Uncountable Realtime Probabilistic Classes

Abuzer Yakaryilmaz, Maksims Dimitrijevs

Uncountable realtime probabilistic classes

Maksims Dimitrijevs, Abuzer Yakaryılmaz*

University of Latvia, Faculty of Computing
Raiņa bulvāris 19, Rīga, LV-1586, Latvia

md09032@lu.lv, abuzer@lu.lv

Abstract. We investigate the minimum cases for realtime probabilistic machines that can define uncountably many languages with bounded error. We show that logarithmic space is enough for realtime PTMs on unary languages. On binary case, we follow the same result for double logarithmic space, which is tight. When replacing the worktape with some limited memories, we can follow uncountable results on unary languages for two counters.

1 Introduction

When using uncountable transitions, bounded-error probabilistic and quantum models can recognize uncountably many languages [1, 8]. It is interesting to identify the minimum resources that are sufficient to follow this result. Some of the known results [8, 3] are as follows:

- Uncountably many unary languages can be defined by poly-time double log-space probabilistic Turing machines (PTMs) and linearithmic ($O(n \log n)$) time log-space one-way PTMs.
- Uncountably many k-ary languages ($k > 1$) can be defined by poly-time constant-space quantum Turing machines, linear-time linear-space two-way probabilistic counter machines, and arbitrarily small but non-constant-space PTMs.

In this paper, we investigate *realtime* probabilistic models that read the input in a streaming mode such that there is no pause on the input symbols. (This is also referred as strict realtime.) On general alphabets, it is known that bounded-error one-way PTMs cannot recognize any nonregular language in space $o(\log \log n)$ [5]. Here we show that $O(\log \log n)$-space is enough for realtime PTMs to define uncountably many languages. Therefore, this bound is tight for general alphabets. On unary alphabet, we follow the same result for $O(\log n)$ space and we leave open whether realtime PTMs can recognize any unary nonregular languages in $o(\log n)$ space. Lastly, we follow the same result for unary realtime probabilistic automata with counters and we show that two counters are sufficient. It is known that one counter is not enough since unary

* Yakaryılmaz is partially supported by ERC Advanced Grant MQC.

one-way probabilistic automata with one stack can recognize only regular languages with bounded error [6]. On the other hand, the case of two stacks is trivial since a work tape can be simulated by two stacks. We leave open to determine the minimum number of counters that use sublinear or sublogarithmic space on the counters.

In the next section, we present some background to follow the rest of the paper and then we present our results in Section 3 under two subsections. We first present the results for unary languages (Section 3.1), and then for general alphabet languages (Section 3.2).

2 Background

We assume the reader is familiar with the basics of complexity theory and automata theory. Throughout the paper, Σ not containing ¢ (the left end-marker) and \$ (the right end-marker) denotes the input alphabet, $\tilde{\Sigma}$ is the set $\Sigma \cup \{¢, \$\}$, Γ not containing blank symbol denotes the work tape alphabet, $\tilde{\Gamma}$ is the set $\Gamma \cup \{\text{blank symbol}\}$, and Σ^* is set of all strings obtained from the symbols in Σ including the empty string.

Formally, a realtime PTM P is a 7-tuple

$$P = (S, \Sigma, \Gamma, \delta, s_1, s_a, s_r),$$

where S is the set of finite internal states, $s_1 \in S$ is the initial state, $s_a \in S$ and $s_r \in S$ ($s_a \neq s_r$) are the accepting and rejecting states, respectively, and δ is the transition function

$$\delta : S \times \tilde{\Sigma} \times \tilde{\Gamma} \times S \times \tilde{\Gamma} \times \{\leftarrow, \downarrow, \rightarrow\} \rightarrow [0, 1]$$

that governs the behaviour of P as follows: When P is in state $s \in S$, reads symbol $\sigma \in \tilde{\Sigma}$ on the input tape, and reads symbol $\gamma \in \tilde{\Gamma}$ on the work tape, it enters state $s' \in S$, writes $\gamma' \in \tilde{\Gamma}$ on the cell under the work tape head, and then the work tape head is updated with respect to $d \in \{\leftarrow, \downarrow, \rightarrow\}$ with probability

$$\delta(s, \sigma, \gamma, s', \gamma', d),$$

where "$\leftarrow$" ("$\downarrow$" and "$\rightarrow$") means the head is moved one cell to the left (the head does not move and the head is moved one cell to the right). Note that input head can only perform "$\rightarrow$" moves. To be well-formed PTM, the following condition must be satisfied: for each triple $(s, \sigma, \gamma) \in S \times \tilde{\Sigma} \times \tilde{\Gamma}$,

$$\sum_{s' \in S, \gamma' \in \tilde{\Gamma}, d \in \{\leftarrow, \downarrow, \rightarrow\}} \delta(s, \sigma, \gamma, s', \gamma', d) = 1.$$

The computation starts in state s_1, and any given input, say $w \in \Sigma^*$, is read as ¢w\$ from the left to the right symbol by symbol, and the computation is terminated and the given input is accepted (rejected) if P enters s_a (s_r). It must be guaranteed that the machine enters a halting state after reading \$.

The space used by P on a given input is the number of all cells visited on the work tape during the computation with some non-zero probability. The machine P is called to be $O(s(n))$ space bounded machine if it always uses $O(s(n))$ on any input with length n.

If (realtime) P is allowed to spend more than one step on an input symbol, then it is called one-way. Formally, its transition function is extended by the move of the input head with $\{\downarrow, \rightarrow\}$ in each transition, and then, the well-formed condition is updated accordingly.

Moreover, any PTM without work tape is called probabilistic finite automaton (PFA).

A counter is a special type of memory containing only the integers. Its value is set to zero at the beginning. During the computation, its status (whether its value is zero or not) can be read similar to reading blank symbol or non-blank symbol on the work tape, and then its value is incremented or decremented by 1 or not changed similar to the position update of the work head. (A counter can be seen as a unary stack.)

A realtime probabilistic automaton with k counters (PkCA) is a realtime PTM having k counters instead of a working tape. In each step, instead of reading the symbol under the work tape head, it checks the statuses of all counters; and then, it updates the value of each counter by a value from $\{-1, 0, 1\}$ instead of updating the content of the work tape.

The language L is said to be recognized by a PTM with error bound ϵ ($0 \leq \epsilon < 1/2$) if every member of L is accepted with probability at least $1 - \epsilon$ and every non-member of L ($w \notin L$) is accepted with probability not exceeding ϵ.

We denote the set of integers $\mathbb{Z}$ and the set of positive integers $\mathbb{Z}^+$. The set $\mathcal{I} = \{I \mid I \subseteq \mathbb{Z}^+\}$ is the set of all subsets of positive integers and so it is an uncountable set (the cardinality is $\aleph_1$) like the set of real numbers ($\mathbb{R}$). The cardinality of $\mathbb{Z}$ or $\mathbb{Z}^+$ is $\aleph_0$ (countably many).

The membership of each positive integer in any $I \in \mathcal{I}$ can be represented as a binary probability value:

$$p_I = 0.x_1 01 x_2 01 x_3 01 \cdots x_i 01 \cdots , \qquad x_i = 1 \leftrightarrow i \in I.$$

3 Our results

In our proof we use a fact presented in our previous paper [3].

Fact 1 *[3] Let $x = x_1 x_2 x_3 \cdots$ be an infinite binary sequence. If a biased coin lands on head with probability $p = 0.x_1 01 x_2 01 x_3 01 \cdots$, then the value x_k can be determined with probability at least $\frac{3}{4}$ after 64^k coin tosses.*

The proof of this fact involves the analysis of probabilistic distributions for the number of heads after tossing 64^k coins that land on the head with probability p. The $(3 \cdot k + 3)$-th bit from the right in obtained number of heads is equal to x_k with probability at least $\frac{3}{4}$.

3.1 Unary languages

In [9], it was shown that realtime deterministic Turing machines (DTMs) can recognize unary nonregular languages in $O(\log n)$ space. By adopting the technique given there, we can show that bounded-error realtime PTMs can recognize uncountably many unary languages.

Theorem 1. *Bounded-error realtime unary PTMs can recognize uncountably many languages in $O(\log n)$ space.*

Proof. We start with defining a unary nonregular language that can be recognized by bounded-error log-space realtime PTMs:

$$\texttt{ULOG} = \{0^{k_i} \mid k_1 = 64 \cdot 28 \text{ and } k_i = k_{i-1} + 64^i \cdot (18i + 10) \text{ for } i > 1\},$$

where each member is defined recursively. Since it is not a periodic language, ULOG is nonregular.

For any $I \in \mathcal{I}$, we define the following language:

$$\texttt{ULOG}(\texttt{I}) = \{a^{k_i} \mid a^{k_i} \in \texttt{ULOG} \text{ for } i \geq 1 \text{ and } i \in I\}.$$

We describe a bounded-error log-space PTM for ULOG(I), say P_I. Then, we can follow the proof since there is a bijection (one-to-one and onto) between $I \in \mathcal{I}$ and ULOG(I) and $\mathcal{I}$ is an uncountable set.

The PTM P_I uses a coin landing on head with probability

$$p_I = 0.x_1 01 x_2 01 x_3 01 \cdots x_i 01 \cdots ,$$

where $x_i = 1$ if and only if $i \in I$. The aim of P_I is iteratively finding the values of $x_1, x_2, \ldots$ with high probability. If all input is read before reaching a decision on one of these values, then the input is always rejected.

During the computation, P_I uses two binary counters on the work tape. At the beginning, the iteration number is one, $i = 1$. The machine initializes the work tape as "#000000#000000#" by reading 15 $(= 9 \cdot 1 + 5 + 1)$ symbols from the input (after 15-th symbol the working tape head is placed on the first zero to the left from the third #). We name the separator symbols #s for the counters as the first, second, and third ones from left to the right. The first (second) counter is kept between the last (first) two #s.

By using the first counter, the machine counts up to 64^i and so meanwhile also tosses 64^i coins. By using the second counter, it counts the number of heads. The value of each counter can be easily increased by 1 when the working tape head passes on the counters from right to left once. Thus, when the working tape head is on the third #, it goes to the first #, and meanwhile increases the value of the first counter by 1, then tosses its coin, and, if it is a head, it also increases the value of the second counter. After tossing 64^i coins, the machine uses the leftmost value of the second counter as its answer for x_i. Once this decision is read from the work tape and immediately after the working tape head is placed on the first #, the current iteration is finished. If (i) an iteration is finished,

(ii) there is no more symbol remaining to be read from the input, and (iii) the decision is positive, then the input is accepted, which is the single condition to accept the input. After an iteration is finished, the next one starts and each counter is initialized appropriately and then the same procedure is repeated as long as there are some input symbols to be read.

Since the input is read in realtime mode, the number of computational steps is equal to the length of the input plus two (the end-markers). Now, we provide the details of each iteration step so that we can identify which strings are accepted by P_I.

At the beginning of the i-th iteration, the working tape head is placed in the first # and the contents of the counters are as follows:

$$\# \underbrace{0\cdots0}_{3(i-1)+3} \#\underbrace{0\cdots0}_{6(i-1)}\#.$$

By reading $9i+5+1$ symbols from the input, the counters are initialized for the current iteration as

$$\#\underbrace{0\cdots0}_{3i+3}\#\underbrace{0\cdots0}_{6i}\#$$

by shifting the second and third #s to 3 and 9 amounts of cells to the right (after initialization the working head is placed on the first zero to the left from the third #).

After the initialization of the counters, the working head goes to the first # and then comes back on the third # $64^i - 1$ times. In each pass from right to left, the first counter is increased by 1, the coin is flipped, and then the second counter is increased by 1 if the result is head. When all digits of the first counter are 1, which means the number of passes reaches $64^i - 1$, the working tape head makes its last pass from the third # to the first #. During the last pass, P_I flips the coin once more and then determines the leftmost digit of the second counter. Meanwhile, it also sets both counters to zeros.

By also considering the initialization step, P_I makes 64^i passes starting from the first #. So, the total number of steps is $64^i \cdot 2 \cdot (9i + 5)$ during the i-th iteration. One can easily verify that this is valid also for the case of $i = 1$.

Therefore, P_I can deterministically detect the i-th shortest member of `ULOG` after reading k_i symbols, where $k_1 = 64 \cdot (28)$ and $k_i = k_{i-1} + 64^i \cdot (18i + 10)$ for $i > 1$. Then, by using Fact 1, we can follow that P_I recognizes `ULOG(I)` with error bound $\frac{1}{4}$. □

It is known that bounded-error unary one-way PFAs with a single stack cannot recognize any nonregular language [6]. Therefore, we can check the case of having two stacks.

Corollary 1. *Bounded-error unary realtime PFA with two stacks using logarithmic amount of space can recognize uncountably many languages.*

Proof. It is a well-known fact that two stacks can easily simulate a worktape of a TMs without any delay on the running time. Therefore, by using Theorem 1, we can follow the result in a straightforward way. □

It is possible to replace stacks with counters by losing the space efficiency. We start with four counters.

Theorem 2. *Bounded-error realtime unary P4CAs can recognize uncountably many languages.*

Proof. We start with describing a realtime P4CA, say P_I, that can use a coin landing head with probability p_I for an $I \in \mathcal{I}$. Let C_i $(1 \leq i \leq 4)$ represent the values of counters.

The automaton P_I executes an iterative algorithm. We use m to denote the iteration steps. At the beginning, $m = 1$. In each iteration, 64^m coin tosses are performed. The details are as follows:

- Set $C_1 = 64^m$ and $C_2 = 4 \cdot 8^m$.
- Perform C_1 coin flips and meanwhile increase/decrease the values of C_2 and C_3 by 1. If the coin flip result is head, one of the counters is increased by 1 and the other one is decreased by 1. When one of them hits zero, update strategy is changed. Since C_3 is zero at the beginning, the first strategy is decreasing the value of C_2 and increasing the value of C_3. Thus, after each $4 \cdot 8^m$ heads, the update strategy on the counters is changed.
- When C_1 hits zero, C_2 and C_3 are equal to X and $4 \cdot 8^m - X$, and, the automaton makes its decision on x_m. If the latest strategy is decreasing the value of C_3 or $C_2 = 0$, then x_m is determined as 1. Otherwise, it is determined as 0.

The described algorithm is similar to the one that is used in the proof of Theorem 1. Here changing the update strategy between C_2 and C_3 refers to the change of bit x_m, which is changed after each $4 \cdot 8^m$ heads: it is 0 initially and then changed as $1, 0, 1, \ldots$.

At the end of the m-th iteration, we have $C_1 = 0$, $C_2 = X$, and $C_3 = 4 \cdot 8^m - X$. We initialize $(m+1)$-th iteration as follows:

- By using C_2 and C_3, we can set $C_1 = 2X + 2(4 \cdot 8^m - X) = 8^{m+1}$. Now $C_2 = C_3 = C_4 = 0$.
- Set $C_2 = C_3 = 8^{m+1}$ by setting $C_1 = 0$. Then, in a loop, until C_2 hits zero: decrease value of C_2 by 1, then transfer C_3 to C_4 (or C_4 to C_3 if at the beginning of loop's iteration $C_3 = 0$) and meanwhile add 8^{m+1} to C_1.
- $C_1 = 8^{m+1}(8^{m+1}) = 64^{m+1}$, $C_2 = 0$, $C_3 = 8^{m+1}$, $C_4 = 0$. Then set $C_2 = 4 \cdot 8^{m+1}$ by setting $C_3 = 0$.

After initializing, we execute the coin-flip procedure. Each iteration finalizes after coin-flip procedure.

The input is accepted if there is no more input symbol to be read exactly at the end of an iteration, say m-th, and x_m is guessed as 1. Otherwise, the input is always rejected.

The coin tosses part is performed in 64^m steps. The initialization part for m-th iteration is performed in $8^m + 8^m + 64^m + 4 \cdot 8^m = 64^m + 6 \cdot 8^m$ steps, where $m > 1$. The initialization part for $m = 1$ is performed in 64 steps.

Based on this analysis, we can easily formulate the language recognized by P_I, which is subset of the following language

$$\texttt{UP4CA} = \{0^{k_i} \mid k_1 = 128 \text{ and } k_i = k_{i-1} + 6 \cdot 8^i + 2 \cdot 64^i \text{ for } i > 1\}.$$

For any $I \in \mathcal{I}$, the realtime P4CA P_I can recognize the language

$$\texttt{UP4CA(I)} = \{a^{k_i} \mid a^{k_i} \in \texttt{U4PCA} \text{ for } i \geq 1 \text{ and } i \in I\}$$

with bounded error. The automaton P_I iteratively determines the values of $x_1, x_2, \ldots$ with high probability and the number of steps for each iteration corresponds with the members of U4PCA.

Since $\mathcal{I}$ is an uncountable set and there is a bijection between $I \in \mathcal{I}$ and UP4CA(I), realtime P4CAs can recognize uncountably many unary languages with bounded error. □

We can establish a similar result also for realtime P2CAs. For this purpose, we can use the well-known simulating technique of k counters by 2 counters.

Theorem 3. *Bounded-error unary realtime P2CAs can recognize uncountably many languages.*

Proof. Let P_I be the realtime P4CA described above and UP4CA(I) be the language recognized by it. Due to the realtime reading mode, the unary inputs to P_I can also be seen as the time steps. For example, P_I can be seen as a machine without any input but still making its transition after each time step. Thus, after each step it can be either in an accepting case or a rejecting case.

It is a well-known fact that two counters can simulate any number of counters with big slowdown [7]. The values of k counters, say $c_1, c_2, \ldots, c_k$, can be stored on a counter as

$$p_1^{c_1} \cdot p_2^{c_2} \cdot \cdots \cdot p_k^{c_k},$$

where $p_1, \ldots, p_k$ are some prime numbers. Then, by the help of the second counter and the internal states, it can be easily detected and stored the status of each simulated counters, and then all updates on the simulated counters are reflected one by one.

Thus, by fixing the above simulation, we can easily simulate P_I by a P2CA, say P'_I. Then, P'_I recognizes a language with bounded error, say UP2CA(I).

It is easy to see that there is a bijection between

$$\{\texttt{UP4CA(I)} \mid I \in \mathcal{I}\} \text{ and } \{\texttt{UP2CA(I)} \mid I \in \mathcal{I}\},$$

and so realtime P2CAs also recognize uncountably many languages with bounded error. Remark that for each member of UP4CA(I), the corresponding member of UP2CA(I) is much longer. □

3.2 Generic alphabet languages

Here, we focus on non-unary alphabets and establish our result for double logarithmic space. For this purpose, we use a fact given by Freivalds in [4].

Fact 2 *Let $P_1(n)$ be the number of primes not exceeding $2^{\lceil log_2 n \rceil}$, $P_2(l, N', N'')$ be the number of primes not exceeding $2^{\lceil log_2 l \rceil}$ and dividing $|N'-N''|$, and $P_3(l, n)$ be the maximum of $P_2(l, N', N'')$ over all $N' < 2^n$, $N'' \leq 2^n$, $N' \neq N''$. Then, for any $\epsilon > 0$, there is a natural number c such that $\lim_{n\to\infty} \frac{P_3(cn,n)}{P_1(cn)} < \epsilon$.*

Let $bin(i)$ denote the unique binary representation of $i > 0$ that always starts with digit 1. The language LOGLOG is composed by the strings

$$bin(1)2bin(2)2bin(3)2...2bin(s)4,$$

where $|bin(s)| = 64^k$ for some positive integer k. For any $I \in \mathcal{I}$, we define language $\texttt{LOGLOG}(\texttt{I}) = \{w \mid w \in \texttt{LOGLOG} \text{ and } k \in I\}$.

Fact 3 *Denote by $\pi(x)$ the number of primes not exceeding x. The Prime Number Theorem states that $\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$ [2].*

Theorem 4. *Bounded–error one–way PTMs can recognize uncountably many languages in $O(\log\log n)$ space.*

Proof. By modifying the one-way algorithm given in [4], we present a PTM, say $P_{c,I}$, shortly P, for language $\texttt{LOGLOG}(\texttt{I})$ for $I \in \mathcal{I}$ and for a specific c that determines the error bound. P performs different checks by using the separate parts of the work tape.

For each i, P keeps two registers storing $m = |bin(i)|$ and $m_0 = |bin(i-1)|$. After reading $bin(i)$, P checks: if $m = m_0$ or ($m = m_0+1$ and $bin(i-1)$ contained only ones), then P continues. Otherwise, P rejects the input.

For each $bin(i)$, P generates a random number of $|m| \cdot c$ bits and tests it for primality. If the generated number is not prime, the same procedure is repeated. Due to Fact 3, we can follow that the probability of picking a prime number of $|m| \cdot c$ bits is $\theta(\frac{1}{|m|\cdot c})$. Therefore, the expected time of finding a prime number is $O(|m| \cdot c)$. Assume that the generated prime number is r_i. For each $bin(i)$, P calculates $bin(i) \mod r_i$ and $bin(i+1) \mod r_i$. If $(bin(i) \mod r_i) + 1 \neq bin(i+1) \mod r_i$, P rejects the input. Otherwise, the computation continues.

After reading "4", P checks whether $m = 64^k$ for some integer $k > 0$. If so, m is written on the tape as $1(000000)^k$. If $m \neq 64^k$, then the input is rejected.

If all previous checks are successful, P tosses 64^k coins and meanwhile calculates the number of heads $\mod (8 \cdot 8^k)$, say C. If after all coin tosses, the leftmost bit of C is 1, then the input is accepted, otherwise it is rejected.

The PTM P reaches symbol "4" without rejecting with probability 1 if the input belongs to LOGLOG, and it rejects the input before reaching "4" with probability at least $1 - \epsilon$ if the input is not in LOGLOG due to Fact 2. Due to Fact 1 the membership of $k \in I$ for $\texttt{LOGLOG}(\texttt{I})$ will be computed with probability at

least $\frac{3}{4}$. Therefore language $\texttt{LOGLOG}(\texttt{I})$ is recognized correctly with probability at least $(1-\epsilon)\cdot\frac{3}{4}$, which can be arbitrarily close to $\frac{3}{4}$ by picking a suitable c.

The space used on the work tape is linear in the length of the counter for $|bin(i)|$. The value of $bin(i)$ is logarithmic to the length of input word, and so the length of the counter is double logarithmic to the input length. Therefore, the space used is in $O(\log\log n)$ throughout the computation. □

Let $L \subseteq \Sigma^*$ be a language recognized by a one-way DTM, say D, and σ be a symbol not in Σ. We can execute D in realtime reading mode on the inputs defined on $\Sigma \cup \{\sigma\}$ as follows [9]: For each original "wait" move on a symbol from Σ, the machine expects to read symbol σ. If it reads something else or there is no more input symbol, then the input is rejected. If there is more than expected σ symbol, then again the input is rejected. Thus, we can say that this modified machine recognizes a language L' and there is a bijection between L and L'. Moreover, the space and time bounds for both machines are the same.

The question is whether we can apply a similar idea for one-way PTM given above in order to get a realtime PTM. A DTM follows a single path during its computation and so the aforementioned bijection can be created in a straightforward way. On the other hand, PTMs can follow different paths with different lengths in each run. So, in order to follow a similar bijection, we need some modifications. The main modification is necessary for the task of picking the prime numbers. Except this task, the other ones can be executed with the same number of steps (remember the algorithms in the previous subsection) in every execution of the machine.

Now, we modify PTM $P_{c,I}$ in order to guarantee that each computation path uses the same amount of time steps on the same input. We represent the new PTM as $P'_{c,I}$ or shortly P'.

The PTM P' uses some registers on the work tape separated by "#":

$$\#1st\#2nd\#\cdots\#last\#.$$

- The 1st register keeps both the lengths of the counters m and m_0. If $m = x_1x_2x_3\cdots$ and $m_0 = y_1y_2y_3\cdots$, then the register keeps the values in the following way: $x_1y_1x_2y_2x_3y_3\cdots$. After reading symbol "2" it is easy to compare m and m_0 bit by bit with a single pass.
- The 2nd register keeps the number of heads for the coin-tosses, based on which the bit x_k is determined. It is set to $\lceil |m|/2 \rceil + 2$ zeros before any coin-toss procedure and it is updated accordingly when the value of m is changed.
- The 3rd register keeps the track of attempts to generate prime number, it has $|m| \cdot c$ bits.
- The 4th and 5th registers keep the prime numbers with some auxiliary numbers. Each register has $|m| \cdot c \cdot 2$ bits. If the (candidate) prime number is $r = r_1r_2r_3\cdots$ and the auxiliary number is $q = q_1q_2q_3\cdots$, then the register keeps both of them as $r_1q_1r_2q_2r_3q_3\cdots$. The machine uses r to store the prime number that is being checked or computed, and q is used to help to perform tasks with r like storing number modulo r and comparing and

copying numbers. For each $j > 0$, the machine uses 4th and 5th registers to work with prime numbers and then checks the correctness of the candidates for $bin(2 \cdot j - 1)$ and $bin(2 \cdot j)$.
- The 6th and 7th registers are the same as 4th and 5th registers, respectively. Only they are responsible for the correctness of the candidates for $bin(2 \cdot j)$ and $bin(2 \cdot j + 1)$.
- The 8th register has a number to keep track of total number of subtractions performed while checking the divisibility of r by d. It has $|m| \cdot c$ bits.
- The 9th register has twice of $\lceil |m|/2 \rceil \cdot c$ bits to keep numbers d and h (each is $\lceil |m|/2 \rceil \cdot c$ bits). If $d = d_1 d_2 d_3 \cdots$ and $h = h_1 h_2 h_3 \cdots$, then the register keeps them as $d_1 h_1 d_2 h_2 d_3 h_3 \cdots$. Both numbers are used to check whether the generated number r is prime. The machine uses d to check whether d does not divide r, such check is performed for different values d. The check is performed by making subtractions. The value of d is subtracted from r multiple times. For this operation, the machine uses h as auxiliary number.

Each member of $\texttt{LOGLOG}(\texttt{I})$ has parts $bin(i)$ at least up to $bin(2^{63})$. P' deterministically checks input up to $bin(2^{63})$ and prepares the work tape with 9 registers.

Now, we describe the steps of picking prime numbers.

For number $bin(i)$, the prime number is generated in $(6 - 2 \cdot (i \mod 2))$-th register. The number r is generated by using $|m| \cdot c$ random bits (bit by bit). After this, the primality check is performed. For this purpose, the machine checks whether r is divided by any natural number between 2 and $2^{\lceil |m|/2 \rceil \cdot c} - 1$, where $2^{\lceil |m|/2 \rceil \cdot c} > sqrt(r)$ because $r < 2^{|m| \cdot c}$. Each candidate natural number is denoted by d below. Remark that the number of ds does not depend on r and so for any candidate prime number, the primary test procedure takes the same number of steps.

To begin the check of divisibility of r by d, the value of r is copied to q bit after bit, and the value of d is copied to h bit after bit. The 8th register is initialized with zeros before check for pair r and d. Then, $2^{|m| \cdot c}$ iterations are performed. In each iteration, the values of q and h are decreased by 1, the value of 8th register is increased by 1. If only h reaches zero, d is again copied into h and the machine continues to perform iterations. When q reaches zero, if h reaches zero at the same time, the machine concludes that r is not a prime number, otherwise, r is not divisible by d. After that, P' continues to perform the iterations but without changing q and checks of value of q until the value of the 8th register reaches $2^{|m| \cdot c}$. Then, P' repeats the procedure for the next d.

If r is not divisible by any of these ds, then the procedure of finding prime is terminated successfully since r is prime, otherwise, the machine continues with the next prime candidate number since r is not a prime number.

The 3rd register counts the number of attempts to generate a prime number. It is initialized with zeros and is increased by one after each try. If P' finds a prime number before 3rd register reaches $2^{|m| \cdot c}$, P' continues performing the algorithm until the register reaches $2^{|m| \cdot c}$ by fixing the candidate with the already found prime number. If the register reaches value $2^{|m| \cdot c}$ (all bits become zeros)

and P' fails to generate a prime number, P' uses the last generated r for the modular check for pair $bin(i)$ and $bin(i+1)$. P' performs each try to generate (or process already generated) prime number in equal number of steps. For any $bin(i)$ P' performs exactly $2^{|m|\cdot c}$ such operations.

After finding and checking prime r, the machine copies r into $(7 - 2 \cdot (i \mod 2))$-th register bit by bit. To perform this operation, the machine sets q to zeros in both registers, copies the bits of r one by one, and marks the copied bit by setting the next bit in q to one.

Now, we describe how the machine calculates the value $bin(i) \mod r$. At the beginning, the register keeps r and zeros for q. Assume that $bin(i) = i_1 i_2 \cdots i_m$. When the machine reads i_j, the value of q is multiplied by 2 and increased by i_j. Therefore, all bits of q are shifted to left by one position, and the machine puts value i_j in leftmost bit. If, after this operation, $q \geq r$, then r is subtracted from q. Because both values are interleaved, it is easy to subtract r from q in one pass. In the case when $q < r$ the machine performs one pass through registers without changing the values. This ensures that each iteration for i_j is performed in equal number of steps. The machine performs the calculation while reading $bin(i)$ for the 5th and the 6th registers if $i \mod 2 = 0$, and for the 4th and the 7th registers otherwise.

After these, the machine compares the values of two modules: the 4th and the 5th registers if $i \mod 2 = 0$; the 6th and the 7th registers otherwise. This time machine sets r in both registers to zeros and marks compared bits of q's by setting bits in r to one.

If r in modular check is not prime, P' cannot guarantee that incorrect pair $bin(i)$ and $bin(i+1)$ will be rejected with probability at least $1 - \epsilon$. The probability not to generate a prime number of $|m| \cdot c$ random bits in $2^{|m|\cdot c}$ tries does not exceed $(1 - \frac{1}{|m|\cdot c})^{2^{|m|\cdot c}}$ because of Fact 3. Note that $\lim_{n\to\infty}(1 - \frac{1}{n})^n = \frac{1}{e}$, therefore $\lim_{m\to\infty}(1 - \frac{1}{|m|\cdot c})^{2^{|m|\cdot c}} = \lim_{m\to\infty} \frac{1}{e}^{\frac{2^{|m|\cdot c}}{|m|\cdot c}} = 0$. The smallest $|m|$ for which a prime number is generated is 7. By picking a suitable c, the value $(1 - \frac{1}{7\cdot c})^{2^{7\cdot c}} = \epsilon_0$ can be arbitrarily close to zero. For each $i > 0$, checking the equality of $bin(i)$ and $bin(i+1)$ by using the generated prime number is performed independently. Therefore, any incorrect pair is accepted with probability at most ϵ due to Fact 2. Since P' can fail to generate a prime number, this probability is increased to at most $\epsilon + \epsilon_0 - \epsilon \cdot \epsilon_0$. If the input belongs to $\texttt{LOGLOG}(\texttt{I})$, P' is guaranteed to not reject the input before reaching "4" on input tape. If at least one pair $bin(i)$ and $bin(i+1)$ is inacceptable, then P' rejects input right after checking this pair with probability at least $1 - \epsilon - \epsilon_0 \cdot (1 - \epsilon)$. Therefore, the error remains bounded.

The other parts of the algorithm are executed with the same number of steps in every execution of P'.

Theorem 5. *Bounded–error realtime PTMs can recognize uncountably many languages in $O(\log\log n)$ space.*

Proof. We can obtain a realtime algorithm from P', say $R_{c,I}$ or shortly R, by using aforementioned technique borrowed from [9]. Let $\texttt{LOGLOG(I)}'$ be the language recognized by R. Then, the language $\texttt{LOGLOG(I)}'$ differs from the language $\texttt{LOGLOG(I)}$ with the presence of symbols "3": for each "wait" move on "0", "1", "2" or "4" by P', R expects to read one symbol of "3". If R fails to read a symbol of "3" when it is expected, the input is rejected.

PTM P' recognizes $\texttt{LOGLOG(I)}$ in $O(\log\log n)$ space, therefore, realtime machine R recognizes $\texttt{LOGLOG(I)}'$ in $O(\log\log n)$ space and there is a bijection between $\texttt{LOGLOG(I)}$ and $\texttt{LOGLOG(I)}'$. □

In [4] Freivalds has proven that only regular languges can be recognized with one-way PTM in $o(\log\log n)$ space and with probability $p > \frac{1}{2}$. Therefore, the presented space bound is tight.

Acknowledgments

We thank to the reviewers for their helpful comments.

References

1. Adleman, L.M., DeMarrais, J., Huang, M.D.A.: Quantum computability. SIAM Journal on Computing 26(5), 1524–1540 (1997)
2. Chandrasekharan, K.: Chebyshev's theorem on the distribution of prime numbers, pp. 63–83. Springer (1968)
3. Dimitrijevs, M., Yakaryılmaz, A.: Uncountable classical and quantum complexity classes. In: Eigth Workshop on Non-Classical Models for Automata and Applications (NCMA2016). books@ocg.at, vol. 321, pp. 131–146. Austrian Computer Society (2016)
4. Freivalds, R.: Space and reversal complexity of probabilistic one-way Turing machines. In: Conference on Fundamentals of Computation Theory. pp. 159–170 (1983)
5. Freivalds, R.: Space and reversal complexity of probabilistic one-way turing machines. Annals of Discrete Mathematics 24, 39–50 (1985)
6. Kaŋeps, J., Geidmanis, D., Freivalds, R.: Tally languages accepted by Monte Carlo pushdown automata. In: RANDOM. LNCS, vol. 1269, pp. 187–195. Springer (1997)
7. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
8. Say, A.C.C., Yakaryılmaz, A.: Magic coins are useful for small-space quantum machines. Tech. Rep. TR14-159, ECCC (2016)
9. Yakaryılmaz, A., Say, A.C.C.: Tight bounds for the space complexity of nonregular language recognition by real-time machines. International Journal of Foundations of Computer Science 24(8), 1243–1253 (2013)