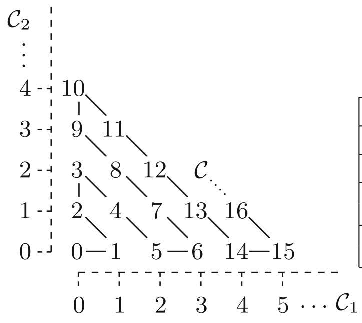
The main idea behind this implementation of a counter is simple: when $\mathcal{A}'$ simulates several operations $\mathcal{C} := \mathcal{C} + 1$ in a row, it imitates a sweep along a line in a two-dimensional grid. In the course of this sweep, the current coordinates in the grid are given by $\langle \mathcal{C}_1, \mathcal{C}_2 \rangle$ and satisfy, for some $v \geq 0$, the condition $\mathcal{C}_1 + \mathcal{C}_2 = v$. When, sweeping along this line, $\mathcal{A}'$ hits either of the axes of the coordinate system (that is, when $\mathcal{C}_1 = 0$ or $\mathcal{C}_2 = 0$), the machine switches the sweep mode, from even to odd or vice versa. After that, $\mathcal{A}'$ starts a sweep that goes to the other axis in the grid along a new line, keeping this time a new invariant, $\mathcal{C}_1 + \mathcal{C}_2 = v + 1$ instead of $\mathcal{C}_1 + \mathcal{C}_2 = v$. The operation $\mathcal{C} := \mathcal{C} - 1$ is implemented as backing up along this zigzag trajectory towards the origin $\langle 0, 0 \rangle$. The point $\langle \mathcal{C}_1, \mathcal{C}_2 \rangle = \langle 0, 0 \rangle$ corresponds to $\mathcal{C} = 0$. Figure 2 (left) reflects the main idea, Fig. 2 (right) displays a more detailed transition table.

Both $\mathcal{C}_1$ and $\mathcal{C}_2$ are bounded by $\lceil (2{\cdot}\mathcal{C})^{1/2} \rceil \leq O(s(n)^{1/2})$. It is also easy to see that one step of the original machine is simulated by exactly one step. $\square$

By the use of the previous theorem, we can decrease the space requirements for arbitrarily many counters, down to $O(s(n)^{1/2})$, preserving the real-time processing of the input.

Lemma 3. *For each $\ell \geq 0$, each automaton $\mathcal{A}$ using ℓ counters with space bound $s(n)$ can be replaced by an equivalent automaton $\mathcal{A}'$ using $2{\cdot}\ell$ counters with space bound $s'(n) \leq O(s(n)^{1/2})$. This holds for all computational models listed in the statement of Theorem 2. In particular, if $\mathcal{A}$ is a real-time machine, so is $\mathcal{A}'$.*

[4] Throughout the entire computation, $s = \mathsf{even}$ if and only if $\mathcal{C}_1 + \mathcal{C}_2$ is even.



$\mathcal{C} := \mathcal{C}+1$					
If			Then		
s	$\mathcal{C}_1$	$\mathcal{C}_2$	s	$\mathcal{C}_1$	$\mathcal{C}_2$
even	$*$	$>$	even	$+$	$-$
		$=$	odd	$+$	$\circ$
odd	$>$	$*$	odd	$-$	$+$
	$=$		even	$\circ$	$+$

$\mathcal{C} := \mathcal{C}-1$					
If			Then		
s	$\mathcal{C}_1$	$\mathcal{C}_2$	s	$\mathcal{C}_1$	$\mathcal{C}_2$
even	$>$	$*$	even	$-$	$+$
	$=$	$>$	odd	$\circ$	$-$
odd	$*$	$>$	odd	$+$	$-$
	$>$	$=$	even	$-$	$\circ$

Fig. 2. A real-time simulation of one counter by two smaller counters.

Proof. Let $\mathcal{A}$ be a machine equipped with ℓ counters, denoted here by $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$. Each of these counters works with space bound $s(n)$. Now, by repeated application of Theorem 2, for $i = 1, \ldots, \ell$, we can replace the i-th counter $\mathcal{C}_i$ by a pair of new counters, $\mathcal{C}'_i$ and $\mathcal{C}''_i$. This also requires to keep a sweep mode $s_i \in \{\mathsf{even}, \mathsf{odd}\}$ in the finite state control. Both $\mathcal{C}'_i$ and $\mathcal{C}''_i$ work with space bound $s'(n) \leq O(s(n)^{1/2})$. Thus, for each $i \in \{1, \ldots, \ell\}$, the intermediate machine uses the counters $\mathcal{C}_{i+1}, \ldots, \mathcal{C}_\ell$ working with space bound $s(n)$ together with the counters $\mathcal{C}'_1, \ldots, \mathcal{C}'_i$ and $\mathcal{C}''_1, \ldots, \mathcal{C}''_i$ working with space bound $s'(n)$. In addition, the intermediate machine manipulates all sweep modes $s_1, \ldots, s_i \in \{\mathsf{even}, \mathsf{odd}\}$ in the finite state control, simultaneously. In the end, for $i = \ell$, we obtain an equivalent machine $\mathcal{A}'$ using $2{\cdot}\ell$ counters, all of them working with space bound $s'(n) \leq O(s(n)^{1/2})$. The price we pay is that $\mathcal{A}'$ uses $\|Q'\| = 2^\ell{\cdot}\|Q\|$ states, where $\|Q\|$ denotes the number of states in the original machine $\mathcal{A}$. □

The space reduction presented in Lemma 3 can be improved, by repeating the application of this lemma h times:

Lemma 4. *For each $h \geq 0$ and each $\ell \geq 0$, each automaton $\mathcal{A}$ using ℓ counters with space bound $s(n)$ can be replaced by an equivalent automaton $\mathcal{A}'$ using $2^h{\cdot}\ell$ counters with space bound $s'(n) \leq O(s(n)^{1/2^h})$. This holds for all computational models listed in the statement of Theorem 2. In particular, if $\mathcal{A}$ is a real-time machine, so is $\mathcal{A}'$.*

By taking $h = \lceil \log(1/\varepsilon) \rceil$ in the above lemma, we then get:

Theorem 5. *For arbitrarily small, but fixed, real constant $\varepsilon > 0$, each automaton $\mathcal{A}$ using ℓ counters with space bound $s(n)$ can be replaced by an equivalent automaton $\mathcal{A}'$ using $\ell' < 2\ell/\varepsilon$ counters with space bound $s'(n) \leq O(s(n)^\varepsilon)$. This holds for all computational models listed in the statement of Theorem 2. In particular, if $\mathcal{A}$ is a real-time machine, so is $\mathcal{A}'$.*

We are now ready to establish one of the main results, showing that $(\log n)^\varepsilon$ is the smallest possible weak space bound for accepting unary nonregular languages by multicounter real-time automata:

Theorem 6. *There exists a unary nonregular language—namely, $\mathcal{L}' = \mathcal{L}\cdot\{1\}$, for $\mathcal{L}$ introduced by (2)—such that, for arbitrarily small, but fixed, real constant $\varepsilon > 0$, it can be accepted by a real-time nondeterministic automaton using $\ell' < 8/\varepsilon$ counters with weak space bound $O((\log n)^{\varepsilon})$.*

Proof. From Theorem 1 we know the language $\mathcal{L}'$ is accepted by a real-time nondeterministic automaton using four counters with weak space bound $O(\log n)$. By Theorem 5 we get, for each $\varepsilon > 0$, an equivalent real-time nondeterministic automaton using $\ell' < 8/\varepsilon$ counters with weak space bound $O((\log n)^{\varepsilon})$. □

The above upper bound cannot be decreased. That is, the constant $\varepsilon > 0$ in $O((\log n)^{\varepsilon})$ cannot be replaced by a function $r(n)$ satisfying $\lim_{n\to\infty} r(n) = 0$, even if we use a more powerful computational model—utilizing the power of alternation and/or two-way input head movement, and even if the witness nonregular language is quite arbitrary—not necessarily unary:

Theorem 7. *If a two-way alternating automaton recognizes a nonregular language, using a finite number of counters with weak space bound $s(n)$, then $s(n) \notin (\log n)^{o(1)}$.*

Proof. Suppose that some nonregular language is accepted by a two-way alternating automaton using some ℓ counters with weak space bound $s(n)$. Such machine can be replaced by an equivalent two-way Turing machine using one worktape, keeping the contents of all counters in binary, separated by a special symbol. The total length of this worktape can be bounded by $s'(n) \leq \ell\cdot(2 + \log s(n)) = \log((4\cdot s(n))^{\ell})$. By [16], if a two-way alternating Turing machine accepts a nonregular language with weak space bound $s'(n)$, then $s'(n) \notin o(\log\log n)$. Hence there exist a real constant $e_0 > 0$ and an infinite sequence of input lengths $n_1 < n_2 < n_3 < \cdots$ such that, for each $i \geq 1$, we have $s'(n_i) \geq e_0\cdot\log\log n_i = \log((\log n_i)^{e_0})$. But then $\log((4\cdot s(n_i))^{\ell}) \geq \log((\log n_i)^{e_0})$, and $s(n_i) \geq \frac{1}{4}\cdot(\log n_i)^{e_0/\ell}$, for infinitely many input lengths. Thus, $s(n) \notin (\log n)^{o(1)}$. □

We point out that the constant $e_0 > 0$ in the proof of the above theorem (given by the proof in [16]) is smaller than one and makes our lower bound dependent also on the number of states in the original multicounter machine.

3 Two-Way Devices

This section shows that also for two-way multicounter automata the minimal useful space is $O((\log n)^{\varepsilon})$. For two-way devices, the bound is independent of whether the machines are deterministic, nondeterministic, or alternating, and of whether they work with strong or weak space bounds.

Theorem 8. *There exists a unary nonregular language—namely, $\mathcal{L}$ introduced by (2)—accepted by a two-way deterministic automaton using two counters with strong space bound $O(\log n)$.*

Proof. On the given input 1^n, for $n > 0$, our two-way machine $\mathcal{A}$ equipped with the counters $\mathcal{C}_1$ and $\mathcal{C}_2$ runs a loop for $k = 2, 3, 4, \ldots$, until it finds the first k not dividing n. In order to check whether a number k divides n, the machine traverses across the entire input, counting modulo k in the same way as in the second k-phase in the proof of Theorem 1. Thus, a traversal starts from one of the endmarkers with $\mathcal{C}_1 + \mathcal{C}_2 = k$ and with either $\mathcal{C}_1 = 0$ or $\mathcal{C}_2 = 0$. If k divides n, the machine reaches the opposite endmarker with either $\mathcal{C}_1 = 0$ or $\mathcal{C}_2 = 0$ again. After that, $\mathcal{A}$ increases $\mathcal{C}_1 + \mathcal{C}_2$ from k to $k+1$ and starts a new traversal across the input in the opposite direction. This is repeated until $\mathcal{A}$ gets to the opposite endmarker with $\mathcal{C}_1 > 0 \wedge \mathcal{C}_2 > 0$.

When this happens, $\mathcal{C}_1 + \mathcal{C}_2 = k = f(n)$. From this moment on, the input head does not move—parked at one of the endmarkers. It only remains to check whether $k = \mathcal{C}_1 + \mathcal{C}_2$ is not equal to a power of 2. This is quite simple; we first move the contents of $\mathcal{C}_2$ to $\mathcal{C}_1$ by decreasing $\mathcal{C}_2$ and increasing $\mathcal{C}_1$, until we get $\mathcal{C}_2 = 0$ and $\mathcal{C}_1 = k$. Next, we divide the contents of $\mathcal{C}_1$ by two. Thus, in a loop, we decrease $\mathcal{C}_1$ by 2 and increase $\mathcal{C}_2$ by 1. When the counter $\mathcal{C}_1$ reaches zero, the original value in $\mathcal{C}_1$ has been halved, but now it is stored in $\mathcal{C}_2$. By swapping the roles of $\mathcal{C}_1$ and $\mathcal{C}_2$, we can halve this value again, ending this time with a result stored in $\mathcal{C}_1$. This halving is repeated until we find out that we have tried to halve a value that is odd. Then, $\mathcal{A}$ accepts if this last value was greater than one, i.e., the last integer division by two must not end with empty counters.

The trivial case of $1^0 \in \mathcal{L}$ is resolved at the very beginning: $\mathcal{A}$ verifies whether $n = 0$ by checking whether the first symbol to the left of the left endmarker is the right endmarker.

Clearly, by (1), the counters are bounded by $k = f(n) \leq O(\log n)$. □

By applying Theorem 5, we then get:

Theorem 9. *There exists a unary nonregular language—namely, $\mathcal{L}$ introduced by (2)—such that, for arbitrarily small, but fixed, real constant $\varepsilon > 0$, it can be accepted by a two-way deterministic automaton using $\ell' < 4/\varepsilon$ counters with strong space bound $O((\log n)^\varepsilon)$.*

Also this upper bound cannot be decreased since, by Theorem 7, even a two-way alternating multicounter automaton with weak space bound $s(n) \in (\log n)^{o(1)}$ recognizes only a regular language.

4 Strong Space Bounds and/or Determinism

Here we present the corresponding minimal space that is required for accepting nonregular languages by real-time multicounter automata with strong bounds on space, and also for real-time deterministic machines with weak bounds on space. It turns out that, for these computational models, the corresponding minimal space is n^ε. To this aim, consider the following language:

$$\mathcal{L}'' = \{1^{(n_1+n_2)\cdot(n_1+n_2+1)/2+n_1} : n_1, n_2 \in \mathbb{N},\ (n_1+n_2) \bmod 2 = 0\}. \qquad (4)$$

Theorem 10. *There exists a unary nonregular language—namely, $\mathcal{L}''$ introduced by (4)—accepted by a real-time deterministic automaton using two counters with strong space bound $O(n^{1/2})$.*

Proof. Our machine $\mathcal{A}$ traverses across the entire input and counts its length. The operation $\mathcal{C} := \mathcal{C} + 1$ is simulated by the use of two counters, $\mathcal{C}_1$, and $\mathcal{C}_2$, in the same way as in Theorem 2. This also requires to keep a sweep mode $s \in \{\mathsf{even}, \mathsf{odd}\}$ in the finite state control. By definition, let even be the only initial and, at the same time, the only final state of $\mathcal{A}$.

Clearly, $\mathcal{A}$ accepts if and only if it halts with $s = \mathsf{even}$ after reading the entire input, with any values $\mathcal{C}_1 \geq 0$ and $\mathcal{C}_2 \geq 0$ in the counters. But (see also Footnote 4) this configuration is reached for each $\mathcal{C}_1, \mathcal{C}_2 \in \mathbb{N}$ such that $\mathcal{C}_1 + \mathcal{C}_2$ is even. This condition gives $\delta = \mathcal{C}_1$ and hence the desired configuration is reached after reading $\mathcal{C}$ symbols from the input, where

$$\mathcal{C} = \sum_{v=0}^{\mathcal{C}_1+\mathcal{C}_2-1}(v+1) + \delta = \sum_{v=1}^{\mathcal{C}_1+\mathcal{C}_2} v + \mathcal{C}_1 = (\mathcal{C}_1 + \mathcal{C}_2)\cdot(\mathcal{C}_1 + \mathcal{C}_2 + 1)/2 + \mathcal{C}_1. \quad (5)$$

Thus, $\mathcal{A}$ is a real-time deterministic machine accepting $\mathcal{L}''$. Moreover, as shown in the proof of Theorem 2, we have $\mathcal{C}_1 + \mathcal{C}_2 < \lceil (2\cdot\mathcal{C})^{1/2} \rceil = \lceil (2\cdot n)^{1/2} \rceil \leq O(n^{1/2})$.

In only remains to show that $\mathcal{L}''$ accepted by $\mathcal{A}$ is not regular. First, for any given $h > 0$, take $\mathcal{C}_1 = 2\cdot h$ and $\mathcal{C}_2 = 0$. Since $\mathcal{C}_1 + \mathcal{C}_2 = 2\cdot h$ is even, the machine $\mathcal{A}$ reaches these two values in the counters with $s = \mathsf{even}$, after reading $\mathcal{C} = h\cdot(2h+3)$ symbols from the input—this value $\mathcal{C}$ is obtained by using $\mathcal{C}_1 = 2\cdot h$ and $\mathcal{C}_2 = 0$ in (5). Thus, for each $h > 0$, the input $1^{h\cdot(2h+3)}$ is accepted but, since $s = \mathsf{even}$ and $\mathcal{C}_2 = 0$, the procedure presented in the proof of Theorem 2 switches the sweep mode to odd after reading one more symbol from the input. Moreover, the sweep mode will not change in the subsequent $\mathcal{C}_1 + \mathcal{C}_2 = 2\cdot h$ steps, and hence $\mathcal{A}$ rejects $1^{h\cdot(2h+3)+i}$, for each $i = 1, \ldots, 2\cdot h + 1$. Consequently, this language cannot be accepted by a deterministic finite state automaton with fewer than $2\cdot h + 2$ states, since, after getting into a cycle the length which is bounded by $2\cdot h + 1$, such automaton cannot accept one input and then reject the next $2\cdot h + 1$ inputs in a row. Since $h > 0$ can be chosen arbitrarily large, this rules out all finite state automata. □

Also here the space requirements can be decreased by using more counters, by application of Theorem 5 on the machine constructed in Theorem 10:

Theorem 11. *There exists a unary nonregular language—namely, $\mathcal{L}''$ introduced by (4)—such that, for arbitrarily small, but fixed, real constant $\varepsilon > 0$, it can be accepted by a real-time deterministic automaton using $\ell' < 4/\varepsilon$ counters with strong space bound $O(n^{\varepsilon})$.*

The above upper bound is optimal and cannot be decreased for one-way machines with strong space bounds, even if we use the power of alternation. The same holds for one-way deterministic machines, even if we use a less restrictive weak space bound, not taking into account space used on inputs that are rejected:

Table 2. Minimal space used by multicounter automata accepting nonregular languages. All these bounds are tight both for unary and general languages.

	Strong[a]	Weak[a]	Two-way[b]
Deterministic	n^{ε}	n^{ε}	$(\log n)^{\varepsilon}$
Nondeterministic	n^{ε}	$(\log n)^{\varepsilon}$	$(\log n)^{\varepsilon}$
Alternating	n^{ε}	$(\log n)^{\varepsilon}$	$(\log n)^{\varepsilon}$

[a]Both real-time and one-way
[b]Both strong and weak

Theorem 12. *If a one-way alternating automaton recognizes a nonregular language, using a finite number of counters with strong space bound $s(n)$, then $s(n) \notin n^{o(1)}$. The same holds for weak space bounds in the case of one-way deterministic multicounter machines.*

Proof. The known lower bounds for accepting nonregular languages state that $s'(n) \notin o(\log n)$ both for one-way alternating Turing machines with strong space bounds and for one-way deterministic Turing machines with weak space bounds ([1,14], [19, Sect. 5.2]). The rest of argument mirrors the proof of Theorem 7, using n instead of $\log n$ the lower bound $s'(n) \notin o(\log n)$ for one-way Turing machines gives us the lower bound $s(n) \notin n^{o(1)}$ for one-way multicounter machines. □

5 Concluding Remarks

We have studied the minimal useful space that is required for recognizing nonregular languages by automata using a finite number of counters. The primary computational model was a real-time machine recognizing a unary language, but several results easily extended to a more general setting. All tight bounds are summarized in Table 2. The results in this table are derived by combining the upper bounds obtained in Theorems 6, 9, and 11 with the lower bounds obtained in Theorems 7 and 12, using also some trivial relations between upper and lower bounds for weaker and stronger computational models.

Both for unary and general nonregular languages, and both for real-time and one-way multicounter machines, the tight bounds do not differ. Allowing an unrestricted number of computation steps in between two forward moves along the input does not help to decrease the minimal useful space. Neither does the use of alternation instead of nondeterminism, for the same computational model.

We conjecture that if we fix the number of counters to some constant ℓ, all bounds n^{ε} in Table 2 will change to $\Theta(n^{1/\ell})$ while $(\log n)^{\varepsilon}$ to $\Theta((\log n)^{1/\ell})$. The argument would require a more efficient use of counters in upper bounds and, at the same time, a more precise analysis of the lower bounds.

However, we cannot exclude the possibility that, with the same number of counters, alternation may become more powerful, especially for small values ℓ, below 4. For example, we know a real-time alternating automaton recognizing a unary nonregular language by the use of one counter with weak space bound $O(\log n)$ [3], but only unary regular languages are accepted by one-way non-deterministic machines using one counter. Second, it is well known that each recursively enumerable language can be accepted by a one-way deterministic automaton with two counters [18] (see also [15]), but the values in the counters are double-exponential in space used by the original Turing machine, and hence such simulation is far from being real-time. The best known upper bound for a real-time deterministic automaton recognizing a unary nonregular language by the use of two counters is $O(n^{1/2})$, presented in Theorem 10. For two-way deterministic machines using two counters, this bound drops to $O(\log n)$, in Theorem 8. It is not clear whether these bounds cannot be decreased by the use of nondeterminism. For a small fixed number of counters, it is even not clear whether the bounds required for recognizing unary and general (or binary) nonregular languages do differ.

References

1. Alberts, M.: Space complexity of alternating Turing machines. In: Budach, L. (ed.) FCT 1985. LNCS, vol. 199, pp. 1–7. Springer, Heidelberg (1985). https://doi.org/10.1007/BFb0028785
2. Alt, H., Mehlhorn, K.: A language over a one symbol alphabet requiring only $O(\log\log n)$ space. SIGACT News **7**, 31–33 (1975)
3. Bednárová, Z., Geffert, V., Reinhardt, K., Yakaryılmaz, A.: New results on the minimum amount of useful space. Int. J. Found. Comput. Sci. **27**, 259–281 (2016)
4. Bertoni, A., Mereghetti, C., Pighizzini, G.: Strong optimal lower bounds for Turing machines that accept nonregular languages. In: Wiedermann, J., Hájek, P. (eds.) MFCS 1995. LNCS, vol. 969, pp. 309–318. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60246-1_137
5. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. J. Assoc. Comput. Mach. **28**, 114–133 (1981)
6. Freedman, A., Ladner, R.: Space bounds for processing contentless inputs. J. Comput. Syst. Sci. **11**, 118–128 (1975)
7. Geffert, V.: A hierarchy that does not collapse: alternations in low level space. RAIRO Inform. Théor. Appl. **28**, 465–512 (1994)
8. Geffert, V.: Space hierarchy theorem revised. Theor. Comput. Sci. **295**, 171–187 (2003)
9. Geffert, V.: Alternating space is closed under complement and other simulations for sublogarithmic space. Inform. Comput. **253**, 163–178 (2017)
10. Geffert, V.: Unary coded PSPACE-complete languages in ASPACE(loglog n). In: Weil, P. (ed.) CSR 2017. LNCS, vol. 10304, pp. 141–153. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58747-9_14
11. Geffert, V., Mereghetti, C., Pighizzini, G.: Sublogarithmic bounds on space and reversals. SIAM J. Comput. **28**, 325–340 (1999)
12. Geffert, V., Yakaryılmaz, A.: Classical automata on promise problems. Discrete Math. Theor. Comput. Sci. **17**, 157–180 (2015)
13. Ginsburg, S., Rice, H.: Two families of languages related to ALGOL. J. Assoc. Comput. Mach. **9**, 350–371 (1962)

14. Hartmanis, J., Lewis II, P., Stearns, R.: Hierarchies of memory limited computations. In: IEEE Conference on Record on Switching Circuit Theory and Logical Design, pp. 179–190 (1965)
15. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston (2001)
16. Iwama, K.: ASPACE($o(\log\log n)$) is regular. SIAM J. Comput. **22**, 136–146 (1993)
17. Mereghetti, C.: Testing the descriptional power of small Turing machines on nonregular language acceptance. Int. J. Found. Comput. Sci. **19**, 827–843 (2008)
18. Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)
19. Szepietowski, A.: Turing Machines with Sublogarithmic Space. LNCS, vol. 843. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58355-6
20. Yakaryılmaz, A., Say, A.: Tight bounds for the space complexity of nonregular language recognition by real-time machines. Int. J. Found. Comput. Sci. **24**, 1243–1253 (2013)