University of Latvia
Faculty of Computing

Maksims Dimitrijevs

Probabilistic computation and verification beyond Turing machines

Doctoral Thesis

Area: Computer Science
Sub-Area: Mathematical Foundations of Computer Science

Scientific Advisors:
Dr. sc. comp., Prof. Andris Ambainis
Dr. sc. comp. Abuzer Yakaryılmaz

Riga, 2019

Acknowledgments

First of all, I would like to thank professor Rūsiņš Freivalds for opening to me the field of computer science and showing how exciting it can be and for being my first advisor of the PhD thesis.

I would like to thank my advisor professor Andris Ambainis for supporting me during the research and helping to find a direction when I needed to change my advisor.

I would like to thank my advisor Dr. Abuzer Yakaryılmaz for helping me in the new research field, for guiding me and helping to find a lot of ways to improve. His optimism and excitement have always motivated me to go further and overcome difficulties. Under his guidance, I have learned many things about how to be a researcher and a teacher.

I would like to express my gratitude to the Faculty of Computing of the University of Latvia for the opportunities to study and to work as a research assistant, and for the financial support to attend the conferences.

I would like to thank Juris Vīksna, Markus Holzer, and Marcos Villagra for accepting to review my thesis and for time spent on reading and commenting on the thesis. I appreciate very helpful comments and recommendations of Juris Vīksna for improvement of my thesis and its summaries.

I also would like to thank the E-pasaule Ltd., especially my manager Toms Ābeltiņš and board member Stas Bervyachonok for giving me many opportunities to combine my work in industry with my studies at the university and my research during PhD studies.

I would like to express my gratitude to the anonymous reviewers of the conferences and journals, to which I have submitted my works, for their very helpful comments and suggested improvements. I would like to also thank to the program committees of the conferences where I have participated for nice experience.

I would like to thank Alexander Rivosh for his active help dealing with many bureaucratic issues and printing the copies of thesis and summaries while I was abroad.

I also would like to thank the secretaries Ruta Ikauniece, Anita Ermuša, and Ella Arša for their help regarding the organizational matters of my defense. I appreciate the patience of Promotion Council and thank to the members for agreeing for the date of the defense.

I appreciate the help of Jānis Iraids and Zita Začа with grammar and recommendations for better translation of summary in Latvian language.

I would like to express my gratitude to professor Wu Junde for inviting me to attend Summer School of Quantum Algorithm Theory at the Zhejiang University in Hangzhou, China, and to professor Tomoyuki Yamakami for inviting me to visit Fukui, Japan, and to give a lecture at the University of Fukui about my research. I thank professor Wu Junde and professor Tomoyuki Yamakami for wonderful hospitality.

I appreciate the collaboration with Dr. Kamil Khadiev and his invitation to collaborate with the Kazan Federal University.

I would like to thank my friends for their patience when I was very busy with my research and for nice time during sports activities and other adventures in free time. I would also like to thank Andrey Magera for his very friendly support.

I would like to thank my family, especially my parents, grandmother Ņina and grandfather Grigorij for their great support and help during these years of intense work and studies.

Abstract

Turing machines can recognize countably many languages. On the other hand, probabilistic and quantum machines can recognize uncountably many languages with bounded error when using real number transitions. Our motivation is to investigate probabilistic models with limited computational resources that define all languages or uncountably many languages.

We aim to investigate different bounded-error probabilistic models that can define uncountably many languages. We begin with stronger condition — we first consider minimal models that can recognize all languages. We consider probabilistic Turing machines, probabilistic counter automata, and probabilistic finite state automata with various restrictions on the input head. We also consider constant-space verifiers that interact with one and two provers and verify all languages. After that, we consider the recognition and verification of uncountably many languages with bounded error for these models. We also present new results for quantum automata models and ultrametric finite automata (which use p-adic numbers as amplitudes) that recognize uncountably many languages.

Contents

List of tables

List of symbols

$\¢$	the left end-marker
$\$$	the right end-marker
$\#$	the blank symbol
ε	the empty string
Σ	the input alphabet, which does not contain symbols $\¢$ and $\$$
$\tilde{\Sigma}$	the set $\Sigma \cup \{\¢, \$\}$
Γ	the work tape alphabet, which does not contain symbols $\¢$ and $\#$
$\tilde{\Gamma}$	the set $\Gamma \cup \{\¢, \#\}$
Υ	the communication alphabet
Σ^*	the set of all strings defined over Σ, which includes ε
Υ^*	the set of all strings defined over Υ, including ε
$\Sigma^*(i)$	the i-th element of Σ^* when lexicographically ordered, where $\Sigma^*(1) = \varepsilon$
w	the input string
$\|w\|$	the length of w
n	the length of the input, $n = \|w\|$
$w[i]$	the i-th symbol of w from the left ($1 \leq i \leq \|w\|$), for any w
$\tilde{w}$	the string $\¢w\$$
$lex(w)$	the lexicographical number of w, where $lex(\Sigma^*(i)) = i$ for any $i > 0$
$bin(i)$	the unique binary representation of $i > 0$ that always starts with digit 1

$(bin(i))^r$	the reverse binary representation of $i > 0$
ϵ	the error bound, $0 < \epsilon < \frac{1}{2}$
s_{init}	the initial state
s_{acc}	the accepting state
s_{rej}	the rejecting state
q_{init}	the initial quantum state
$a(w)$	the probability of being in s_{acc} after processing w
$r(w)$	the probability of being in s_{rej} after processing w
S	the set of internal states
S_r	the set of reading states, $s_{init} \in S_r$
S_c	the set of communicating states
S_h	the set of halting states, $S_h = \{s_{acc}, s_{rej}\}$
Q	the set of quantum states
Q_{acc}	the set of accepting quantum states
δ	the transition function
δ_r	the transition function when in a reading state
δ_c	the transition function when in a communicating state
δ_q	the transition function for quantum states
$\leftarrow$	the symbol that represents that the head is moved one cell to the left
$\downarrow$	the symbol that represents that the head does not move
$\rightarrow$	the symbol that represents that the head is moved one cell to the right
$\aleph_0$	countable cardinality
$\mathbb{Z}$	the set of integers
$\mathbb{Z}^+$	the set of positive integers
$\mathbb{R}$	the set of real numbers
$\mathbb{Q}_p$	the field of p-adic numbers
$\mathcal{I}$	the set of all subsets of positive integers, $\mathcal{I} = \{I \mid I \subseteq \mathbb{Z}^+\}$

p_I	the binary probability value that represents the membership of each positive integer for any $I \in \mathcal{I}$, $p_I = 0.x_1 01 x_2 01 x_3 01 \cdots x_i 01 \cdots , \quad x_i = 1 \leftrightarrow i \in I$
p_L	the binary probability value that represents the membership of each string for any given language $L \subseteq \Sigma^*$, $p_L = 0.x_1 01 x_2 01 x_3 01 \cdots x_i 01 \cdots , \quad x_i = 1 \leftrightarrow \Sigma^*(i) \in L$
$\mathfrak{coin}_I$	the coin landing on head with probability p_I
$\mathfrak{coin}_L$	the coin landing on head with probability p_L

List of acronyms/abbreviations

PTM	probabilistic Turing machine
2PTM	two-way probabilistic Turing machine
1PTM	one-way probabilistic Turing machine
DTM	deterministic Turing machine
PCA	probabilistic counter automaton with one counter
2PCA	two-way probabilistic counter automaton with one counter
1PCA	one-way probabilistic counter automaton with one counter
PFA	probabilistic finite automaton
2PFA	two-way probabilistic finite automaton
1PFA	one-way probabilistic finite automaton
PostPFA	postselecting probabilistic finite automaton
PostPTM	postselecting probabilistic Turing machine
PostPCA	postselecting probabilistic counter automaton with one counter
IPS	interactive proof system
QCFA	finite automaton with quantum and classical states
2QCFA	two-way finite automaton with quantum and classical states
2QCCA	two-way finite automaton with quantum and classical states with a counter
QFA	quantum finite automaton

Introduction

Relevance of the Thesis

The class of languages recognized by deterministic Turing machines (DTMs) is called recursive enumerable languages, and it forms a countable set. However, all languages form an uncountable set, and hence there exist many languages that DTMs cannot recognize.

On contrary to a DTM, a probabilistic (or quantum) model can be defined with real transition values, and hence there are uncountably many of them. Therefore, it is natural to ask whether they define uncountable classes or not, or even further to ask whether they define all languages or not.

There are two basic language recognition modes: bounded-error recognition and recognition with cutpoint (also called unbounded error). In the case of the recognition with cutpoint, any member of the language recognized by a machine is accepted with probability greater than the cutpoint, and any non-member is accepted with probability at most cutpoint. Therefore, even though the accepting probability of a member is greater than the accepting probability of a non-member, the difference can be arbitrarily small. In the case of recognition with bounded error, the minimal difference between the accepting probability of any member and that of any non-member is bounded by a fixed gap value.

With unbounded error (recognition with cutpoint), it is already known that even unary 2-state quantum finite automata (QFAs) or unary 3-state probabilistic finite automata (PFAs) recognize uncountably many languages [39, 40].[1] Therefore, probabilistic and quantum models with very small amount of resources can recognize uncountably many languages with cutpoint, even in unary case, and the existing known bounds on the required resources are tight.

On the other hand, answering the same question for bounded-error probabilistic models is not trivial. The best known upper bound was known as

[1] Remark that unary 2-state PFAs define only regular languages (see [31, 40]).

logarithmic space, and any better bound was open [36]. Regarding quantum models, the better bound (i.e., constant-space) was known, but the complete characterization was still missing [36].

Probabilistic computation with bounded error can be almost as reliable as deterministic computation. If the computation is performed with bounded error, it is possible to repeat the process of computation multiple times to lower the resulting error (see [25] for more details). In fact, it is possible to obtain arbitrarily small error bound with such approach, and resulting error probability may be even lower than the probability of hardware failure.

The verification power of the models is a fundamental concept in computational complexity theory. Therefore, besides the recognition power of computational models, their verification powers have also been widely considered. An interactive proof system (IPS) is composed by a prover and a verifier, who can communicate with each other. The aim of the verifier is to verify the correctness of the input by the help of a proof, provided by the prover. The verification power of models is usually more than their standalone computational power. As an example, we can consider a book with mathematical theorems. It would take a lot of time for someone to obtain all theorems in the book. But if the book provides proofs for the theorems, then it takes much less time to verify the correctness of the theorems.

We can consider the recognition/verification of all languages or uncountably many languages with bounded-error probabilistic models to investigate their capabilities.

Subject and Goals of the Research

The goal of this thesis is to investigate the restricted cases of different probabilistic models that can define all languages or uncountably many languages. The models have restrictions on space complexity, time complexity, memory types, or input head movements. The main task is to investigate different bounded-error probabilistic models that recognize/verify all languages or uncountably many languages, and to present better upper bounds.

Our first task is to develop a method to encode the members of a given language as a single probability, and then to describe certain experiments in order to guess the digits of this probability correctly with high probability. Our second task is to investigate bounded-error probabilistic models that can recognize/verify all languages with as few computational resources as possible. Our third task is the same but for uncountably many languages. Our last task is to investigate quantum and ultrametric automata that can recognize uncountably many languages.

Space-bounded probabilistic Turing machines (PTMs) and probabilistic counter automata are the main models that we investigate. We also consider their sweeping, postselecting, one-way, and realtime versions.

The languages recognized by PFAs with cutpoint are called stochastic languages. The set of stochastic languages is uncountable. There also exist languages that are not stochastic. Therefore, constant-space is not enough to define all languages with cutpoint, and we need additional memory to recognize all languages.

It is know that logarithmic-space PTMs can recognize uncountably many languages with bounded error [36]. Therefore, we naturally investigate whether PTMs can recognize uncountably many languages with smaller space bounds. In parallel, we also investigate the more restricted memory type — counters.

Time restrictions may reduce the computational powers of space-bounded PTMs. For example, constant-space PTMs (two-way PFAs (2PFAs)) can recognize certain nonregular languages in exponential expected time [18], but in polynomial expected time PTMs can recognize only regular languages when using $o(\log \log n)$ space, even with unrestricted transition probabilities [14]. Thus, we also consider polynomial and super-polynomial time complexities separately.

PTMs can read the given input by using their head that can move both directions (two-way). Certain restrictions can be defined on the input head (e.g., [41]). We consider sweeping head (the direction of the head can be changed only on the end-markers), one-way head (the input head cannot move to the left), realtime head (the input head must move to the right after each step). Additionally, we consider a very special case of a sweeping head (restarting): if the computation is not terminated on the right end-marker, it is restarted from the initial configuration. We remark that this special case is equivalent to having the ability of postselection [1, 49].

Furthermore, we investigate private-coin IPSs with probabilistic verifiers interacting with single or two provers [17, 15, 10]. We focus on three different types of protocols: two-way communication (the verifier can send different symbols to the verifier(s)), one-way communication (the verifier always sends the same symbol when communicating), and weak-soundness (the nonmembers may not be rejected with high probability). Similarly to recognizers, we examine the verifiers under different restrictions.

The behavior of bounded-error recognizers and verifiers can be different on unary and binary inputs. For example, bounded-error 2PFAs cannot recognize any unary nonregular language [22], however, they can recognize nonregular binary languages [18]. Therefore, we focus on unary and binary languages separately.

The Structure of the Thesis and Research Questions

The thesis contains six chapters. In the first chapter, we provide the necessary background to follow the rest of the thesis. This chapter contains notations and the definitions of all bounded-error probabilistic models that are considered in the thesis.

In the second chapter, we present how to encode the members of a given language as a single probability, and then describe certain experiments in order to guess the digits of this probability correctly with high probability. We also discuss alternative encoding here. The encoding schema and experiments given in this chapter form a technical lemma, which we call as *the main lemma* (Lemma 1) since it is used in the rest of the thesis.

In the third chapter, we focus on the recognition and verification of all languages. We begin the chapter with a straightforward application of our main lemma to the sets of all languages, and obtain linear and exponential space complexity for unary and binary languages, respectively. Then, we focus on lowering the amount of used resources by interacting with prover(s). We show that constant-space probabilistic machines can verify any language by interacting with two provers. We obtain the same result also for the case of interacting with one prover but this time the non-members may not be rejected with high probability. On the other hand, for the case that the non-members are rejected with high probability, we show that every unary (resp., binary) language is verifiable by a logarithmic-space (resp., linear-space) PTM by interacting with a prover. Additionally, we show that every language is recognized by a one-way probabilistic automaton with two counters with any error bound.

In the fourth chapter, we investigate the uncountable sets of languages. The chapter is divided into three sections, each is devoted to different bounded-error probabilistic models.

In the first section, we consider different two-way PTMs, including restricted ones. We first lower the space complexity bounds and obtain arbitrarily small non-constant space complexity for two-way PTMs to recognize uncountably many languages. After that, we improve this result by obtaining the same space complexity for PTMs with very restricted input head, and prove this result for probabilistic one-counter automata. We also prove that $O(\log \log n)$-space sweeping PTMs can recognize uncountably many unary languages. After that, we consider the verification of uncountably many binary and unary languages with constant space complexity, and we obtain the linear and quadratic expected time complexities, respectively. We also

show how to verify uncountably many unary languages in constant space in restarting realtime reading mode.

In the second section, we examine realtime PTMs. We obtain a logarithmic space bound for recognition of uncountably many unary languages, and then we obtain $O(\log \log n)$ space complexity for uncountably many binary languages. Here the double logarithmic bound is tight.

In the third section, we continue with probabilistic counter automata. We obtain certain tight bounds on space complexity and the number of counters when recognizing uncountably many languages. For example, in realtime reading mode, one counter is required for binary languages, and two counters are required for unary languages, and we present witnesses for both cases.

In the fifth chapter, we present our results on different computational models: quantum and ultrametric automata. In the first part, we give the definitions of these models, and in the second part, we present the results. We present two improvements on existing results for the recognition of uncountably many languages with bounded-error quantum automata. After that, we show that quantum automata with two states can recognize uncountably many unary languages with fixed cutpoint. Then, we show ultrametric automaton with two states for uncountably many languages.

In the last chapter, we summarize our results, list open problems, and mention about the possible directions for future research.

Description of the Methodology

The used methods are common with ones used in theoretical computer science and mathematical papers — proofs of mathematical statements such as theorems, corollaries and lemmas. We use different proof methods. Most of the proofs contain combinations of techniques and refer to some previously obtained results, either our results or results of other researchers. All the proofs provided in the thesis are constructive. We also present new techniques and new constructions.

Approbation of the Results

The results of this thesis are published in 8 publications of which 3 are indexed in Elsevier Scopus and 2 in Web of Science. One paper is accepted for publication and is indexed in Elsevier Scopus and Web of Science.

1. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Uncountable Classical and Quantum Complexity Classes.

Proceedings of Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016), books@ocg.at, vol. 321, Austrian Computer Society, 2016, pp. 131–146.

2. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Uncountable Realtime Probabilistic Classes.
   Proceedings of DCFS 2017: Descriptional Complexity of Formal Systems, Lecture Notes in Computer Science, vol. 10316, pp. 102–113, 2017.
   (indexed in Scopus, Web of Science)

3. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Postselecting probabilistic finite state recognizers and verifiers.
   Proceedings of Tenth Workshop on Non-Classical Models of Automata and Applications (NCMA 2018), books@ocg.at, vol. 332, Austrian Computer Society, 2018, pp. 65–81.

4. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Recognition of uncountably many languages with one counter.
   Short Papers of Tenth Workshop on Non-Classical Models of Automata and Applications (NCMA 2018), pp. 7–13.

5. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Verifying every language in constant space.
   Proceedings of International Workshop DLT's Satellite Workshop in Kyoto, 2018, abstract published in https://satellitedlt.sciencesconf.org/resource/page/id/10, full version to appear.

6. Maksims Dimitrijevs
   Capabilities of Ultrametric Automata with One, Two, and Three States.
   Proceedings of SOFSEM 2016: Current Trends in Theory and Practice of Computer Science, Lecture Notes in Computer Science, vol. 9587, pp. 253–264, 2016.
   (indexed in Scopus)

7. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Private-coin verification with magic coins.
   SOFSEM 2019: Current Trends in Theory and Practice of Computer Science, Proceedings of Student Research Forum, pp. 1–12, 2019.

8. Maksims Dimitrijevs and Abuzer Yakaryılmaz
   Uncountable Classical and Quantum Complexity Classes.

Proceedings of RAIRO - Theoretical Informatics and Applications (RAIRO: ITA), vol. 52 2-3-4, pp. 111-126, 2018.
DOI: https://doi.org/10.1051/ita/2018012
Invited to special issue dedicated to the conference NCMA 2016.
(indexed in Scopus, Web of Science)

9. Maksims Dimitrijevs and Abuzer Yakaryılmaz
Uncountable realtime probabilistic classes.
International Journal of Foundations of Computer Science, to appear.
(indexed in Scopus, Web of Science)

The author has presented the results of the thesis in the following conferences:

1. NCMA 2016 (Eighth Workshop on Non-Classical Models of Automata and Applications), Debrecen, Hungary, 2016.
Presentation: *Uncountable Classical and Quantum Complexity Classes.*

2. Joint Estonian-Latvian Theory Days, Lilaste, Latvia, 2016.
Presentation: *Uncountable Classical and Quantum Complexity Classes with extended results.*

3. DCFS 2017 (Descriptional Complexity of Formal Systems), Milan, Italy, 2017.
Presentation: *Uncountable Realtime Probabilistic Classes.*

4. Joint Estonian-Latvian Theory Days, Tartu, Estonia, 2017.
Presentation: *Extended results for uncountable realtime probabilistic classes.*

5. Discrete models in the theory of control systems, conference, Moscow, Russia, 2018.
Presentation: *Additional results for uncountable realtime probabilistic classes.*

6. NCMA 2018 (Tenth Workshop on Non-Classical Models of Automata and Applications), Košice, Slovakia, 2018.
Presentation: *Postselecting probabilistic finite state recognizers and verifiers.*

7. NCMA 2018 (Tenth Workshop on Non-Classical Models of Automata and Applications), Košice, Slovakia, 2018.
Presentation: *Recognition of uncountably many languages with one counter.*

8. International Workshop DLT's Satellite Workshop in Kyoto, Kyoto, Japan, 2018.
   Presentation: *Verifying every language in constant space.*

9. Joint Estonian-Latvian Theory Days, Riga, Latvia, 2018.
   Presentation: *Probabilistic verification of all languages.*

10. Student Research Forum of SOFSEM 2019 (45th International Conference on Current Trends in Theory and Practice of Computer Science), Nový Smokovec, Slovakia, 2019.
    Presentation and poster: *Private-coin verification with magic coins.*

11. SOFSEM 2016 (42nd International Conference on Current Trends in Theory and Practice of Computer Science), Harrachov, Czech Republic, 2016.
    Presentation: *Capabilities of Ultrametric Automata with One, Two, and Three States.*

12. Quantum Computing Theory in Practice, Bristol, England, 2019.
    Poster: *Two-state quantum automata with fixed cutpoint.*

During doctoral studies the author has also participated in the following publications not directly related to the thesis. 1 of these publications is indexed in Elsevier Scopus and 1 in Web of Science.

1. Maksims Dimitrijevs
   State Complexity Advantages of Ultrametric Automata.
   Proceedings of SOFSEM 2016: Current Trends in Theory and Practice of Computer Science, Volume II: Student Research Forum, pp. 13–24, 2016.
   (indexed in Scopus)

2. Maksims Dimitrijevs, Kristīne Cīpola
   Ultrametric Finite Automata and Their Capabilities.
   Baltic Journal of Modern Computing, vol. 4 (2016), No. 4, pp. 896—909.
   (indexed in Web of Science)

3. Kaspars Balodis, Maksims Dimitrijevs, Abuzer Yakaryılmaz
   Two-Way Frequency Finite Automata.
   Proceedings of Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016), books@ocg.at, vol. 321, Austrian Computer Society, 2016, pp. 75–88.

Chapter 1

Background

In the thesis we denote the left and the right end-markers by ¢ and \$, and the blank symbol by #. The input alphabet not containing symbols ¢ and \$ is denoted by Σ, $\tilde{\Sigma}$ denotes the set $\Sigma \cup \{¢, \$\}$, the work tape alphabet not containing symbols ¢ and # is denoted by Γ, $\tilde{\Gamma}$ denotes the set $\Gamma \cup \{¢, \#\}$. Σ^* denotes the set of all strings (including the empty string ε) defined over Σ. We order the elements of Σ^* lexicographically and then represent the i-th element by $\Sigma^*(i)$, where $\Sigma^*(1) = \varepsilon$. For any natural number i, $bin(i)$ denotes the unique binary representation and $(bin(i))^r$ denotes the reverse binary representation. We denote the input string by w. For any given string $w \in \Sigma^*$, $|w|$ is its length, $w[i]$ is its i-th symbol ($1 \leq i \leq |w|$), $\tilde{w} = ¢w\$$, and $lex(w)$ denotes the lexicographical number of w, such that $lex(\Sigma^*(i)) = i$ for any $i > 0$.

Turing machine is a well-known mathematical model of computation accepted as the fundamental model in theoretical computer science. In general, a deterministic Turing machine (DTM) represents a basic model of computer with infinite memory that processes input string by performing some algorithms.

In this thesis we investigate the computational power of probabilistic machines with limited memory, therefore, we begin with the formal definition of space-bounded probabilistic Turing machine (PTM).

A space-bounded PTM has two tapes — the input tape and the work tape. Each tape has a single head that operates on it, called the input head and the work head, respectively. The input tape is read-only tape and stores the input string, while the work tape is a read/write tape. Both tapes are infinite to the right. Each tape is divided into cells, and each cell contains a symbol. The left-most cell of both tapes contains the left end-marker (¢) to ensure that the head of the tape never leaves the tape, i.e., the head is not allowed to move to the left from ¢. The work tape is also not allowed to

overwrite the symbol ¢, as well as is not allowed to write ¢ on other cells of the tape. The cells on the tape can be numbered with nonnegative integers, with the zero index being assigned to the left-most cell, which contains ¢. The head of the tape is placed on one of the cells, and hence it sees exactly one of the symbols of the tape.

Formally, a space-bounded PTM M is a 7-tuple

$$M = (S, \Sigma, \Gamma, \delta, s_{init}, s_{acc}, s_{rej}),$$

where

- S is the finite set of states composed by two disjoint sets of states — the set of reading states S_r and the set of halting states $S_h = \{s_{acc}, s_{rej}\}$,
- Σ is the input alphabet,
- Γ is the alphabet of work tape,
- δ is the transition function,
- s_{init} is the initial state,
- s_{acc} is the accepting state, and,
- s_{rej} is the rejecting state.

The transition function

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

governs the behavior of M as follows: When M is in state $s \in S_r$, 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 input tape head is updated with respect to $d_1 \in \{\leftarrow, \downarrow, \rightarrow\}$ and the work tape head is updated with respect to $d_2 \in \{\leftarrow, \downarrow, \rightarrow\}$ with probability

$$\delta(s, \sigma, \gamma, s', \gamma', d_1, d_2),$$

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). To be a well-formed PTM, the following condition must be satisfied: For each triple $(s, \sigma, \gamma) \in S_r \times \tilde{\Sigma} \times \tilde{\Gamma}$,

$$\sum_{s' \in S, \gamma' \in \tilde{\Gamma}, d_1 \in \{\leftarrow, \downarrow, \rightarrow\}, d_2 \in \{\leftarrow, \downarrow, \rightarrow\}} \delta(s, \sigma, \gamma, s', \gamma', d_1, d_2) = 1.$$

The input is placed on the input tape as $\tilde{w}$, and, in the beginning of the computation, the work tape contains # on each cell except the left-most cell. The computation starts in state s_{init} with the input head placed on ¢ and the work tape head placed on the cell that contains the first #. In each step of the computation M makes a transition according to δ. The computation is terminated and the given input is accepted (rejected) if M enters s_{acc} (s_{rej}). It must be guaranteed that the input head never leaves $\tilde{w}$ during the computation.

The space used by M 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 M is called to be $O(s(n))$ space-bounded machine if it always uses $O(s(n))$ space on any input with length $n \geq 0$.

A counter is a limited type of memory. It stores some nonnegative integer value and, at each step of the computation, the value can be checked whether it is equal to zero, and the value can be updated with one of the values from the set $\{-1, 0, 1\}$, that is, the value can be increased or decreased by one or the value remains unchanged.

If we replace the work tape of a Turing machine with k counters, the model is called counter automaton with k counters. Notice that a work tape can store any number of counters.

Formally, a probabilistic counter automaton with k counters (PkCA, if $k = 1$ we denote it by PCA) M is a 6-tuple

$$M = (S, \Sigma, \delta, s_{init}, s_{acc}, s_{rej}),$$

where

- S is the finite set of states composed by two disjoint sets of states — the set of reading states S_r and the set of halting states $S_h = \{s_{acc}, s_{rej}\}$,
- Σ is the input alphabet,
- δ is the transition function,
- s_{init} is the initial state,
- s_{acc} is the accepting state, and,
- s_{rej} is the rejecting state.

The transition function

$$\delta \colon S_r \times \tilde{\Sigma} \times \{0,1\}^k \times S \times \{\leftarrow, \downarrow, \rightarrow\} \times \{-1,0,1\}^k \rightarrow [0,1]$$

governs the behavior of M as follows: When M is in state $s \in S_r$, reads symbol $\sigma \in \tilde{\Sigma}$ on the input tape, and checks the status of each of k counters $c^k = \{0,1\}^k$, whether the value of the counter is zero ($c = 0$) or not ($c = 1$), it enters state $s' \in S$, updates the input tape head with respect to $d \in \{\leftarrow, \downarrow, \rightarrow\}$ and updates the value of each counter with the value from $e^k = \{-1, 0, 1\}^k$ with probability

$$\delta(s, \sigma, c^k, s', d, e^k),$$

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) and $e = -1$ ($e = 0$ and $e = 1$) means the value of the counter is decreased by one (the value of the counter remains unchanged and the value of the counter is increased by one). To be a well-formed $PkCA$, the following condition must be satisfied: For each triple $(s, \sigma, c^k) \in S_r \times \tilde{\Sigma} \times \{0,1\}^k$,

$$\sum_{s' \in S, d \in \{\leftarrow,\downarrow,\rightarrow\}, e^k \in \{-1,0,1\}^k} \delta(s, \sigma, c^k, s', d, e^k) = 1.$$

The input is placed on the input tape as $\tilde{w}$, and, in the beginning of the computation, the value of each counter is equal to zero. At the beginning of the computation, the machine starts in state s_{init} and the input head is placed on the left end-marker ($\mathfrak{c}$). In each step of the computation, M makes its transition with respect to δ. The computation is terminated and the given input is accepted (rejected) when M enters s_{acc} (s_{rej}). It must be guaranteed that the input head never leaves $\tilde{w}$ during the computation.

The space used by M on a given input is the maximum absolute value of any of k counters during the computation with some non-zero probability. The machine M is called to be $O(s(n))$ space bounded machine if it always uses $O(s(n))$ space on any input with length $n \geq 0$.

A stack is another type of memory, which works according to the last-in first-out (LIFO) principle. The stack allows to check whether it is empty, as well as to push a new symbol into the memory and to pop out of the memory the last pushed symbol. A counter can be considered as a stack with unary alphabet, i.e., it only remembers the number of symbols stored in memory and also allows to check whether the number of symbols equals zero. Therefore, a counter is weaker type of memory than a stack.

A PTM without work tape is a probabilistic finite automaton (PFA). It is also called constant-space PTM since a PFA can use internal states as a memory.

We also consider different restrictions on the movement of input head. The models described until now can move the input head in both directions,

and so they are also called "two-way". The model is called sweeping if the direction of the input head is changed only on the end-markers, i.e., the input is read from left to right, and then right to left, and so on. The model is called "one-way" if the input head can only perform moves from $\{\downarrow, \rightarrow\}$, hence, in this case the input head cannot move to the left and the model reads the input string only once. The model is called (strict) realtime if it reads any given input as $\tilde{w} = ¢w\$$ from the left to the right and symbol by symbol without any pause on any symbol. Therefore, in case of realtime models the input head can only perform "$\rightarrow$" moves.

If M is our considered model, we denote its two-way version by $2M$ and one-way version by $1M$. For example, we denote one-way PTM by 1PTM.

Postselection is the ability to give a decision by assuming that the computation is terminated with pre-determined outcome(s) and discarding the rest of the outcomes. In [1], Aaronson introduced bounded-error postselecting quantum polynomial time and proved that it is identical to the unbounded-error probabilistic polynomial time. Later, postselecting quantum and probabilistic finite automata models have been investigated in [37, 38, 47, 49]. We denote postselecting PFA, PCA, and PTM by PostPFA, PostPCA, and PostPTM, respectively.

Restarting realtime automaton is a restricted variant of two-way finite automata. The input head of restarting realtime automaton reads the input symbol by symbol without pauses, and, when the input finishes, accepts the input, rejects it, or restarts the computation from the initial configuration. It was proved that postselecting realtime finite automata are equivalent to restarting realtime automata [47, 49]. Therefore, postselecting realtime machines can be considered as very restricted variation of two-way machines. Later, it was also shown that these two automata models are also equivalent to the realtime automata that have the ability to send a classical bit through closed timelike curves (CTCs) [33, 34]. In the thesis we do consider only postselecting realtime and restarting realtime models.

In this thesis we consider the computational power of constant-space postselecting realtime machines. Formally, a realtime PostPFA M is a 6-tuple

$$M = (S, \Sigma, \delta, s_{init}, s_{acc}, s_{rej}),$$

where

- S is the finite set of states composed by two disjoint sets of states — the set of reading states S_r and the set of halting states $S_h = \{s_{acc}, s_{rej}\}$,
- Σ is the input alphabet,

- δ is the transition function,
- s_{init} is the initial state,
- s_{acc} is the accepting state, and,
- s_{rej} is the rejecting state.

The states from S_h are called postselecting, while the other states are called non-postselecting. The transition function

$$\delta\colon S_r \times \tilde{\Sigma} \times S \to [0,1]$$

governs the behavior of M as follows: When M is in state $s \in S_r$ and reads symbol $\sigma \in \tilde{\Sigma}$, then it switches to state $s' \in S$ with probability

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

and places the input head on the next symbol to the right. To be a well-formed machine, the transition function must satisfy that for any $(s, \sigma) \in S_r \times \tilde{\Sigma}$,

$$\sum_{s' \in S} \delta(s, \sigma, s') = 1.$$

The input is placed on the input tape as $\tilde{w}$, and the computation starts in state s_{init} with input head placed on ¢. Then, the machine reads the input and behaves with respect to δ. After reading the whole input, M is in a probability distribution, which can be represented as a stochastic vector, where each entry represents the probability of being in the corresponding state.

Due to postselection, we assume that the computation ends either in s_{acc} or s_{rej}. We denote the probabilities of being in s_{acc} and s_{rej} by $a(w)$ and $r(w)$, respectively. It must be guaranteed that $a(w) + r(w) > 0$. (Otherwise, postselection cannot be done.) Then, the decision is given by normalizing these two values: w is accepted and rejected with probabilities

$$\frac{a(w)}{a(w) + r(w)} \text{ and } \frac{r(w)}{a(w) + r(w)},$$

respectively. We also note that the automaton M ends its computation in non-postselecting state(s) (if there is any) with probability $1 - a(w) - a(r)$, but the ability of making postselection discards this probability (if it is non-zero).

By making a simple modification on a realtime PostPFA, we can obtain a restarting realtime PFA [47, 49]:

- each non-postselecting state is changed to restarting state,
- postselecting accepting and rejecting states are changed to accepting and rejecting states, and then,
- if the automaton ends in a restarting state, the whole computation is started again from the initial configuration (state).

The analysis of accepting and rejecting probabilities for the input remains the same, and hence both models have the same accepting (and rejecting) probabilities on every input.

Moreover, if we have $a(w) + r(w) = 1$ for any input $w \in \Sigma^*$, then the automaton is simply a PFA since making postselection or restarting mechanism does not have any effect on the computation or decision.

Language $L \subseteq \Sigma^*$ is said to be recognized by a probabilistic machine M with error bound ϵ if

- any member is accepted by M with probability at least $1 - \epsilon$, and,
- any non-member is rejected by M with probability at least $1 - \epsilon$.

We can also say that L is recognized by M with bounded error or recognized by bounded-error machine M.

In this thesis we also consider verification power of probabilistic models. To investigate the verification power of a machine it is common to consider the interactive proof systems.

An interactive proof system (IPS) [21, 6] is composed by a prover (P) and a (probabilistic) verifier (V), denoted by pair (P, V), who can communicate with each other. The aim of the verifier is to make a decision on a given input and the aim of the prover (assumed to have unlimited computational power) is to convince the verifier to make positive decision. Therefore, the verifier should be able to verify the correctness of the information (proof) provided by the prover since the prover may be cheating when the decision should be negative.

In this thesis, we focus on memory-bounded verifiers [15] and our verifiers are space-bounded PTMs. The PTM verifier has two tapes (the read-only input tape and the read/write work tape) and a communication channel. The communication between the prover and verifier is done via a communication cell holding a single symbol. The prover can see only the given input and the symbols written on the communication cell by the verifier. The

prover may know the program of the verifier but may not know which probabilistic choices are done by the verifier. Since the outcomes of probabilistic choices are not visible to the prover, such IPS is called private-coin. If the probabilistic outcomes are not hidden (sent via the communication channel), then it is called public-coin, i.e., the prover can have complete information about the verifier during the computation. Public-coin IPS is also known as Arthur-Merlin games [6]. In the thesis we do consider only private-coin IPSs.

Formally, a space-bounded PTM verifier V is a 8-tuple

$$V = (S, \Sigma, \Gamma, \Upsilon, \delta, s_{init}, s_{acc}, s_{rej}),$$

where

- S is the finite set of states composed by three disjoint sets of states — the set of reading states S_r, the set of communicating states S_c, and the set of halting states $S_h = \{s_{acc}, s_{rej}\}$,
- Σ is the input alphabet,
- Γ is the alphabet of work tape,
- Υ is the communication alphabet,
- δ is the transition function composed by transition function from the reading states δ_r and transition function from the communicating states δ_c,
- s_{init} is the initial state,
- s_{acc} is the accepting state, and,
- s_{rej} is the rejecting state.

When in a communicating state, say $s \in S_c$, V writes symbol $\tau_s \in \Upsilon$ on the communication cell (τ_s depends only on the current internal state) and the prover writes back a symbol, say $\tau \in \Upsilon$. Then, V switches to state $s' = \delta_c(s, \tau) \in S$.

When in a reading state, V behaves as an ordinary PTM:

$$\delta_r \colon S_r \times \tilde{\Sigma} \times \tilde{\Gamma} \times S \times \tilde{\Gamma} \times \{\leftarrow, \downarrow, \rightarrow\} \times \{\leftarrow, \downarrow, \rightarrow\} \rightarrow [0, 1].$$

That is, when V is in a reading state $s \in S_r$, 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 input tape

head is updated with respect to $d_1 \in \{\leftarrow, \downarrow, \rightarrow\}$ and the work tape head is updated with respect to $d_2 \in \{\leftarrow, \downarrow, \rightarrow\}$ with probability

$$\delta_r(s, \sigma, \gamma, s', \gamma', d_1, d_2),$$

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

$$\sum_{s' \in S, \gamma' \in \tilde{\Gamma}, d_1 \in \{\leftarrow, \downarrow, \rightarrow\}, d_2 \in \{\leftarrow, \downarrow, \rightarrow\}} \delta(s, \sigma, \gamma, s', \gamma', d_1, d_2) = 1.$$

Like a regular PTM, V accepts (rejects) the input when it enters the state s_{acc} (s_{rej}).

The language $L \subseteq \Sigma^*$ is verifiable by V with error bound $\epsilon < \frac{1}{2}$ if

1. there exists an honest prover (a prover, that provides reliable information, expected by the verifier) P such that any $x \in L$ is accepted by V with probability at least $1 - \epsilon$ by communicating with P, and,

2. any $x \notin L$ is always rejected by V with probability at least $1 - \epsilon$ when communicating with any possible prover (P^*).

It is also said that there is an IPS (P, V) with error bound ϵ for language L. The first property is known as completeness and the second one is known as soundness. Generally speaking, completeness means there is a proof for a true statement and soundness means none of the proofs works for a false statement. The case when every member is accepted with probability 1 is also called as perfect completeness.

We remark that all the time and memory bounds are defined for the verifier since we are interested in the verification power of the machines with limited resources.

We also consider so-called weak IPS, which is obtained by replacing the condition 2 above by the following condition.

2′. Any $x \notin L$ is accepted by V with probability at most ϵ when communicating with any possible prover (P^*).

Therefore, in weak IPSs, the computation may not halt with high probability on non-members, or, in other words, each non-member may not be rejected with high probability.

Due to communications with the prover, the program of the verifier with the possible communications is called protocol. A private-coin protocol is called one-way if the verifier always sends the same symbol to the prover. In this case, we can assume that the prover provides a single string (possible infinite) and this string is consumed in every probabilistic branch. It is also possible that this string (certificate) is placed on a separate one-way read-only tape (certificate tape) at the beginning of the computation and the verifier can read the certificate from this tape.

We also consider interactive proof systems with two provers [8, 17]. IPS with two provers is composed by two provers (P_1, P_2) and a probabilistic verifier (V), denoted by (P_1, P_2, V). The verifier has a different communication channel with each prover and one prover does not see the communication with the other prover. In such IPSs, the verifier V has different transition function from communicating states (δ_c): When in a communicating state, say $s \in S_c$, V writes symbol $\tau_{s1} \in \Upsilon$ on the communication cell of the first prover (P_1) and $\tau_{s2} \in \Upsilon$ on the communication cell of the second prover (P_2), and the provers P_1 and P_2 write back symbols, say $\tau_1 \in \Upsilon$ and $\tau_2 \in \Upsilon$, respectively. Then, V switches to the state $s' = \delta_c(s, \tau_1, \tau_2) \in S$. Note that both τ_{s1} and τ_{s2} depend only on current internal state.

There are different models of IPS with two provers. In Multi-Prover model by [8] both provers collaborate such that both of them are honest, or both of them are cheating. In Noisy-Oracle model by [17] both provers oppose each other such that at least one of them is honest, and other may be cheating. The latter model can also be formalized as a debate system (e.g., see [13]) where the second prover is called as refuter. In this thesis we consider the IPSs with two provers working for both models equally well.

Constant-space verifiers are defined similarly to PTM verifiers. They do not have the work tape. Restrictions on the moves of the input head (sweeping, one-way, realtime, restarting realtime) apply for the verifiers in the same way as described for the recognizers

We also provide the definition of realtime PostPFA verifiers in one-way private-coin IPSs. While the input is read in realtime mode, the automaton reads the provided certificate in one-way mode, and hence it can make pauses on some symbols of the certificate.

Formally, a realtime PostPFA verifier V as a part of a one-way IPS is a 7-tuple

$$V = (S, \Sigma, \Upsilon, \delta, s_{init}, s_{acc}, s_{rej}),$$

where

- S is the finite set of states composed by two disjoint sets of states — the set of reading states S_r and the set of halting states $S_h = \{s_{acc}, s_{rej}\}$,

- Σ is the input alphabet,
- Υ is the alphabet of certificate,
- δ is the transition function,
- s_{init} is the initial state,
- s_{acc} is the accepting state, and,
- s_{rej} is the rejecting state.

The transition function

$$\delta\colon S_r \times \tilde{\Sigma} \times \Upsilon \times S \times \{0,1\} \to [0,1]$$

governs the behavior of V as follows: When V is in state $s \in S_r$, reads input symbol $\sigma \in \tilde{\Sigma}$, and reads certificate symbol $\upsilon \in \Upsilon$, it switches to state $s' \in S$, places the input head on the next symbol to the right and makes the action $d \in \{0,1\}$ on the certificate with probability

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

where the next (resp., the same) symbol of the certificate is selected for the next step if $d = 1$ (resp., $d = 0$).

To be a well-formed machine, the transition function must satisfy that for any $(s, \sigma, \upsilon) \in S_r \times \tilde{\Sigma} \times \Upsilon$,

$$\sum_{s' \in S,\ d \in \{0,1\}} \delta(s, \sigma, \upsilon, s', d) = 1.$$

Let $w \in \Sigma^*$ be the given input. For a given certificate, say $c_w \in \Upsilon^*$, V starts in state s_{init}, reads the input as $\tilde{w}$ and the certificate in realtime and one-way modes, respectively, and behaves with respect to δ. After reading the whole input, V is in a probability distribution, which can be represented as a stochastic vector, where each entry represents the probability of being in the corresponding state.

Due to postselection, we assume that the computation ends either in s_{acc} or s_{rej}. We denote the probabilities of being in s_{acc} and s_{rej} by $a(w)$ and $r(w)$, respectively. It must be guaranteed that $a(w) + r(w) > 0$. (Otherwise, postselection cannot be done.) Then, the decision is given by normalizing these two values: w is accepted and rejected with probabilities

$$\frac{a(w)}{a(w) + r(w)} \text{ and } \frac{r(w)}{a(w) + r(w)},$$

respectively.

We denote the set of integers by $\mathbb{Z}$ and the set of positive integers by $\mathbb{Z}^+$. The set $\mathcal{I} = \{I \mid I \subseteq \mathbb{Z}^+\}$ is the set of all subsets of positive integers.

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

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

Similarly, the membership of each string for language $L \subseteq \Sigma^*$ is represented as a binary probability value:

$$p_L = 0.x_1 01 x_2 01 x_3 01 \cdots x_i 01 \cdots , \qquad x_i = 1 \leftrightarrow \Sigma^*(i) \in L.$$

The coin landing on head with probability p_I (resp., p_L) is named as $\mathfrak{coin}_I$ (resp., $\mathfrak{coin}_L$).

Chapter 2

Main lemma

There are different techniques in the literature to encode a language as a single real number, which can be used as a transition value by a probabilistic or quantum machine such that, for a given input, the machine can determine its membership based on this transition value.

In [2], it was shown that polynomial-time real-valued quantum Turing machines can recognize uncountably many languages with bounded error by using the following encoding. The membership of each positive integer in any $I \in \mathcal{I}$ can be represented as a single rotation on $\mathbb{R}^2$ with the angle, originally given in [2]:

$$\theta_I = 2\pi \sum_{i=1}^{\infty} \left(\frac{x_i}{8^{i+1}} \right), \qquad \begin{array}{rrl} x_i = & 1, & \text{if } i \in I \\ x_i = & -1, & \text{if } i \notin I \end{array}.$$

To recognize uncountably many languages with bounded error probabilistically, Say and Yakaryılmaz [36] mentioned a coin which lands heads with probability $0.x$, where x is an infinite sequence of digits whose k-th member encodes whether the k-th unary string is in some given language L.

In this chapter, we propose a modified probabilistic encoding to guarantee bounded error for recognition of any language, which is also easily applicable to any error bound.

We begin with considering two tasks, and then we introduce a lemma, which helps to solve these tasks.

Suppose that we have $\texttt{coin}_I$ that lands on head with probability p_I, where $I \subseteq \mathbb{Z}^+$ is some set. The probability p_I encodes the set I. Given $k \in \mathbb{Z}$, the task is to check whether $k \in I$.

The next task is a bit more complicated. Suppose that we have $\texttt{coin}_L$ that lands on head with probability p_L. The probability p_L encodes some language $L \subseteq \Sigma^*$. Given $w \in \Sigma^*$, the task is to check whether $w \in L$.

In the case of the first task it is enough to guess the value of the bit x_k from p_I. To solve the second task, we first find $l = lex(w)$, and after that we guess the value of the bit x_l in p_L. The following lemma shows how to correctly guess the bit x_k (resp., x_l) from the $\mathfrak{coin}_I$ (resp., $\mathfrak{coin}_L$) with high probability.

Lemma 1. *Let $x = x_1x_2x_3\cdots$ be an infinite binary sequence. If a biased coin lands on head with binary probability value $p = 0.x_101x_201x_301\cdots$, then the value x_k can be determined with probability at least $\frac{3}{4}$ after 64^k coin tosses.*

Proof. Let X be the random variable denoting the number of heads after 64^k coin flips. The expected value of X is $E[X] = p\cdot 64^k$. The value of x_k is equal to $(3\cdot k+3)$-th bit from the left of the binary point in $E[X]$. If $|X-E[X]| \leq 8^k$, we still have the correct x_k since in $E[X]$ $(= x_101x_201x_301\cdots x_k01\cdots)$ x_k01 is followed by $3k$ bits, and, if we add a number in the interval $[-8^k, 8^k]$ to $E[X]$, we can get a number between

$$x_101x_201x_301\cdots x_k00\cdots \text{ and } x_101x_201x_301\cdots x_k10\cdots .$$

By using this fact with Chebyshev's inequality, we can conclude that

$$Pr[|X - E[X]| \geq 8^k] \leq \frac{p\cdot(1-p)\cdot 64^k}{(8^k)^2} = \frac{p\cdot(1-p)\cdot 64^k}{64^k} = p\cdot(1-p),$$

where the function $p\cdot(1-p)$ is parabolic and its global maximum is $\frac{1}{4}$, i.e., $p\cdot(1-p) \leq \frac{1}{4}$ for any chosen probability p.

Therefore, by returning the $(3k+3)$-th digit of the counter value that keeps the number of heads after 64^k coin tosses, we can correctly guess x_k with the probability at least $\frac{3}{4}$. □

As a straightforward method (Method 1) of application of this lemma, we can determine the value x_k correctly with probability at least $\frac{3}{4}$ after 64^k coin tosses, where we guess the value x_k as the $(3k+3)$-th digit of the binary number representing the total number of heads after the whole coin tosses. This lemma will be used in the following chapters many times.

For counter models we need some tricks, and so we use a modified method (Method 2) of application of Lemma 1. After 64^k coin tosses the number of heads is written as a binary counter and then the $(3\cdot k+3)$-th bit from the right is equal to x_k with probability at least $\frac{3}{4}$. Such counter has a length $6k+1$. On the other hand, we can use a binary counter of length $3k+3$ for the same task. Starting from 0, if $t = 2^{3k+2}$ is added to the counter, then the $(3\cdot k+3)$-th bit becomes 1. If another t is added, then the same bit becomes 0. Therefore, we can simply use a counter of length $3k+3$ that implements

a counter modulo 2^{3k+3} and its leftmost digit is the same as the $(3 \cdot k + 3)$-th bit of the longer counter. Due to its simplicity, we use this shorter counter in some of our proofs.

Now we continue with analysis of some other possible encodings that contain the information about the members of a set.

Let our biased coin land on head with binary probability value

$$p = 0.x_1a_1a_2x_2a_3a_4x_3a_5a_6 \cdots x_k a_{2k-1}a_{2k} \cdots ,$$

where for each $k > 0$: $a_{2k-1} \neq a_{2k}$. Like in the proof of Lemma 1, we can show that the value x_k can be determined with probability at least $\frac{3}{4}$ after 64^k coin tosses. Now we have to consider two cases:

- If $a_{2k-1} = 1$ and $a_{2k} = 0$, then the expected number of heads after 64^k coin tosses with probability $\frac{3}{4}$ is between

$$x_1a_1a_2x_2a_3a_4x_3a_5a_6 \cdots x_k 01 \cdots \text{ and } x_1a_1a_2x_2a_3a_4x_3a_5a_6 \cdots x_k 11 \cdots .$$

- If $a_{2k-1} = 0$ and $a_{2k} = 1$, then the expected number of heads after 64^k coin tosses with probability $\frac{3}{4}$ is between

$$x_1a_1a_2x_2a_3a_4x_3a_5a_6 \cdots x_k 00 \cdots \text{ and } x_1a_1a_2x_2a_3a_4x_3a_5a_6 \cdots x_k 10 \cdots .$$

Now we consider encoding that does not have any padding bits. Note that this case does not cover any infinite binary sequence. Let our biased coin land on head with binary probability value $p = 0.x_1x_2x_3 \cdots$, and for some $k > 0$: $x_{k+1} \neq x_{k+2}$. Then the value x_k can be determined with probability at least $\frac{3}{4}$ after $4^k \cdot 16$ coin tosses. We can follow the previous analysis and can show that

$$Pr[|X - E[X]| \geq 2^k \cdot 4] \leq \frac{p \cdot (1-p) \cdot 4^k \cdot 16}{(2^k \cdot 4)^2} = \frac{p \cdot (1-p) \cdot 4^k \cdot 16}{4^k \cdot 16} = p \cdot (1-p),$$

which is sufficient to prove the statement. Now we have shown the encoding that decreases the sufficient number of coin tosses. It is also possible to compute the value x_k with bounded error if for some $l \geq k$, $x_{l+1} \neq x_{l+2}$. In this case it is enough to perform $4^l \cdot 16$ coin tosses.

If we remove the last considered constraint, we can have arbitrary binary sequence, but the approach to have arbitrary binary probability value p and guessing any bit of p with bounded error seems to fail with mentioned approach. The reason lies in the nature of real numbers. For any $k > 0$, if $p_1 = 0.x_1x_2x_3 \cdots x_k 100 \cdots 0 \cdots$ and $p_2 = 0.x_1x_2x_3 \cdots x_k 011 \cdots 1 \cdots$, then

$p_1 = p_2$, therefore, for any $l > k$, the value x_l is guessed as 1 with equal probabilities for p_1 and p_2 because p_1 and p_2 give equal probabilities of the event. On the other hand, if for some binary probability value $p = 0.x_1x_2x_3\cdots$ for some k for any $l \geq k$ $x_{l+1} = x_{l+2}$, then the encoded language is regular (because the language has finite number of members or finite number of non-members) and can be recognized by a deterministic finite state automaton. Binary probability value representing any nonregular language has the last considered constraint, i.e., for any $k > 0$ exists $l \geq k$ such that $x_{l+1} \neq x_{l+2}$. In any case, the encoding presented in the Lemma 1 is sufficient to show the power of probabilistic machines. All mentioned approaches still assume exponential number of coin tosses.

We continue with the improvement of the error bound when computing the bit x_k.

Lemma 2. *Let $x = x_1x_2x_3\cdots$ be an infinite binary sequence. If a biased coin lands on head with binary probability value $p = 0.x_101x_201x_301\cdots$, then the value x_k can be determined with probability at least $1 - \frac{1}{4\cdot 2^m}$ after $64^k \cdot 2^m$ coin tosses.*

Proof. The analysis is similar to the one in the proof of Lemma 1, therefore, we can show that

$$Pr[|X - E[X]| \geq 8^k \cdot 2^m] \leq \frac{p\cdot(1-p)\cdot 64^k\cdot 2^m}{(8^k\cdot 2^m)^2} = \frac{p\cdot(1-p)\cdot 64^k\cdot 2^m}{64^k\cdot 2^{2m}}$$

$$= \frac{p\cdot(1-p)}{2^m},$$

therefore, the probability to incorrectly compute the value x_k does not exceed $\frac{1}{4\cdot 2^m}$. □

We conclude that by increasing the number of coin tosses by a constant factor, we can obtain arbitrary small error bound.

Chapter 3

The class of all languages

In this chapter, we investigate the recognition and verification of any language with bounded error. Condon and Lipton [11] proved that for any recursively enumerable language L, there is an IPS with a bounded-error 2PFA verifier, in which each non-member may not be rejected with high probability. Feige and Shamir have shown that a 1PFA verifier can simulate the work tape of a Turing machine reliably with high probability by interacting with two provers [16]. Therefore, in such IPS with a 1PFA verifier any recursive language can be verified with bounded error. Up to our knowledge the recognition of any language with bounded-error space-bounded PTMs has not been examined. We show how much space is enough for a PTM to recognize any unary and binary language with bounded error, and we also present results for one-way counter automata. We also present results about the verification of any language with bounded error.

We start with straightforward application of Lemma 1. We first consider the recognition of unary languages.

Theorem 1. *Any unary language is recognizable by a linear-space PTMs with any error bound.*

Proof. Let $\Sigma = \{a\}$ be our alphabet. For any unary language $L \subseteq \Sigma^*$, we design a PTM for L, say M_L, that uses $\mathtt{coin}_L$.

Let $w = a^n$ be the given input for $n \geq 0$. PTM M_L implements Method 1 of application of Lemma 1 in a straightforward way and gives its decision accordingly, which will be correct with probability not less than $\frac{3}{4}$. The machine only uses linear-size binary counters to implement 64^k coin tosses and to count the number of heads (for unary L and $\mathtt{coin}_L$, $k = n + 1$). By repeating the same procedure, the success probability is increased arbitrarily close to 1. □

Now, we continue with the languages with arbitrary alphabet.

Remark 1. *Let $L \subseteq \Sigma^*$ be a k-ary language for $k > 1$, where $\Sigma = \{a_1, \ldots, a_k\}$. For any given k-ary string $w \in \Sigma^*$, let x_l represent its membership bit in p_L. Then, by using the exactly the same algorithm given in the above proof, we can determine x_l correctly with high probability. However, l is exponential in $|w|$, and hence the PTM uses exponential space.*

Corollary 1. *Any k-ary ($k > 1$) language is recognizable by a exponential-space PTMs with any error bound.*

We can conclude that when we use a straightforward application of Lemma 1, we can show how many computational resources is enough for a PTM to recognize any language with bounded error:

- Any unary language can be recognized by $O(n)$-space bounded-error PTM in exponential time.
- Any k-ary language (for any $k > 1$) can be recognized by $O(k^n)$-space bounded-error PTM in double exponential time.

Now we reduce the space bounds by allowing the PTM to interact with a single prover. For this purpose, we use the probabilistic fingerprint method: For comparing two l-bit numbers, say m_1 and m_2, we can randomly pick a $(c \log l)$-bit prime number p for some positive integer c and compare $m_1' = m_1 \mod p$ with $m_2' = m_2 \mod p$. Being verifiable from the fact given below, this reduction works with high probability.

Fact 1 (cf. [19])**.** *Let $R_1(m)$ be the number of primes not exceeding $2^{\lceil log_2 m \rceil}$, $R_2(l, t', t'')$ be the number of primes not exceeding $2^{\lceil log_2 l \rceil}$ and dividing $|t' - t''|$, and $R_3(l, m)$ be the maximum of $R_2(l, t', t'')$ over all $t' < 2^m$, $t'' \leq 2^m$, $t' \neq t''$. Then, for any $\epsilon > 0$, there is a natural number c such that $\lim_{m \to \infty} \frac{R_3(cm,m)}{R_1(cm)} < \epsilon$.*

Theorem 2. *Any unary language $L \subseteq \{a\}^*$ is verifiable by a logarithmic-space PTM with any error bound.*

Proof. There is two-way communication between the prover and the verifier. Let $w = a^n$ be the given input for $n > 0$. (The decision on the empty string is given deterministically.) Remember that the membership bit of a^n is x_{n+1} in p_L. Let $k = n+1 \geq 2$. We pick a value of c (see Fact 1) satisfying the error bound ϵ_1, a value of l (see Lemma 2) satisfying the error bound $\epsilon_2 = \frac{1}{4 \cdot 2^l}$, and a value of $d > 6$, which also determines the error bound.

The protocol has three phases. In the first phase, there is no communication. The verifier picks two random $(c \cdot (4 \cdot \log(k + l + \lceil \log d \rceil)))$-bit

prime numbers, say p_1 and p_2, and then, it calculates and stores $r_1 = 64^k \cdot 2^l \mod p_1$ in binary on the work tape. The verifier also prepares two binary counters c_1 and c_2 for storing the total number of coin tosses modulo p_1 and the total number of heads modulo p_2, respectively, and one "halting" counter $c_h = 0$.

In the second phase, the verifier asks from the prover to send $a^{64^k \cdot 2^l} b$. Once the verifier receives symbol b, the communication is ended in this phase. Therefore, we assume that the prover sends either the finite string $y = a^m b$ for some $m \geq 0$ or an infinite sequence of a's.

For each a received from the prover, the verifier reads the whole input and adds one to c_h with probability $\left(\frac{1}{64}\right)^k \cdot \left(\frac{1}{2}\right)^l$. We call it a halting walk. If $c_h = d$, the verifier terminates the computation and rejects the input. If the computation is not terminated, the verifier tosses $\mathfrak{coin}_L$, sends the result to the prover, and increases c_1 by 1. If the result is heads, the verifier also increases c_2 by 1. When the second part is ended, the verifier checks whether previously calculated r_1 is equal to $r_1' = m \mod p_1$, stored on c_1. If they are not equal, then the input is rejected. In other words, if the prover does not send $64^k \cdot 2^l$ a's, then the verifier detects this with probability at least $1 - \epsilon_1$.

Let r_2 be the binary value stored on c_2 and t be the total number of heads obtained in the second phase. In the third phase, the verifier asks from the prover to send $(bin(t))^r$ (the least significant bits are first). By using the input head, the verifier easily reads the $(3k + 3 + l)$-th bit of $bin(t)$, say x_k', and also checks whether the length of $bin(t)$ does not exceed $(6k + 1 + l)$. Meanwhile, the verifier also calculates $r_2' = t \mod p_2$. If the length of $bin(t)$ is greater than $6k + 1 + l$ or $r_2 \neq r_2'$, then the input is rejected. In other words, if the prover does not send $bin(t)$, then the verifier can catch it with probability at least $1 - \epsilon_1$. At the end of third phase, the verifier accepts the input if x_k' is 1, and rejects it if x_k' is 0.

According to Chebyshev's inequality, the value of c_h reaches d after more than $2 \cdot d \cdot 64^k \cdot 2^l$ a's with probability

$$Pr[|X - E[X]| \geq d + 1] \leq \frac{(\frac{1}{64^k \cdot 2^l}) \cdot (1 - \frac{1}{64^k \cdot 2^l}) \cdot 2 \cdot d \cdot 64^k \cdot 2^l}{(d+1)^2} < \frac{2}{d},$$

where $E[X]$ is expected value of c_h. This bound is important, since, for $m > 2 \cdot d \cdot 64^k \cdot 2^l$, Fact 1 cannot guarantee the error bound ϵ_1. (Remember that prime numbers p_1 and p_2 do not exceed

$$2^{c \cdot (4 \cdot \log(k+l+\lceil \log d \rceil))} \geq 2^{c \cdot (\log(6 \cdot k + l + 1 + \lceil \log d \rceil))}$$

for $k \geq 2$, $l \geq 0$, and $d > 6$.)

If w is not a member, then the accepting probability can be at most $\max(\epsilon_1 + \frac{2}{d}, \epsilon_2)$.

- If $m \neq 64^k \cdot 2^l$, then the input is rejected with probability at least $1 - \epsilon_1 - \frac{2}{d}$, and hence the accepting probability cannot be greater than $\epsilon_1 + \frac{2}{d}$.
- Assume that $m = 64^k \cdot 2^l$. If the prover does not send $(bin(t))^r$, then the input is rejected with probability at least $1 - \epsilon_1$, and hence the accepting probability cannot be greater than ϵ_1.
- Assume that $m = 64^k \cdot 2^l$ and the verifier sends $(bin(t))^r$, then the input is accepted with probability at most ϵ_2.

The expected running time for the non-members is exponential in n due to the halting walks.

If w is a member, then the honest prover sends all information correctly, and the verifier guesses x_{n+1} correctly with probability at least $1 - \epsilon_2$ if the computation is not terminated by halting walks. The probability of halting the computation (and rejecting the input) in the second phase is

$$Pr[|X - E[X]| \geq d - 1] \leq \frac{(\frac{1}{64^k \cdot 2^l}) \cdot (1 - \frac{1}{64^k \cdot 2^l}) \cdot 64^k \cdot 2^l}{(d-1)^2} < \frac{1}{(d-1)^2} < \frac{1}{d}.$$

Therefore, the verifier accepts w with probability at least $1 - \epsilon_2 - \frac{1}{d}$. The expected running time for members is also exponential in n.

The verifier uses $O(\log n)$ space and the success probability can be arbitrarily close to 1 by picking suitable c, l and d. □

Due to Remark 1, we can obtain the same result also for k-ary languages with exponential increase in time and space.

Corollary 2. *Any k-ary ($k > 1$) language $L \subseteq \{a_1, \ldots, a_k\}^*$ is verifiable by a linear-space PTM with any error bound.*

Now we consider multi-counter models and we start with four counters.

Theorem 3. *Any k-ary ($k \geq 1$) language L is recognizable by a 1P4CA M_L with any error bound.*

Proof. Let $\Sigma = \{a_1, \ldots, a_k\}$ be the input alphabet with k symbols, and for each $1 \leq i \leq k$, $lex(a_i) = i + 1$. Let $w = w[1]w[2] \cdots w[n-1]w[n]$ ($|w| = n$)

be the given input string. If $n = 0$, then $lex(w) = 1$. If $n > 0$, then $lex(w)$ is calculated as follows:

$$lex(w) = 1 + \sum_{i=1}^{n} k^{i-1} + \sum_{i=1}^{n} (lex(w[i]) - 2) \cdot k^{n-i}$$

since there are total $\sum_{i=1}^{n} k^{i-1}$ strings with length less than n, and there are total $\sum_{i=1}^{n} (lex(w[i]) - 2) \cdot k^{n-i}$ strings with length n that are coming before w in the lexicographic order.

If w is the empty string, then the decision is given deterministically. In the following part, we assume that $n > 0$. Let c_j $(1 \leq j \leq 4)$ represent the value of j-th counter. At the beginning of computation $c_1 = c_2 = c_3 = c_4 = 0$. Firstly, M_L reads w and sets $c_1 = lex(w)$ as follows. M_L reads $w[1]$ and sets $c_1 = lex(w[1])$. After that, for each $m \in \{2, \ldots, n\}$, M_L reads $w[m]$ and multiplies the value c_1 by k and increases it by $(2-k)+(lex(w[m])-2)$ with the help of other counters.

We claim that after reading w, $c_1 = lex(w)$. We prove this claim by induction on m. The basis, when $m = 1$, is trivial, since $c_1 = lex(w[m])$.

Suppose that for some $m > 0$:

$$c_1 = lex(w[1]w[2] \cdots w[m]) = 1 + \sum_{i=1}^{m} k^{i-1} + \sum_{i=1}^{m} (lex(w[i]) - 2) \cdot k^{m-i}.$$

Then M_L reads $w[m+1]$, and c_1 is updated to

$$\begin{aligned}
c_1 &= (1 + \sum_{i=1}^{m} k^{i-1} + \sum_{i=1}^{m} (lex(w[i]) - 2) \cdot k^{m-i}) \cdot k + (2-k) + (lex(w[m+1]) - 2) \\
&= k + \sum_{i=1}^{m} k^{i} + \sum_{i=1}^{m} (lex(w[i]) - 2) \cdot k^{m+1-i} + (2-k) + (lex(w[m+1]) - 2) \\
&= 2 + \sum_{i=2}^{m+1} k^{i-1} + \sum_{i=1}^{m} (lex(w[i]) - 2) \cdot k^{m+1-i} + (lex(w[m+1]) - 2) \\
&= 1 + \sum_{i=1}^{m+1} k^{i-1} + \sum_{i=1}^{m+1} (lex(w[i]) - 2) \cdot k^{m+1-i}.
\end{aligned}$$

Thus, $c_1 = lex(w[1] \cdots w[m]w[m+1])$, and hence the claim is proven.

After reading w, M_L stays on the right end-marker and does the rest of the computation without moving the input head. Let $l = c_1 = lex(w)$. M_L decreases c_1 by 1, and sets $c_2 = 64$ and $c_3 = 4 \cdot 8$. Then, until c_1 hits zero, M_L decreases c_1 by 1, and then multiplies c_2 by 64 and c_3 by 8 with the help of 4th counter. When $c_1 = 0$, we have $c_2 = 64^l$, $c_3 = 4 \cdot 8^l$, and $c_4 = 0$.

After that, M_L tosses $\mathfrak{coin}_L$ 64^l times and guesses the value of x_l in p_L by using the number of total heads. Remember that $x_l = 1$ if and only if $w \in L$. Let h be the total number of heads in binary. Due to Lemma 1, the $(3l+3)$-th bit of h is x_l with probability at least $\frac{3}{4}$. Notice that when counting the total number of heads, the $(3l+3)$-th bit of h is flipped after each

$$t = 2^{3l+2} = 4 \cdot 8^l$$

number of heads. Thus, by switching the values of the 3rd and the 4th counters, M_L determines each block of t heads. By using its internal states, M_L keeps the candidate value for x_l, say x'_l. M_L sets $x'_l = 0$ before the coin tosses and updates it to $1 - x'_l$ after each t number of heads.

At the end of the coin tosses, if $x'_l = 0$ (resp., $x'_l = 1$), the input is rejected (resp., accepted). The decision will be correct with probability at least $\frac{3}{4}$.

During coin tosses, M_L can set $c_1 = 64^l$ while setting $c_2 = 0$. After coin tosses, $c_3 + c_4 = 4 \cdot 8^l$. Therefore, after coin tosses, M_L can set $c_1 = 0$, $c_2 = 64^l$, $c_3 = 4 \cdot 8^l$, and $c_4 = 0$, and repeat the procedure of coin tosses. By repeating the procedure, M_L can increase the success probability arbitrarily close to 1 by picking the most frequently calculated value x'_l. □

It is a well-known fact that two counters can simulate any number of counters with a huge slowdown [28]. The values of k counters, say $c_1, c_2, \ldots, c_k$, can be stored on a single 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 distinct prime numbers. Then, by the help of the second counter and the internal states, the status of each simulated counter can be easily detected and stored, and then all updates on the simulated counters are applied one by one.

We can apply this simulation technique to the algorithm given in the proof of Theorem 3, and then, we obtain a 1P2CA for any given language.

Corollary 3. *Any k-ary ($k \geq 1$) language L is recognizable by a 1P2CA with any error bound.*

It is known that bounded-error unary 1PFAs with a single stack cannot recognize any nonregular language [24]. Therefore, at least two counters are needed.

Now we consider protocol for IPS with a single prover and constant-space verifier. Condon and Lipton [11] proved that for any recursively enumerable language L, there is an IPS with a bounded-error 2PFA verifier, in which each non-member may not be rejected with high probability. It is a well-known

fact that two-way deterministic finite automaton with 2 counters (2D2CA) can recognize any recursively enumerable language, and in [11] it was shown how 2PFA verifier can verify with bounded error the computation of 2D2CA provided by the prover for the input string.

We extend the results of Condon and Lipton by considering the verification of all languages in such IPSs.

Theorem 4. *There is a weak-IPS (P, V) for any language L where V is a sweeping PFA with any error bound.*

Proof. For any language L, V simulates the algorithm of 1P4CA M_L (which recognizes L with probability at least $1 - \epsilon$ for any $\epsilon > 0$) described in the proof of Theorem 3 on the given input w and asks the prover to store the contents of four counters.

For each step of M_L, V interacts with the prover. First, V asks from the prover the values of four counters as

$$a^{t_1} b^{t_2} c^{t_3} d^{t_4} z,$$

where t_j is the value of j-th counter ($1 \leq j \leq 4$). Second, V implements the transition of M_L based on the current state, the symbol under the input head, the probabilistic outcome, and the status of the counters. Then, V updates the state and head position by itself and sends the updates on the values of the counters to the prover as

$$f_1 f_2 f_3 f_4.$$

Here $f_j \in \{-1, 0, 1\}$ and it means that the verifier asks from the prover to add f_j to the value of the j-th counter, where $1 \leq j \leq 4$.

Without loss of generality, we pick an arbitrary computation path of M_L. For this path, let

$$c_V = f_{1,1} f_{1,2} f_{1,3} f_{1,4} \; f_{2,1} f_{2,2} f_{2,3} f_{2,4} \; \cdots \; f_{i,1} f_{i,2} f_{i,3} f_{i,4} \; \cdots$$

be the string representing the messages sent by the verifier and let

$$c_P = a^{t_{1,1}} b^{t_{1,2}} c^{t_{1,3}} d^{t_{1,4}} \; z \; a^{t_{2,1}} b^{t_{2,2}} c^{t_{2,3}} d^{t_{2,4}} \; z \; \cdots \; z \; a^{t_{i,1}} b^{t_{i,2}} c^{t_{i,3}} d^{t_{i,4}} \; z \; \cdots$$

be the string representing the messages sent by the prover. The verifier V can check the validity of c_P as described below. For each i, V can compare $t_{i,1}$, $t_{i,2}$, $t_{i,3}$, and $t_{i,4}$ with $t_{i+1,1}$, $t_{i+1,2}$, $t_{i+1,3}$, and $t_{i+1,4}$, respectively, by using the values $f_{i,1} f_{i,2} f_{i,3} f_{i,4}$.

Let $y < \frac{1}{2}$ be the parameter to determine the error bound in the following checks. For the validity check of c_P, V makes four comparisons in parallel such that one comparison is responsible for one counter. During the j-th comparison ($1 \leq j \leq 4$), V creates two paths with equal probabilities. In the first path, V says "A" with probability, denoted by $Pr[A_j]$,

$$Pr[A_j] = \prod_{i=1}^{g-1} y^{2t_{i,j}+2(t_{i+1,j}-f_{i,j})},$$

in the second path, it says "R" with probability, denoted by $Pr[R_j]$,

$$Pr[R_j] = \prod_{i=1}^{g-1} \frac{y^{4t_{i,j}} + y^{4(t_{i+1,j}-f_{i,j})}}{2},$$

where g is the total number of computational steps of M_L. Here each comparison executes two parallel procedures such that the first procedure produces the probabilities for odd i's and the second procedure produces the probabilities for even i's.

Once the simulation of M_L is finished, V says only "A"s in all comparisons with probability

$$Pr[A] = Pr[A_1] \cdot Pr[A_2] \cdot Pr[A_3] \cdot Pr[A_4]$$

and V says only "R"s in all comparisons with probability

$$Pr[R] = Pr[R_1] \cdot Pr[R_2] \cdot Pr[R_3] \cdot Pr[R_4].$$

It is easy to see that if $t_{i,j} = (t_{i+1,j} - f_{i,j})$ for each i and j, then

$$Pr[A] = Pr[R] = \prod_{i=1}^{g-1} y^{4(t_{i,1}+t_{i,2}+t_{i,3}+t_{i,4})}.$$

On the other hand, if $t_{i,j} \neq (t_{i+1,j} - f_{i,j})$ for some i and j, then

$$\frac{Pr[R]}{Pr[A]} \geq \frac{y^{2t_{i,j}-2(t_{i+1,j}-f_{i,j})}}{2} + \frac{y^{2(t_{i+1,j}-f_{i,j})-2t_{i,j}}}{2} > \frac{1}{2y^2}$$

since either $(2t_{i,j} - 2(t_{i+1,j} - f_{i,j}))$ or $(2(t_{i+1,j} - f_{i,j}) - 2t_{i,j})$ is a negative even integer.

If c_P is finite (a cheating prover may provide an infinite c_P), then V gives a positive decision for c_P with probability $Pr[A]$ and negative decision for c_P

with probability $(y \cdot Pr[R])$. Therefore, if c_P is valid, then the probability of positive decision is $\frac{1}{y}$ times of the probability of negative decision. If c_P is not valid, then the probability of negative decision is at least $\frac{1}{2y}$ times of the probability of positive decision.

When the computation is finished, if V does not give a decision on c_P, V moves the input head on the left end-marker and restarts the process of computation and interaction. If V gives a negative decision on c_P, the input is rejected. If V gives a positive decision on c_P, the input is accepted if x_l is computed as 1, and the input is rejected if x_l is computed as 0.

If the prover is honest, then $Pr[A] = Pr[R]$. If $w \in L$, the probability to accept the input is at least $Pr[A] \cdot (1 - \epsilon)$ and the probability to reject the input is at most $Pr[A] \cdot \epsilon + y \cdot Pr[R]$. Therefore, the total probability to accept the input is at least

$$\frac{Pr[A] \cdot (1 - \epsilon)}{Pr[A] \cdot (1 - \epsilon) + Pr[A] \cdot \epsilon + y \cdot Pr[A]} = \frac{1 - \epsilon}{1 + y},$$

which can be arbitrarily close to $1 - \epsilon$ by picking sufficiently small value of y. If $w \notin L$ and the prover is honest, then the probability to accept the input is at most $Pr[A] \cdot \epsilon$ and the probability to reject the input is at least $Pr[A] \cdot (1-\epsilon) + y \cdot Pr[R]$. Therefore, the total probability to reject the input is at least

$$\frac{Pr[R] \cdot (1 - \epsilon) + y \cdot Pr[R]}{Pr[R] \cdot (1 - \epsilon) + y \cdot Pr[R] + Pr[R] \cdot \epsilon} = \frac{1 - \epsilon + y}{1 + y} > 1 - \epsilon.$$

If the prover is cheating and provides a finite c_P, the probability to reject the input is at least $\frac{1}{2y}$ times of the probability to accept the input. Therefore, the total probability to reject the input is at least

$$\frac{\frac{1}{2y}}{1 + \frac{1}{2y}} = \frac{1}{2y + 1},$$

which can be arbitrarily close to 1 by picking sufficiently small value of y.

By picking sufficient small values of $\epsilon > 0$ and $y < \frac{1}{2}$ the probability of correct decision can be arbitrarily close to 1. A cheating prover may provide an infinite c_P, in which case V does not give any decision. □

When constant-space verifier interacts with two provers, it can verify any language while also rejecting the non-members with high probability. It is known that a 1PFA verifier can simulate the work tape of a Turing machine reliably with high probability by interacting with two provers [16]. Since we

use this fact, we present its explicit proof here.

Fact 2 (cf. [16]). *A 1PFA verifier V can simulate the work tape of a Turing machine reliably with high probability by interacting with two provers P_1 and P_2.*

Proof. Let m denote the contents of the infinite to the right work tape including the position of the head, where for each $i > 0$, m_i denotes i-th symbol from the left on the work tape. To store the position of head one of m_i's is marked. We accomplish this by doubling the alphabet of the work tape, hence one symbol can simultaneously store the value and the marker of presence of the head.

At the beginning of the computation, V secretly picks random values a, b, and r_0, where each of them is between 0 and $q-1$ and q is a predetermined prime number greater than the alphabet of the work tape. The verifier V interacts with provers P_1 and P_2 and asks them to store m in the following way: For each odd (resp., even) i, V asks P_1 (resp., P_2) to store m_i, r_i, sig_i, where $0 \leq r_i \leq q-1$ is picked by V randomly, and

$$sig_i = (m_i \cdot a + r_i \cdot b + r_{i-1}) \mod q$$

is a signature. Therefore, each prover stores a sequence of triples (m_i, r_i, sig_i) for i in ascending order.

To read the contents from the work tape, V scans m from the left to the right, i.e., for each i, V requests from the provers to send the triples (m_i, r_i, sig_i). While V scans the input, it checks the correctness of signatures. If V finds a defect, it rejects the input and also identifies the prover giving the incorrect signature as a cheater.

In order to update the content of the work tape, V picks new random r_0, scans the input, for each triple (m_i, r_i, sig_i) generates new r_i, recalculates sig_i, and asks the provers to replace the values. If m_i is changed for some i, then the values of triple are changed accordingly. Note that one update may include the change of one m_i and the change of the position of work head by one cell, in which case at most 2 sequential m_i's are updated. In this case, V asks the provers to update the contents of the corresponding triples.

The provers cannot learn the values of a and b from the information provided by V, since, for each $i > 0$, V sends m_i that is derived from the computation, random value r_i, and value sig_i, which has one-to-one correspondence with r_{i-1}, which was also chosen randomly. Note that r_0 is also randomly chosen and kept in secret.

Let (m_i, r_i, sig_i) be the triple provided by V to one of the provers, say P, for some i, and (m'_i, r'_i, sig'_i) be the values provided by P, where at least one of

the values is different from the one sent by V. Assume that sig_i' corresponds to m_i' and r_i' correctly. Therefore,

$$sig_i' = (m_i' \cdot a + r_i' \cdot b + r_{i-1}) \mod q.$$

Then, we have the following relation between a and b:

$$(sig_i' - sig_i) = ((m_i' - m_i) \cdot a + (r_i' - r_i) \cdot b + r_{i-1}) \mod q.$$

Since q is a prime number, exactly q pairs of (a, b)'s satisfy this equation, and there are total q^2 different pairs of (a, b)'s. Thus, since P does not know the values of a and b, the probability that P can provide valid values is $\frac{1}{q}$.

If both provers are honest, then they send the correct stored values to V. This implies that all signature checks will be passed successfully and V accepts the interactions. If a prover changes at least one symbol of the contents that V entrusted to store, the signature check will fail for the triple of changed symbol with probability at least $\frac{q-1}{q}$, in which case V rejects the interaction and identifies the cheating prover. Note that the described protocol works equally well in both IPS models with two provers because any case of cheating is recognized individually. □

Now we can present our result about the verification of every language. We close the chapter with constant-space verification of all languages.

Theorem 5. *There is an IPS with two provers (P_1, P_2, V) for any language where V is a 1PFA with any error bound.*

Proof. A 2PFA verifier, say V', can execute the algorithm given for the Corollary 1 by interacting with two provers as described in the proof of Fact 2 such that the provers reliably store the contents of the counters for coin tosses and processing the number of heads. If the content of the work tape is changed by a prover, V' can catch this with probability at least $\frac{q-1}{q}$, and hence rejects the input with the same probability.

Let $w \in \Sigma^*$ be the given input and let x_l represent its membership bit in p_L. If both provers are honest, then V' correctly performs 64^l tosses of $\mathtt{coin}_L$ and then processes the number of heads. If the bit x_l ($(3l+3)$-th bit in the number of heads) is guessed as 1, the input is accepted, and, if x_l is guessed as 0, then the input is rejected. Therefore, the input is verified correctly with probability at least $\frac{3}{4}$. If at least one of the provers is not honest, then any change of the contents stored by the prover is detected with probability at least $\frac{q-1}{q}$, and the input is rejected. This is true if either one or both provers are cheating.

We can modify V' and obtain a 1PFA verifier V. At the beginning of the computation, V reads the input from left to right and writes it on the work tape (asks the provers to store it). Then, V implements the rest of the protocol by staying on the right end-marker.

After writing the input on work tape, V can execute the algorithm of calculation of value x_l multiple times and choose the most frequent outcome of the algorithm as the decision for recognition, thus, increase the probability of correct decision arbitrarily close to 1. □

Chapter 4

Uncountable classes of languages

In this chapter, we consider different bounded-error probabilistic models that can define uncountably many languages. The sets of languages in this chapter are proper subsets of the class of all languages.

In [36], Say and Yakaryılmaz showed that bounded-error logarithmic-space unary PTMs can recognize uncountably many languages. In [3], Alt and Mehlhorn showed that DTMs can recognize nonregular unary languages in $O(\log \log n)$ space. In [18], Freivalds has showed that nonregular language LOG can be recognized by 2PFA with bounded error, and then concluded that, if a binary language L is recognized by a bounded-error PTM in $s(n)$ space, then the binary language LOG(L) is recognized by a bounded-error PTM in $\log(s(n))$ space. In [46], it was shown how to recognize the nonregular language EQUAL with bounded-error PostPFA. We modify the proofs to recognize the language LOG with bounded-error PostPFA and prove the result of Freivalds for PostPTMs. As a consequence, we show that PostPTMs and PostPCAs with arbitrarily small non-constant space complexity can recognize uncountably many languages with bounded error. We also present that bounded-error unary PostPFAs can verify uncountably many languages.

In [50], it was shown that realtime DTMs can recognize unary nonregular languages in $O(\log n)$ space. In [19], Freivalds proved that 1PTMs can recognize nonregular languages in $O(\log \log n)$ space. We extend these results and show that bounded-error realtime PTMs can recognize uncontably many unary and binary languages in $O(\log n)$ and $O(\log \log n)$ space, respectively. We also extend our results to counter automata. We use the well-known simulation technique of any number of counters by 2 counters (see [28]), and also implement some techniques to lower the amount of space used on counters by using additional counters or internal states.

4.1 Two-way probabilistic Turing machines

4.1.1 Recognizers

We begin this section with a basic example. The set $\mathcal{I} = \{I \mid I \subseteq \mathbb{Z}^+\}$ is the set of all subsets of positive integers, and hence it is an uncountable set like the set of real numbers ($\mathbb{R}$). The cardinality of $\mathbb{Z}$ or $\mathbb{Z}^+$ is $\aleph_0$ (countably many). It is easy to see that there exist uncountably many unary languages. As we have shown in the previous chapter, bounded-error $O(n)$-space PTM can recognize any unary language.

For each $I \subseteq \mathbb{Z}^+$, we can define the following language: $L_I = \{a^{64^k} \mid k \in I\}$. There are uncountably many different languages L_I since there is a bijection (one-to-one and onto) between $I \in \mathcal{I}$ and L_I, and, $\mathcal{I}$ is an uncountable set. A PTM, say M, can recognize the language L_I in a straightforward way. First, the machine deterministically checks whether the input w has length 64^k for some $k > 0$. For this purpose, it reads w once, counts the number of a's, and stores the result on the work tape in binary. The machine expects the result on the work tape to be written as

$$1(000000)^k$$

for some $k > 0$. If not, then the input is rejected. If it is, then we know that $w = a^{64^k}$ for some $k > 0$. After that, M performs 64^k coin tosses and implements the algorithm explained in Method 1 of application of Lemma 1 in a straightforward way. If x_k is guessed as 1, the input is accepted, and the input is rejected otherwise. Note that M uses $O(\log n)$ space and reads the input once. Therefore, bounded-error one-way $O(\log n)$-space PTMs can recognize uncountably many languages.

We have shown that bounded-error logarithmic-space PTMs can recognize uncountably many languages in a straightforward way. We can ask whether the same result can be obtained for smaller space bounds. We continue this section with the models that have smaller amount of computational resources. Note that we consider uncountably many unary and binary languages separately since the behavior of machines on unary and binary languages can be different.

Now, we show that $O(\log \log n)$ space is enough to recognize uncountably many unary languages in $O(n \log n)$ time.

Theorem 6. *Unary sweeping PTMs can recognize uncountably many languages in $O(\log \log n)$ space and $O(n \log n)$ time steps with any error bound.*

Proof. For our purpose, we define languages based on the following unary

language given by Alt and Mehlhorn in 1975 [3]:

$$\texttt{AM75} = \{a^n \mid n > 0 \text{ and } F(n) \text{ is a power of } 2\},$$

where $F(n) = min\{i \mid i \text{ does not divide } n\} \in \{2, 3, 4, \ldots\}$. It is known that $O(\log \log n)$-space DTMs can recognize $\texttt{AM75}$. It is clear that the following language

$$\texttt{AM75}' = \{a^n \mid n > 0 \text{ and } F(n) \text{ is a power of } 64\}$$

can also be recognized by $O(\log \log n)$-space DTMs. For the sake of completeness, we provide the details of the algorithm (see also [41]).

Assume that the input is $w = a^n$ for some $n > 0$. In order to check if a number k written in binary on the work tape divides n, we can use $O(\log k)$ space for binary values of k that form a counter, and then we can check whether $n \mod k$ is equal to zero or not. In order to compute $F(n)$, we can check each $k = 2, 3, 4, \ldots$ by reading w in sweeping mode in order to determine the first k such that $n \mod k \neq 0$. It is known that $F(n) < c \cdot \log n$ for some constant c (see Lemma 4.1.2(d) in [41]). Therefore, we use $O(\log \log n)$ space to find $F(n)$. We remark that when the number $F(n)$ is found, it is written on the work tape, and it is easy to check whether this number is a power of 64, i.e., it must start with 1 and should be followed by only zeros and the number of zeros must be a multiple of 6 ($64 = 2^6$).

For any $I \in \mathcal{I}$, we can define a corresponding language:

$$\texttt{AM75}'(\texttt{I}) = \{a^n \mid a^n \in \texttt{AM75}' \text{ and } \Sigma^*(\log_{64} F(n)) \in I\}.$$

For any input a^n, we can deterministically check whether $a^n \in \texttt{AM75}'$ by using the above algorithm. If not, the input is rejected. Otherwise, we continue with a probabilistic procedure. Notice that the work tape can still contain the binary value of $F(n)$ that is 64^k for some positive integer k in the beginning of the probabilistic procedure.

We use a biased coin landing on head with probability p_I ($\mathfrak{coin}_I$) encoding the memberships of positive integers in I as described before.

By definition we know that $a^n \in \texttt{AM75}'(\texttt{I})$ if and only if $k \in I$. Therefore, if we compute the value of x_k correctly, we are done. Since the work tape contains the value of 64^k, we can toss $\mathfrak{coin}_I$ 64^k times and count the number of heads. Due to Lemma 1, we know that we can correctly compute x_k with probability at least $\frac{3}{4}$. Here the number of heads is kept in binary and we check the $(3k + 3)$-th bit of the result after finishing the all coin tosses. By executing the probabilistic procedure a few more times, the success probability can be increased. Notice that the space used on the work tape does not exceed $O(\log \log n)$. The machine computes the value $F(n)$

in $O(n \log n)$ time steps since it makes $O(\log n)$ iterations and each iteration takes $O(n)$ time steps. During the probabilistic procedure the machine tosses $\texttt{coin}_I$ $O(\log n)$ times, and hence the procedure uses sublinear number of computational steps. Therefore, the total time complexity of the algorithm is $O(n \log n)$.

The cardinality of the set of all subsets of positive integers is uncountably many, and hence the cardinality of the following set

$$\left\{ \texttt{AM75}'(\texttt{I}) \mid I \subseteq \mathbb{Z}^+ \right\}$$

is also uncountably many, each element of which is recognized by a bounded-error unary sweeping PTM using $O(\log \log n)$ space in $O(n \log n)$ time steps. □

With polynomial expected time, we cannot do better since it was proven that bounded-error polynomial-time PTMs using $o(\log \log n)$ space can recognize only regular languages even with unrestricted transition probabilities [14].

On the other hand, with super-polynomial expected time, PTMs can recognize nonregular binary languages even with constant-space [18]. Here we show that PTMs can recognize uncountably many binary languages in arbitrarily small non-constant space and we leave open the case of constant-space. Regarding unary languages we know that constant-space PTMs and $o(\log \log n)$-space one-way PTMs can recognize only regular languages [23, 22], and, up to our knowledge, it is still open whether PTMs can recognize a unary nonregular language in $o(\log \log n)$ space.

For our purpose, we use a fact given by Freivalds in [18] after a slight modification in order to keep the input alphabet binary: For any binary language $L \subseteq \{0,1\}^*$, we define another language $\texttt{LOG}(\texttt{L})$ as follows:

$$\texttt{LOG}(\texttt{L}) = \{0(1w_1)0^{2^1}(1w_2)0^{2^2} \cdots 0^{2^{m-1}}(1w_m)0^{2^m} \mid w = w_1 \cdots w_m \in L\}.$$

Fact 3 (cf. [18])**.** *If a binary language L is recognized by a bounded-error PTM in $s(n)$ space, then the binary language $\texttt{LOG}(\texttt{L})$ is recognized by a bounded-error PTM in $\log(s(n))$ space.*

Theorem 7. *For any $I \in \mathcal{I}$, the language $\texttt{LOG}(\texttt{AM75}'(\texttt{I}))$ can be recognized by a bounded-error PTM in $O(\log \log \log n)$ space.*

Proof. It follows from Theorem 6 and Fact 3. □

Similarly, we can conclude that the language $\texttt{LOG}^\texttt{k}(\texttt{AM75}'(\texttt{I}))$ for $k > 1$ can be recognized by a bounded-error PTM in $O(\log^{k+2} n)$ space.

Corollary 4. *The cardinality of languages recognized by bounded-error PTMs with arbitrary small non-constant space bound is uncountably many.*

Obtained results are valid for any error bound $\epsilon > 0$ because Fact 3 follows from the result that the language

$$\texttt{LOG} = \{010^{2^1}10^{2^2}10^{2^3}\cdots 0^{2^{m-1}}10^{2^m} \mid m > 0\}$$

can be recognized by 2PFAs with any error bound [18]. We can also conclude that the statement about space bounds for the language LOG(L) is also valid for 2PCAs.

Corollary 5. *If a binary language L is recognized by a bounded-error 2PCA in $s(n)$ space, then the binary language* LOG(L) *is recognized by a bounded-error 2PCA in $\log(s(n))$ space.*

We continue with the models that have the most restricted versions of two-way input head. Since restarting realtime automata are equivalent to postselecting realtime automata, we consider postselecting realtime machines as very restricted variation of two-way machines.

First, we adopt and also simplify the techniques presented in [18, 11, 46]. We start with a simple language:

$$\texttt{EQUAL} = \{0^m10^m \mid m > 0\}.$$

It is known that EQUAL is recognized by realtime PostPFAs with bounded error [46, 49], but we still present an explicit proof that will be used in the other proofs.

Fact 4. *For any $x < \frac{1}{2}$,* EQUAL *is recognized by a realtime PostPFA M_x with error bound $\frac{2x}{2x+1}$.*

Proof. Let $w = 0^m10^{m'}$ be the given input for some $m, m' > 0$. Any other input is rejected deterministically.

At the beginning of the computation, M_x splits the computation into two paths with equal probabilities. In the first path, M_x says "A" with probability $Pr[A] = x^{2m+2m'}$, and, in the second path, it says "R" with probability $Pr[R] = \left(\frac{x^{4m} + x^{4m'}}{2}\right)$.

In the first path, M_x starts in a state, say s_A. Then, for each symbol 0, it stays in s_A with probability x^2 and quits s_A with the remaining probability. Thus, when started in s_A, the probability of being in s_A upon reaching on

the right end-marker is

$$\underbrace{x^2 \cdot x^2 \cdot \ \cdots \ \cdot x^2}_{m \text{ times}} \cdot \underbrace{x^2 \cdot x^2 \cdot \ \cdots \ \cdot x^2}_{m' \text{ times}} = x^{2m} \cdot x^{2m'} = x^{2m+2m'}.$$

In the second path, we assume that M_x starts in a state, say s_R, and then immediately switches to two different states, say s_{R1} and s_{R2}, with equal probabilities. For each 0 until the symbol 1, M_x stays in s_{R1} with probability x^4 and quits s_{R1} with the remaining probability. After reading symbol 1, it switches from s_{R1} to s'_{R1} and stays there until the right end-marker. Thus, when started in s_{R1}, the probability of being in s'_{R1} upon reaching on the right end-marker is x^{4m}.

When in s_{R2}, M_x stays in s_{R2} on the first block of 0's. After reading symbol 1, it switches from s_{R2} to s'_{R2}, and then, for each 0, it stays in s'_{R2} with probability x^4 and quits s'_{R2} with the remaining probability. Thus, when started in s_{R2}, the probability of being in s'_{R2} upon reaching on the right end-marker is $x^{4m'}$. Therefore, when started in state s_R, the probability of being in s'_{R1} or s'_{R2} upon reaching on the right end-marker is

$$\frac{x^{4m} + x^{4m'}}{2}.$$

It is easy to see that if $m = m'$, then $Pr[A] = Pr[R] = x^{4m}$. On the other hand, if $m \neq m'$, then

$$\frac{Pr[R]}{Pr[A]} = \frac{\frac{x^{4m}+x^{4m'}}{2}}{x^{2m+2m'}} = \frac{x^{2m-2m'}}{2} + \frac{x^{2m'-2m}}{2} > \frac{1}{2x^2}$$

since either $(2m - 2m')$ or $(2m' - 2m)$ is a negative even integer.

On the right end-marker, M_x enters s_{acc} and s_{rej} with probabilities $Pr[A]$ and $(x \cdot Pr[R])$, respectively. Therefore, if w is a member, then $a(w)$ is x^{-1} times of $r(w)$, and hence, w is accepted with probability

$$\frac{x^{-1}}{1 + x^{-1}} = \frac{1}{x+1}.$$

If w is not a member, then $r(w)$ is at least $\frac{1}{2x}$ times of $a(w)$, and hence, w is rejected with probability at least

$$\frac{(2x)^{-1}}{1 + (2x)^{-1}} = \frac{1}{2x+1}.$$

Thus, the error bound ϵ is $\frac{2x}{2x+1}$, i.e.,

$$\epsilon = \max\left(1 - \frac{1}{x+1}, 1 - \frac{1}{2x+1}\right) = 1 - \frac{1}{2x+1} = \frac{2x}{2x+1},$$

which is less than $\frac{1}{2}$ when $x < \frac{1}{2}$. (Notice that $\epsilon \to 0$ when $x \to 0$.) □

We continue with language

$$\texttt{EQUAL-BLOCKS} = \{0^{m_1}10^{m_1}10^{m_2}10^{m_2}1 \cdots 10^{m_t}10^{m_t} \mid t > 0\}.$$

Theorem 8. *For any $x < \frac{1}{2}$, EQUAL-BLOCKS is recognized by a realtime PostPFA M_x with error bound $\frac{2x}{2x+1}$.*

Proof. Let $w = 0^{m_1}10^{m'_1}10^{m_2}10^{m'_2}1 \cdots 10^{m_t}10^{m'_t}$ be the given input for some $t > 0$, where for each $i \in \{1, \ldots, t\}$ both m_i and m'_i are positive integers. Any other input is rejected deterministically.

Similarly to previous proof, after reading whole input, M_x says "A" with probability

$$Pr[A] = \underbrace{\left(x^{2m_1+2m'_1}\right)}_{a_1}\underbrace{\left(x^{2m_2+2m'_2}\right)}_{a_2} \cdots \underbrace{\left(x^{2m_t+2m'_t}\right)}_{a_t}$$

and says "R" with probability

$$Pr[R] = \underbrace{\left(\frac{x^{4m_1} + x^{4m'_1}}{2}\right)}_{r_1}\underbrace{\left(\frac{x^{4m_2} + x^{4m'_2}}{2}\right)}_{r_2} \cdots \underbrace{\left(\frac{x^{4m_t} + x^{4m'_t}}{2}\right)}_{r_t}.$$

Here M_x can easily implement both probabilistic events by help of internal states. As analyzed in the previous proof, for each $i \in \{1, \ldots, t\}$, either $a_i = r_i$ or r_i is at least $\frac{1}{2x^2}$ times greater than a_i. Thus, if w is a member, then $Pr[A] = Pr[R]$, and, if w is not a member, then

$$\frac{Pr[R]}{Pr[A]} > \frac{1}{2x^2}.$$

On the right end-marker, M_x enters s_{acc} and s_{rej} with probabilities $Pr[A]$ and $(x \cdot Pr[R])$, respectively. Therefore, we obtain the same error bound as given in the previous proof. □

Let f be the linear mapping $f(m) = am + b$ for some nonnegative integers

a and b, and, let

$$\texttt{EQUAL-BLOCKS(f)} = \{0^{m_1}10^{f(m_1)}10^{m_2}10^{f(m_2)}1\cdots 10^{m_t}10^{f(m_t)} \mid t > 0\}$$

be a new language.

Theorem 9. *For any $x < \frac{1}{2}$, $\texttt{EQUAL-BLOCKS(f)}$ is recognized by a realtime PostPFA M_x with error bound $\frac{2x}{2x+1}$.*

Proof. Let $w = 0^{m_1}10^{m'_1}10^{m_2}10^{m'_2}1\cdots 10^{m_t}10^{m'_t}$ be the given input for some $t > 0$, where for each $i \in \{1,\ldots,t\}$ both m_i and m'_i are positive integers. Any other input is rejected deterministically.

In the above proofs, the described automata make transitions with probabilities x^2 or x^4 when reading a symbol 0. Here M_x makes some additional transitions:

- Before starting to read a block of 0's, M_x makes a transition with probability x^{2b} or x^{4b}.
- After reading a symbol 0, M_x makes a transition with probability x^{2a} or x^{4a}.

Thus, after reading a block of m 0's, M_x can be designed to be in a specific event with probability $x^{2am+2b} = x^{2f(m)}$ or $x^{4am+4b} = x^{4f(m)}$, where $m > 0$.

Therefore, M_x is constructed in a way that, after reading whole input, it says "A" with probability

$$Pr[A] = \underbrace{\left(x^{2f(m_1)+2m'_1}\right)}_{a_1}\underbrace{\left(x^{2f(m_2)+2m'_2}\right)}_{a_2}\cdots\underbrace{\left(x^{2f(m_t)+2m'_t}\right)}_{a_t}$$

and says "R" with probability

$$Pr[R] = \underbrace{\left(\frac{x^{4f(m_1)}+x^{4m'_1}}{2}\right)}_{r_1}\underbrace{\left(\frac{x^{4f(m_2)}+x^{4m'_2}}{2}\right)}_{r_2}\cdots\underbrace{\left(\frac{x^{4f(m_t)}+x^{4m'_t}}{2}\right)}_{r_t}.$$

Then, for each $i \in \{1,\ldots,t\}$, if $m'_i = f(m_i)$, $a_i = r_i = x^{4f(m_i)}$, and, if $m'_i \neq f(m_i)$,

$$\frac{r_i}{a_i} = \frac{\frac{x^{4f(m_i)}+x^{4m'_i}}{2}}{x^{2f(m_i)+2m'_i}} = \frac{x^{2f(m_i)-2m'_i}}{2} + \frac{x^{2m'_i-2f(m_i)}}{2} > \frac{1}{2x^2}.$$

As in the above algorithms, on the right end-marker, M_x enters s_{acc} and s_{rej} with probabilities $Pr[A]$ and $(x \cdot Pr[R])$, respectively. Therefore, we obtain the same error bound as given in the previous two proofs. □

As an application of the last result, we present a realtime PostPFA algorithm for language

$$\texttt{LOG} = \{010^{2^1}10^{2^2}10^{2^3}\cdots 0^{2^{m-1}}10^{2^m} \mid m > 0\},$$

which was also shown to be recognized by 2PFAs [18].

Theorem 10. *For any $x < \frac{1}{2}$, LOG is recognized by a realtime PostPFA M_x with error bound $\frac{2x}{2x+1}$.*

Proof. Let $0^{2^0}10^{m_1}10^{m_2}1\ldots10^{m_t}$ be the given input for $t > 1$, where $m_1 = 2^1$. The decision on any other input is given deterministically.

After reading the whole input, M_x says "A" with probability

$$Pr[A] = \underbrace{\left(x^{4m_1+2m_2}\right)}_{a_1}\underbrace{\left(x^{4m_2+2m_3}\right)}_{a_2}\cdots\underbrace{\left(x^{4m_{t-1}+2m_t}\right)}_{a_{t-1}}$$

and says "R" with probability

$$Pr[R] = \underbrace{\left(\frac{x^{8m_1}+x^{4m_2}}{2}\right)}_{r_1}\underbrace{\left(\frac{x^{8m_2}+x^{4m_3}}{2}\right)}_{r_2}\cdots\underbrace{\left(\frac{x^{8m_{t-1}}+x^{4m_t}}{2}\right)}_{r_{t-1}}.$$

In the previous languages, the blocks are nicely separated, but the blocks are overlapping for language LOG. Therefore, we modify the previous methods. As described in the first algorithm, M_x splits the computation into two paths with equal probabilities at the beginning of the computation. In the first path, the event happening with probability $Pr[A]$ is implemented by executing two parallel procedures: The first procedure produces the probabilities a_i's where i is odd and the second procedure produces the probabilities a_i's where i is even. Similarly, in the second path, the event happening with probability $Pr[R]$ is implemented by also executing two parallel procedures. Thus, the previous algorithm is also used for LOG by using the solution for overlapping blocks. □

Similarly to the padding argument presented in Fact 3, we can easily obtain the following two corollaries for a realtime PostPTM and a realtime PostPCA.

Corollary 6. *If a binary language L is recognized by a bounded-error realtime PostPTM in $s(n)$ space, then the binary language* $\texttt{LOG(L)}$ *is recognized by a bounded-error realtime PostPTM in* $\log(s(n))$ *space.*

Corollary 7. *If a binary language L is recognized by a bounded-error realtime PostPCA in $s(n)$ space, then the binary language* $\texttt{LOG(L)}$ *is recognized by a bounded-error realtime PostPCA in* $\log(s(n))$ *space.*

Now, we show how to obtain results for arbitrarily small space bounds by restricting two-way reading mode to realtime reading mode with postselection. We start with the recognition of the following nonregular binary language, a modified version of DIMA[1]:

$$\texttt{DIMA3} = \{0^{2^0}10^{2^1}10^{2^2}1\cdots10^{2^{6k-2}}110^{2^{6k-1}}11^{2^{6k}}(0^{2^{3k}-1}1)^{2^{3k}} \mid k>0\}.$$

Theorem 11. *For any $x < \frac{1}{3}$,* DIMA3 *is recognized by linear-space realtime PostPCA M_x with error bound $\frac{x}{1+x}$.*

Proof. Let w be the given input of the form

$$w = 0^{t_1}10^{t_2}1\cdots10^{t_{m-1}}110^{t_m}11^{t'_0}0^{t'_1}10^{t'_2}1\cdots10^{t'_{m'}}1,$$

where $t_1 = 1$, m and m' are positive integers, m is divisible by 6, and $t_i, t'_j > 0$ for $1 \leq i \leq m$ and $0 \leq j \leq m'$. (Otherwise, the input is rejected deterministically.)

M_x splits computation into four paths with equal probabilities. In the first path, with the help of the counter, M_x makes the following comparisons:

- for each $i \in \{1,\ldots,\frac{m}{2}\}$, whether $2t_{2i-1} = t_{2i}$,
- for each $j \in \{1,\ldots,\frac{m'}{2}\}$, whether $t'_{2j-1} = t'_{2j}$.

In the second path, with the help of the counter, M_x makes the following comparisons:

- for each $i \in \{1,\ldots,\frac{m}{2}-1\}$, whether $2t_{2i} = t_{2i+1}$,
- whether $2t_m = t'_0$ (this also helps to set the counter to 0 for the upcoming comparisons),
- for each $j \in \{1,\ldots,\frac{m'}{2}-1\}$, whether $t'_{2j} = t'_{2j+1}$.

[1]Historiacally, we defined first DIMA, and then DIMA2, and lastly DIMA3. But due to the structure of thesis, we mention DIMA3 before DIMA.

In the third path, M_x checks whether $1 + \sum_{i=1}^{m} t_i = m' + \sum_{j=1}^{m'} t'_j$. In the fourth path, M_x checks whether $t'_1 + 1 = m'$.

It is easy to see that all comparisons are successful if and only if $w \in$ DIMA3.

If every comparison in a path is successful, then M_x enters s_{acc} with probability $\frac{x}{3}$ in the path. If it is not, then M_x enters s_{rej} with probability 1 in the path. Therefore, if $w \in$ DIMA3, then w is accepted with probability 1 since $r(w) = 0$. If $w \notin$ DIMA3, then the maximum accepting probability is obtained when M_x enters s_{rej} only in one of the paths. That is,

$$\frac{r(w)}{a(w)} = \frac{\frac{1}{4}}{3 \cdot \frac{1}{4} \cdot \frac{x}{3}} = \frac{1}{x}.$$

Thus, w is rejected with probability at least $\frac{1}{1+x}$. The error bound is $\frac{x}{1+x}$. □

Theorem 12. *Linear-space realtime PostPCAs can recognize uncountably many languages with error bound* $\frac{2}{5}$.

Proof. Let w_k be the k-th shortest member of DIMA3 for $k > 0$. For any $I \in \mathcal{I}$, we define the following language:

$$\texttt{DIMA3}(I) = \{w_k \mid k > 0 \text{ and } k \in I\}.$$

We obtain our result by presenting a realtime PostPCA, say $M_{I,y}$, to recognize DIMA3(I), where $y < \frac{1}{19}$. Let w be the given input of the form

$$w = 0^{t_1}10^{t_2}1 \cdots 10^{t_{m-1}}110^{t_m}11^{t'_0}0^{t'_1}10^{t'_2}1 \cdots 10^{t'_{m'}}1,$$

where $t_1 = 1$, m and m' are positive integers, m is divisible by 6, and $t_i, t'_j > 0$ for $1 \leq i \leq m$ and $0 \leq j \leq m'$. (Otherwise, the input is rejected deterministically.)

At the beginning of the computation, $M_{I,y}$ splits into two paths with equal probabilities. In the first path, $M_{I,y}$ executes the realtime PostPCA M_y for DIMA3 described in the proof above with the following modification: In each path of M_y, if every comparison is successful, then M_y enters state s_{acc} with probability $\frac{y}{16}$ (M_y enters the path with probability $\frac{1}{4}$, and then enters state s_{acc} with probability $\frac{y}{4}$), and, if it is not, then M_y enters state s_{rej} with probability 1.

In the second path, $M_{I,y}$ sets the value of counter to $c = 1 + \sum_{j=1}^{m} t_i$ by reading the part of the input $0^{t_1}10^{t_2}1 \cdots 10^{t_{m-1}}110^{t_m}1$. We remark that if $w \in$ DIMA3, c is 64^k for some $k > 0$. Then, $M_{I,y}$ attempts to toss $\mathtt{coin}_I$ c times. After each coin toss, if the result is heads (resp., tails), then $M_{I,y}$

moves on the input two symbols (resp., one symbol). If h is the number of total heads, then $M_{I,y}$ reads $(c-h)+2h = c+h$ symbols. During attempt to read $c+h$ symbols, if the input is finished, then the computation ends in state s_{rej} with probability 1 in this path. Otherwise, $M_{I,y}$ guesses the value x_k with probability at least $\frac{3}{4}$ (described in details at the end of the proof) and gives a parallel decision with probability y, i.e., if the guess is 1 (resp., 0), then it enters state s_{acc} (resp., s_{rej}) with probability y.

If $w \in \texttt{DIMA3}(I)$, then the probability of entering state s_{acc} is $\left(4 \cdot \frac{y}{16}\right)$ in the first path and at least $\frac{3y}{4}$ in the second path. The probability of entering s_{rej} in the second path is at most $\frac{y}{4}$. Thus, w is accepted with probability at least $\frac{4}{5}$.

If $w \notin \texttt{DIMA3}(I)$, then we have two cases. Case 1: $w \in \texttt{DIMA3}$. In this case, the probability of entering state s_{acc} is $\left(4 \cdot \frac{y}{16}\right)$ in the first path and at most $\frac{y}{4}$ in the second path. The probability of entering s_{rej} in the second path is at least $\frac{3y}{4}$. Thus, w is rejected with probability at least $\frac{3}{5}$.

Case 2: $w \notin \texttt{DIMA3}$. In this case, the probability of entering state s_{rej} is at least $\frac{1}{8}$ in the first path and this is at least 4 times of the total probability of entering state s_{acc}, which can be at most

$$\frac{1}{2} \cdot 3 \cdot \frac{y}{16} + \frac{1}{2}y = \frac{19y}{32} < \frac{1}{32}$$

for $y < \frac{1}{19}$. Then, the input is rejected with probability greater than $\frac{4}{5}$.

As can be seen from the above analysis, when $w \notin \texttt{DIMA3}$, guessing the correct value of x_k is insignificant. Therefore, in the following part, we assume that $w \in \texttt{DIMA3}$ when explaining how to guess x_k correctly. Thus, we assume that $w = w_k$:

$$w_k = 0^{2^0}10^{2^1}10^{2^2}1 \cdots 10^{2^{6k-2}}110^{2^{6k-1}}11^{2^{6k}}(0^{2^{3k}-1}1)^{2^{3k}}$$

for $k > 0$. In the second path, $M_{I,y}$ tosses $\texttt{coin}_I$ $c = 64^k$ times and it can read $64^k + h$ symbols from the input. In other words, it reads h symbols from the part $w'_k = (0^{2^{3k}-1}1)^{2^{3k}}$. We can write h as

$$h = i \cdot 8^{k+1} + j \cdot 8^k + q = (8i+j)8^k + q,$$

where $i \geq 0$, $j \in \{0, \ldots, 7\}$, and $q < 8^k$.

Due to Lemma 1, x_k is the $(3k+3)$-th digit of $bin(h)$ with probability $\frac{3}{4}$. In other words, x_k is guessed as 1 if $j \in \{4, \ldots, 7\}$, and as 0, otherwise. $M_{I,y}$ sets $j = 0$ at the beginning. We can say that for each heads, it consumes a symbol from w'_k. After reading 8^k symbols, it updates j as $(j+1) \mod 8$. When the value of counter reaches zero, $M_{I,y}$ guesses x_k by checking the

value of j. □

Now we can combine Corollary 7 and Theorem 12 to obtain new results for uncountable probabilistic classes.

Corollary 8. *The cardinality of languages recognized by bounded-error realtime PostPCAs with arbitrary small non-constant space bound is uncountably many.*

Note that if we replace realtime PostPCA with sweeping PCA in Theorem 12, then all checks can be repeated multiple times, and therefore, the error bound can be decreased arbitrarily close to zero. By combining this result with Corollary 7, we can obtain arbitrarily small error bound for sweeping PCAs with arbitrary small non-constant space bound that recognize uncountably many languages.

We have obtained arbitrarily small space bounds for very restricted version of two-way machines that recognize uncountably many languages with bounded error. Currently, it is an open problem whether two-way machines with constant space (2PFAs) are capable of recognition of uncountably many binary languages with bounded error. On the other hand, verification of uncountably many languages is possible in constant space, and we continue with these results.

4.1.2 Verifiers

We present two nonregular unary languages and one nonregular binary language that can be verified by 2PFAs in quadratic and linear time, respectively. The protocols presented here will be also used for our results about uncountably many languages.

Theorem 13. $\texttt{USQUARE} = \{a^{m^2} \mid m > 0\}$ *is verifiable by a 2PFA in quadratic expected time with any error bound.*

Proof. The protocol is one-way and the verifier expects from the prover a string of the form

$$(a^m b)^m b$$

for the members of the language, where $m > 0$.

Let $w = a^n$ be the given input for $n > 3$ (the decisions on the shorter strings are given deterministically) and let y be the string provided by the prover. The verifier deterministically checks whether y is of the form

$$y = a^{m_1} b a^{m_2} b \cdots b a^{m_i} b \cdots \text{ or } y = a^{m_1} b a^{m_2} b \cdots b a^{m_t} b b$$

for some $t > 0$. If the verifier sees a defect on y, then the input is rejected.

In the remaining part, we assume that y is in one of these forms. At the beginning of the computation, the verifier places the input head on the left end-marker and splits the computation in four paths with equal probabilities.

In the first path, the verifier reads w and y in parallel and checks whether y is finite, i.e.,

$$y = a^{m_1}ba^{m_2}b\cdots ba^{m_t}bb,$$

and whether it satisfies the equality $n = \sum_{j=1}^{t} m_j$, where $t > 1$. If one of the checks fails, the input is rejected. Otherwise, it is accepted.

The second path is very similar to the first path and the following equality is checked:

$$n = \sum_{j=2}^{t} m_j + \sum_{j=1}^{t} 1,$$

i.e., the verifier skips a^{m_1} from y and counts b's instead. If the equality is satisfied, the input is accepted. Otherwise, it is rejected.

The computation in the first and second paths is deterministic (a single decision is given in each) and both paths terminate in linear time.

In the third path, the verifier tries to make the following consecutive comparisons:

$$m_1 = m_2, m_3 = m_4, \ldots, m_{2j-1} = m_{2j}, \ldots.$$

For each j, the verifier can easily determine whether $m_{2j-1} = m_{2j} < n$ by attempting to move the input head to the right by m_{2j-1} squares and then to the left by m_{2j} squares. If the right end-marker is visited ($m_{2j-1} \geq n$), or the left end-marker is visited earlier than expected ($m_{2j} > m_{2j-1}$) or is not visited ($m_{2j} < m_{2j-1}$), then the comparison is not successful, and hence the input is rejected. Otherwise, the comparison is successful and the verifier continues with a random walk (described below) before the next comparison, except that if the last comparison is successful, then the input is accepted without making the random walk.

The aim of the random walk is to determine whether the prover sends a finite string or not, i.e., the prover may cheat by sending the infinite string $(a^m b)^*$ for some $m < n$, which passes successfully all comparison tests described above.

The random walk starts by placing the input head on the first symbol of the input and terminates after hitting one of the end-markers. During the random walk, the verifier pauses the reading of the string y. It is a well-known fact that this walk terminates in $O(n)$ expected number of steps and the probability of ending on the right (resp., the left) end-marker is $\frac{1}{n}$ (resp.,

$1 - \frac{1}{n}$).

If the walk ends on the left end-marker, then the verifier continues with the next comparison. If the walk ends on the right end-marker, the verifier checks whether the number of a's in the remaining part of y is less than n or not by reading whole input from right to left. If it is less than n, then the input is accepted. Otherwise (y contains more than n a's), the input is rejected. In any case, the computation is terminated with probability $\frac{1}{n}$ after the walk.

The fourth path is identical to the third path by shifting the comparing pairs: The verifier tries to make the following consecutive comparisons:

$$m_2 = m_3, m_4 = m_5, \ldots, m_{2j} = m_{2j+1}, \ldots.$$

Now, we can analyze the overall protocol. If $n = m^2$ for some $m > 1$, then the prover provides $y = (a^m b)^m b$ and the input is accepted in every path, and hence the overall accepting probability is 1. Thus, every member is accepted with probability 1. Moreover there will be at most m random walks, and hence the overall running time is $O(n\sqrt{n})$.

If the input is not a member, then the input is rejected in at least one of the paths. If it is rejected in the first or second path, then the overall rejecting probability is at least $\frac{1}{4}$. If it is rejected in the third or fourth paths, then the overall rejecting probability cannot be less than $\frac{3}{16}$ as explained below.

We assume that the input is not rejected in the first and second paths. Then, we know that y is finite, the number of a's in y is n, and y is composed by m_1 blocks. Since n is not a perfect square, there is at least one pair of consecutive blocks that have different number of a's. Therefore, at least one of the comparisons will not be successful, and, the input will be rejected in one of these paths. Let l be the minimum index such that the comparison of the l-th pair is not successful (in the third or fourth path). Then, $l < \frac{\sqrt{n}}{2}$. (If not, y contains at least $2\left\lceil \frac{\sqrt{n}}{2} \right\rceil$ blocks and each of these blocks contains $m_1 = \lceil \sqrt{n} \rceil$ a's, and, this implies that y contains more than n a's.) Then, the maximum accepting probability in the corresponding path is bounded from above by

$$\sum_{i=1}^{l} \frac{1}{n}\left(1 - \frac{1}{n}\right)^{i-1} = 1 - \left(1 - \frac{1}{n}\right)^{l} < 1 - \left(1 - \frac{1}{n}\right)^{\frac{\sqrt{n}}{2}} \leq \frac{1}{4}.$$

(Remember that $n > 3$.) Therefore, the rejecting probability in the third or fourth path is at least $\frac{3}{4}$, and hence, the overall rejecting probability cannot be less than $\frac{1}{4} \cdot \frac{3}{4} = \frac{3}{16}$.

The maximum (expected) running time occurs when the prover sends the infinite $y = (a^m b)^*$ for some $m > 0$. In this case, the protocol is terminated in the third and fourth paths with probability 1 after $O(n)$ random walks, and hence, the expected running time is quadratic in n, $O(n^2)$.

By repeating the protocol above many times, say $r > 1$, we obtain a new protocol such that any non-member is rejected with probability arbitrarily close to 1, i.e., $1 - \left(1 - \frac{3}{16}\right)^r$. □

Theorem 14. `UPOWER64` $= \{a^{64^m} \mid m > 0\}$ *is verifiable by a 2PFA in quadratic expected time with any error bound.*

Proof. The proof is very similar to the proof given for `USQUARE`. The protocol is one-way and the verifier expects to receive the string

$$y_n = aba^{64}b \cdots ba^{64^{k-2}}ba^{64^{k-1}}bb$$

from the prover for some $k > 0$.

Let $w = a^n$ be the given input for $n > 64$. (The decisions on the shorter strings are given deterministically.) Let y be the string sent by the prover. The verifier deterministically checks whether y is of the form

$$y = a^{m_1}ba^{m_2}b \cdots ba^{m_i}b \cdots \text{ or } y = a^{m_1}ba^{m_2}b \cdots ba^{m_t}bb$$

for some $t > 0$, where $m_1 = 1$. If the verifier sees a defect on y, the input is rejected. Therefore, we assume that y is in one of these forms in the remaining part.

The verifier splits into three paths with equal probabilities at the beginning of the computation. The first path checks whether y is finite and

$$n = 1 + 63 \cdot \sum_{j=1}^{t} m_j.$$

If not, the input is rejected (because $\sum_{j=0}^{k-1} 64^j = \frac{64^k - 1}{63}$). Otherwise, the input is accepted.

The second and third paths are very similar to the third and fourth paths in the proof for `USQUARE`. In the second path, the verifier checks

$$64 \cdot m_{2j-1} = m_{2j}$$

for each $j > 0$. The random walk part is implemented in the same way. The third path is the same except the comparing pairs: The verifier checks

$$64 \cdot m_{2j} = m_{2j+1}$$

for each $j > 0$.

If $w = a^{64^k}$, then the honest prover sends

$$y_m = aba^{64}b \cdots ba^{64^{k-1}}bb$$

and the input is accepted in all paths.

If w is not a member, then the input is rejected in at least one of the paths. If it is rejected in the first path, then the overall rejecting probability is at least $\frac{1}{3}$. If the input is accepted in the first path, then the rejecting probability in the second or third path is at least $\frac{99}{100}$, and hence the overall rejecting probability is greater than $\frac{33}{100}$.

We can use the analysis given in the previous proof. Let l be the minimum index such that the comparison of l-th pair is not successful (in the second or third path). Then, $l < \frac{\log_{64} n}{2} = \frac{\log n}{12}$. (If not, y starts with

$$aba^{64}b \cdots ba^{64^{2l-3}}ba^{64^{2l-2}}ba^{64^{2l-1}}$$

where $l > \frac{\log_{64} n}{2}$, and hence, y contains $\frac{64^{2l}-1}{63}$ a's, which is greater than $\frac{n-1}{63}$.) The maximum accepting probability in the corresponding path can be bounded from above by

$$\sum_{i=1}^{l} \frac{1}{n}\left(1-\frac{1}{n}\right)^{i-1} = 1 - \left(1-\frac{1}{n}\right)^{l} < 1 - \left(1-\frac{1}{n}\right)^{\frac{\log n}{12}} < \frac{1}{100}.$$

(Remember that $n > 64$.) Therefore, the rejecting probability in the second or third path is greater than $\frac{99}{100}$.

The maximum expected running time is quadratic in n, when the prover sends the infinite $y = aba^{64}b \cdots ba^{64^i}ba^{64^{i+1}}b \cdots$. By repeating the protocol many times, we obtain a protocol with better success probability. □

We continue with the verification of a binary nonregular language, a modified version of DIMA[2]:

$$\texttt{DIMA2} = \{0^{2^0}10^{2^1}10^{2^2}1 \cdots 10^{2^{3k-1}}11(0^{2^{3k}}1)^{2^{3k}} \mid k > 0\}.$$

Theorem 15. DIMA2 *is verifiable by a sweeping PFA in linear time with any error bound.*

Proof. The protocol is one-way and the verifier expects to receive the same input from the prover.

[2]Historically DIMA appeared earlier than DIMA2.

Let w be the given input of the form

$$w = 0^{t_1}10^{t_2}1\cdots 10^{t_m}110^{t'_1}10^{t'_2}1\cdots 10^{t'_{m'}}1,$$

where $t_1 = 1$, m and m' are positive integers, m is divisible by 3, and $t_i, t'_j > 0$ for $1 \leq i \leq m$ and $1 \leq j \leq m'$. (Otherwise, the input is rejected deterministically.)

Let y be the string provided by the prover. When on the left end-marker, the verifier splits into three paths with equal probabilities. In the first path, it checks whether $w = y$. If not, the input is rejected. Otherwise, the input is accepted.

In the following part, we assume that $y = w$. In the second path, the verifier checks whether each 0-block has double length of the previous 0-block before symbols "11" and each 0-block has the same length of the previous 0-block after symbols "11": When reading w and y in parallel, the verifier makes the following comparisons:

- for each $i \in \{1, \ldots, m-1\}$, whether $2t_i = t_{i+1}$,
- whether $2t_m = t'_1$, and,
- for each $j \in \{1, \ldots, m'-1\}$, whether $t'_j = t'_{j+1}$.

If one of the comparisons is not successful, then the input is rejected. If all of them are successful, then we know that the input is of the form

$$w = 0^{2^0}10^{2^1}10^{2^2}1\cdots 10^{2^{3k-1}}11(0^{2^{3k}}1)^{m'}.$$

In the third path, the verifier simply checks whether $m' = 2^{3k}$ or not, i.e., the verifier compares t'_1 from y with the number of 1's appearing after "11" in w. If not, the input is rejected. Otherwise, it is accepted.

If w is a member, then the honest prover sends $y = w$, and the verifier accepts the input in all paths with probability 1.

If w is not a member, then the verifier rejects the input in one of the paths. If the prover sends $y \neq w$, then the input is rejected in the first path. If $y = w$ and then one of the comparisons in the second path may not be successful, and then the input is rejected in this path. If all of them are successful, then the comparison in the third path cannot be successful and the input is rejected in this path.

The overall protocol terminates in linear time. By repeating the protocol many times, we obtain a protocol with better success probability. □

Now we are ready to present two constant-space protocols for verifying uncountably many unary and binary languages.

Theorem 16. *Sweeping PFAs can verify uncountably many languages in linear time with any error bound.*

Proof. Let w_k be the k-th shortest member of DIMA2 for $k > 0$. For any $I \in \mathcal{I}$, we define the following language:

$$\texttt{DIMA2}(I) = \{w_k \mid k > 0 \text{ and } k \in I\}.$$

We describe a one-way protocol for $\texttt{DIMA2}(I)$. Let w be the given input. The verifier determines whether $w = w_k$ for some $k > 0$ by using the protocol for DIMA2 with high probability. If not, then the input is rejected. In the remaining part, we continue with

$$w = w_k = 0^{2^0}10^{2^1}10^{2^2}1\cdots 10^{2^{3k-1}}11(0^{2^{3k}}1)^{2^{3k}}.$$

The verifier attempts to toss $\texttt{coin}_I$ 64^k times and in parallel processes the total number of heads for determining x_k in p_I. The verifier asks from the prover to send $0^{64^k}1$ and the prover sends $y = 0^m1$ (the input is deterministically rejected if y is not in this form).

The verifier splits into two paths with equal probabilities. In the first path, it easily determines whether $m = 64^k$ by passing over the input once (the number of 0's after symbols "11" is 64^k). If $m \neq 64^k$, then the input is rejected. Otherwise, the input is accepted.

The second path is responsible for coin-tosses and processing the total number of heads. The verifier performs m coin-tosses. For counting the heads, the verifier uses the part of w_k after symbols "11" as a read-only counter, which is composed of 8^k blocks of 0's and length of each block is 8^k. Let h be the total number of heads:

$$h = i \cdot 8^{k+1} + j \cdot 8^k + q = (8i + j)8^k + q,$$

where $i \geq 0$, $j \in \{0, \ldots, 7\}$, and $q < 8^k$. Due to Lemma 1, x_k is the $(3k+3)$-th digit of $bin(h)$ with probability at least $\frac{3}{4}$. In other words, x_k is guessed as 1 if $j \in \{4, \ldots, 7\}$, and as 0, otherwise. The verifier sets $j = 0$ at the beginning. Then, for each heads, it reads a symbol 0 from the input and after 8^k heads it updates j as $(j + 1) \mod 8$. If the number of heads exceeds 64^k, then the input is rejected. If not, the decision given is parallel to the value of j: The input is accepted if $j \in \{4, \ldots, 7\}$ and rejected if $j \in \{0, \ldots, 3\}$.

The verifier operates in sweeping mode and each path terminates in linear time. If w is a member, then the input is accepted with probability at least $\frac{3}{4}$. If $w \notin \texttt{DIMA2}$, then it is rejected with high probability. If $w \in \texttt{DIMA2}$ and $w \notin \texttt{DIMA2}(I)$, then the input is rejected with probability at least $\frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8}$.

By repeating the protocol, the probability to correctly compute the value x_k and catch the prover in case of cheating can get arbitrarily close to 1.

Since the cardinality of set $\{I \mid I \in \mathcal{I}\}$ is uncountable, there are uncountably many languages in $\{\texttt{DIMA2}(I) \mid I \in \mathcal{I}\}$, each of which is verified by a bounded-error linear-time sweeping PFA. □

Theorem 17. *2PFAs can verify uncountably many unary languages in quadratic expected time with any error bound.*

Proof. Here we use all protocols given in the proofs of Theorems 13, 14, and 16.

Let w_k be the k-th shortest member of UPOWER64 for $k > 0$. For any $I \in \mathcal{I}$, we define language:

$$\texttt{UPOWER64}(I) = \{w_k \mid k > 0 \text{ and } k \in I\}.$$

We construct a verifier for this language.

Let w be the given input. By using the protocol given for UPOWER64, the verifier can determine whether $w \in$ UPOWER64 or not. If $w \notin$ UPOWER64, then the input is rejected with high probability.

In the following part, we assume that $w = w_k$ for some $k > 0$. The verifier asks from the prover the following string: $y_k = (a^{8^k}b)^{8^k}b$. Let y be the string provided by the prover. The verifier splits into two paths with equal probabilities. In the first path, it checks whether $y = y_k$ by using the protocol for USQUARE. In the second path, the verifier assumes that $y = y_k$ and implements the part of the protocol for DIMA2(I), which is responsible for the coin tosses and for determining whether $k \in I$ or not by checking the total number of heads. The verifier reads w for 64^k coin tosses and y for determining the value x_k.

If the prover sends y_k, then the verifier correctly determines whether $k \in I$ or not with probability at least $\frac{3}{4}$ in the second path. If $w \in$ UPOWER64(I), the honest prover sends y_k, and hence the input is accepted with probability 1 in the first path and accepted with probability at least $\frac{3}{4}$ in the second path.

If $w \notin$ UPOWER64(I), then the input is rejected with probability at least $\frac{3}{16}$ if the prover does not send y_k in the first path, and rejected with probability at least $\frac{3}{4}$ in the second path if the prover does send y_k. Therefore, the overall rejecting probability is at least $\frac{3}{32}$.

Since each called protocol runs no more than quadratic expected time in $|w|$, the running time is quadratic. By repeating the protocol many times, we obtain a protocol with better success probability. □

We continue with realtime PostPFA protocols for nonregular unary languages

$$\texttt{UPOWER} = \{a^{2^m} \mid m > 0\}$$

and `USQUARE`. We have shown how to verify `USQUARE` by a 2PFA verifier, and the verification of `UPOWER64` is similar to the verification of `UPOWER`. It is also known that these languages are verifiable by private alternating realtime automata [12]. Here we use similar protocols but with certain modifications for realtime PostPFAs.

Theorem 18. `UPOWER` *is verified by a realtime PostPFA* V_x *with perfect completeness and error bound* $\frac{x}{2+x}$*, where* $x < 1$*.*

Proof. Let w_m be the m-th shortest member of `UPOWER` ($m > 0$) and let $w = 0^n$ be the given input string for $n > 1$. (If the input is empty string or 0, then it is rejected deterministically.)

The verifier expects the certificate to be composed by $t > 0$ block(s) followed by symbol \$, and each block has form of 0^+1 except the last one which is 1. The verifier also never checks a new symbol on the certificate after reading a \$ symbol. Let c_w be the given certificate in this format:

$$c_w = u_1 \cdots u_{t-1} u_t \$\$^*,$$

where for each $j \in \{1, \ldots, t-1\}$, $u_j \in \{0^+1\}$, and $u_t = 1$. Any other certificate is detected deterministically, and then, the input is rejected. Let $u_w = u_1 \cdots u_{t-1} u_t \$$ and $l_j = |u_j|$.

The verifier checks that (1) l_j is twice of l_{j+1} for each $j \in \{1, \ldots, t-2\}$, (2) each block except the last one contains at least one 0 symbol, (3) the last block is 1, and (4) $|w| = |u_w|$. Notice that these conditions are satisfied only for members: The expected certificate for w_m is

$$c_{w_m} = \underbrace{0^{2^{m-1}-1}1}_{1st\ block}\underbrace{0^{2^{m-2}-1}1}_{2nd\ block}\cdots 1\underbrace{0001}_{\cdots}\underbrace{01}_{\cdots}\underbrace{1}_{m\text{-}th\ block}\ \$\$^*$$

and the length of all blocks and a single \$ symbol is

$$2^{m-1} + 2^{m-2} + \cdots + 2^1 + 2^0 + 1 = 2^m.$$

In other words, $l_1 = \frac{|w|}{2}$, $l_2 = \frac{|w|}{4}$, $\ldots$, $l_m = \frac{|w|}{2^m}$.

At the beginning of the computation, V_x splits the computation into two paths with equal probabilities, called the accepting path and the main path. In the accepting path, the computation ends in s_{acc} with probability $\frac{x}{2^t}$ and in some non-postselecting state with the remaining probability. Since there

are t blocks, it is easy to obtain this probability. This is the path in which V_x enters s_{acc}. Therefore, $a(w) = \frac{x}{2^{t+1}}$ (the accepting path is selected with probability $\frac{1}{2}$).

During reading the input and the certificate, the main path checks (1) whether $|w| = |u_w|$, (2) each block of the certificate except the last one contains at least one 0 symbol, and (3) the last block is 1. If one of checks fails, the computation ends in state s_{rej}. The main path also creates subpaths for checking whether

$$l_1 = \frac{|w|}{2}, l_2 = \frac{l_1}{2}, \ldots, l_{m-1} = \frac{l_{m-2}}{2}.$$

After the main path starts to read a block starting with 0 symbol, it creates a subpath with half probability and stays in the main path with remaining probability. Thus, the main path reaches the right end-marker with probability $\frac{1}{2^t}$. On the other hand, the j-th subpath is created with probability $\frac{1}{2^{j+1}}$, where $1 \leq j \leq t-1$.

The first subpath tries to read $2l_1$ symbols from the input. If there are exactly $2l_1$ symbols, i.e., $2l_1 = |w|$, then the test is successful and the computation is terminated in a non-postselecting state. Otherwise, the test is failed and the computation is terminated in state s_{rej}.

The second path is created after reading l_1 symbols from the input. Then, the second subpath also tries to read $2l_2$ symbols from the input. If there are exactly $2l_2$ symbols, i.e., $l_1 + 2l_2 = |w|$, then the test is successful and the computation is terminated in an non-postselecting state. Otherwise, the test is failed and the computation is terminated in state s_{rej}.

The other subpaths behave exactly in the same way. The last ($(t-1)$-th) subpath checks whether $l_1 + l_2 + \cdots + l_{t-2} + 2l_{t-1} = |w|$. If all previous tests are successful, then $l_{t-1} = \frac{l_{t-2}}{2} = \frac{|w|}{2^{t-1}}$.

It is clear that if w is a member, say w_m, and V_x reads w_m and c_{w_m}, then $a(w) = \frac{x}{2^{m+1}}$. On the other hand, neither the main path nor any subpath enters state s_{rej} with some non-zero probability. Therefore, any member is accepted with probability 1.

If w is not a member, then one of the checks done by the main path and the subpaths is failed, and hence V_x enters s_{rej} with non-zero probability. The probability of being in s_{rej} at the end, i.e., $r(w)$, is at least $\frac{1}{2^t}$. Thus,

$$\frac{r(w)}{a(w)} \geq \frac{\frac{1}{2^t}}{\frac{x}{2^{t+1}}} = \frac{2}{x}.$$

Therefore, any non-member is rejected with probability at least $\frac{2}{2+x}$. □

In the above proof, the verifier can also check deterministically whether the number of blocks is a multiple of k or not for some $k > 1$. Thus, we can easily conclude the following result.

Corollary 9. $\texttt{UPOWER}_{\texttt{k}} = \{0^{2^{km}} \mid m > 0\}$ *is verified by a realtime PostPFA with perfect completeness and any error bound.*

Theorem 19. `USQUARE` *is verified by a realtime PostPFA* V_x *with perfect completeness and error bound* $\frac{x}{x+1}$*, where* $x < 1$*.*

Proof. The proof is very similar to the above proof. Let w_m be the m-th shortest member of `USQUARE` ($m > 1$). Let $w = 0^n$ be the given input for $n > 3$. (The decisions on the shorter strings are given deterministically.) The verifier expects to obtain a certificate composed by t blocks:

$$c_w = a^{m_1} b^{m_2} a^{m_3} \cdots d^{m_t} \$\$^*,$$

where d is a (b) if t is odd (even). Let $u_w = a^{m_1} b^{m_2} a^{m_3} \cdots d^{m_t}\$$. The verifier never reads a new symbol after reading u_w on the certificate.

The verifier checks the following equalities:

$$m_1 = m_2 = \cdots = m_t = t + 1$$

and

$$|w| = m_1 + m_2 + \cdots + m_t + (t + 1).$$

If we substitute m_1 with m in the above equalities, then we obtain that $|w| = (m-1)m + m = m^2$, and hence $w = w_m$.

At the beginning of the computation, V_x splits into the accepting path and the main path with equal probabilities, and, as a result of the accepting path, it always enters s_{acc} with probability $a(w) = \frac{x}{2^{t+1}}$.

In the following paths, if the comparison is successful, then the computation is terminated in a non-postselecting state, and, if it is not successful, then the computation is terminated in state s_{rej}. The main path checks the equality $|w| = m_1 + m_2 + \cdots + m_t + (t + 1)$.

For each $j \in \{1, \ldots, t\}$, the main path also creates a subpath with probability $\frac{1}{2}$ and remains in the main path with the remaining probability. The j-th subpath checks the equality

$$|w| = m_j + m_1 + \cdots + m_t,$$

where m_j is added twice.

If all comparisons in the subpaths are successful, then we have

$$m_1 = m_2 = \cdots = m_t = m$$

for some $m > 0$. Additionally, if the comparison in the main path is successful, then we obtain that $t = m-1$. Thus, $w = w_m$. Therefore, any member is accepted with probability 1 by help of the proof composed by $(m-1)$ blocks and the length of each block is m.

If w is not a member, then one of the comparisons will not be successful. (If all are successful, then, as described above, the certificate should have $(m-1)$ blocks of length m and the input has length m^2.) The minimum value of $r(w)$ is at least $\frac{1}{2^{t+1}}$, and hence

$$\frac{r(w)}{a(w)} \geq \frac{1}{x}.$$

Therefore, any non-member is rejected with probability at least $\frac{1}{x+1}$. □

Now we can proceed with the verification of uncountably many unary languages for realtime PostPFAs.

Theorem 20. *Realtime PostPFAs can verify uncountably many unary languages with error bound* $\frac{2}{5}$.

Proof. We obtain the result by designing a realtime PostPFA, say V_I, for the language

$$\texttt{UPOWER}_6(\texttt{I}) = \texttt{UPOWER64}(\texttt{I}) = \{0^n \mid n = 2^{6k},\ k > 0 \text{ and } k \in I\}$$

for $I \in \mathcal{I}$. Let $w = 0^n$ be the given input for $n > 64$. (The decisions on the shorter strings are given deterministically.)

The verifier V_I expects a certificate, say c_w, having two tracks containing the certificates c'_w and c''_w as

$$c_w = \begin{array}{|c|c|c|c|c|c|} \hline c'_w[1] & c'_w[2] & c'_w[3] & \cdots & c'_w[j] & \cdots \\ \hline c''_w[1] & c''_w[2] & c''_w[3] & \cdots & c''_w[j] & \cdots \\ \hline \end{array}.$$

The certificate c'_w is to verify that $n = 64^k$ for some $k > 0$ and c''_w is to verify that $n = m^2$ for some $m > 0$. Here we use the certificates given for $\texttt{UPOWER}_6$ and USQUARE. Notice that if $n = 64^k$, then $m = 8^k$.

At the beginning of the computation, V_I splits into three paths with equal probabilities. In the first path, V_I executes the realtime PostPFA, say M_1, designed for language $\texttt{UPOWER}_6$ with a single modification. Let t_1 be the

number of blocks in c'_w. Remember that $2t_1 \leq |w|$. Then, the minimum probability of entering s_{rej} in this path is 2^{-t_1} if $w \notin \texttt{UPOWER}_6$. On the other hand, we modify the probability of entering s_{acc} in this path to $a_1(w) = 2^{-|w|-5}$ (originally it depends on the parameter x and the number of blocks: $(x \cdot 2^{-t_1-1})$).

In the second path, V_I executes the realtime PostPFA, say M_2, designed for language USQUARE with a single modification. Let t_2 be the number of blocks in c''_w. Then, the minimum probability of entering s_{rej} in this path is 2^{-t_2-1} if $w \notin \texttt{USQUARE}$. On the other hand, we modify the probability of entering s_{acc} in this path to $a_2(w) = 2^{-|w|-5}$ (originally it depends on the parameter x and the number of blocks: $(x \cdot 2^{-t_2-1})$).

In the third path, V_I assumes that c''_w has t_2 blocks and each block has length t_2+1. Then, V_I tosses $\texttt{coin}_I$ for each input symbol, and then it moves on the certificate c''_w by one symbol for each outcome "heads". If $w \in \texttt{UPOWER}_6$ and c'_w and c''_w are as expected, then V_I tosses $\texttt{coin}_I$ 64^k times and meanwhile uses c''_w to calculate the bit x_k correctly: x_k is set to 0 at the beginning, and then, after each $4 \cdot 8^k$ heads, the value of x_k is set to $1 - x_k$. As described in the proof of Theorem 12, if c''_w is a valid certificate, x_k is calculated correctly in this way with probability $\frac{3}{4}$. In this path, if x_k is guessed as 1 (resp., 0), then V_I enters state s_{acc} (resp., s_{rej}) with probability $2^{-|w|-2}$.

If $w \in \texttt{UPOWER}_6(\texttt{I})$, then both certificates are as expected and x_k is calculated correctly in the third path. Since M_1 and M_2 do not enter state s_{rej} and V_I enters state s_{acc} with probability three times of the probability of entering s_{rej} in the third path, w is accepted with probability greater than $\frac{3}{4}$.

If $w \notin \texttt{UPOWER}_6(\texttt{I})$, then we have two cases. Case 1: Both certificates are as expected ($w \in \texttt{UPOWER}_6$), and hence M_1 and M_2 enter state s_{acc} with the probability $2^{-|w|-5}$. Then, $x_k = 0$ is calculated correctly in the third path and the probabilities of entering states s_{acc} and s_{rej} can be

$$2^{-|w|-2} \cdot \frac{1}{4} \text{ and } 2^{-|w|-2} \cdot \frac{3}{4},$$

respectively, in the worst case. Thus, the overall probability of being in state s_{acc} is

$$a(w) = \frac{1}{3} \cdot 2^{-|w|-5} + \frac{1}{3} \cdot 2^{-|w|-5} + \frac{1}{3} \cdot 2^{-|w|-4} = \frac{1}{3} \cdot 2^{-|w|-3}.$$

On the other hand, the probability of being in state s_{rej} is $2^{-|w|-4}$ ($\frac{1}{3} \cdot 2^{-|w|-2} \cdot \frac{3}{4}$) and it is $\frac{3}{2}$ times of $a(w)$. Therefore, w is rejected by V_I with probability $\frac{3}{5}$.

Case 2: In this case, $w \notin \texttt{UPOWER}_6$. Then, M_1 enters state s_{rej} with probability 2^{-t_1}, which is definitely much bigger than the probability of being in state s_{acc} at the end of the computation. Therefore, w is rejected by V_I with high probability. □

The realtime PostPFA given above can be converted into a restarting realtime PFA as described in the Chapter 1, but the expected running time will be exponential.

Corollary 10. *Restarting realtime PFAs can verify uncountably many unary languages with bounded error in exponential expected time.*

On the other hand, on binary languages, we obtain the same result in linear expected time. For this purpose we use a modification of DIMA2:

$$\texttt{DIMA2}_l = \{0^{2^0}10^{2^1}10^{2^2}1\cdots10^{2^{3k+l-1}}11(0^{2^{3k+l}}1)^{2^{3k}} \mid k > 0\}.$$

Theorem 21. *Restarting realtime PFAs can verify uncountably many languages in linear expected time with any error bound.*

Proof. Let w_k be the k-th shortest member of $\texttt{DIMA2}_l$ for $k > 0$ and for some even integer $l > 0$. For any $I \in \mathcal{I}$, we define the following language:

$$\texttt{DIMA2}_l(I) = \{w_k \mid k > 0 \text{ and } k \in I\}.$$

Let w be the given input of the form

$$w = 0^{t_1}10^{t_2}1\cdots10^{t_m}110^{t'_1}10^{t'_2}1\cdots10^{t'_{m'}}1,$$

where $t_1 = 1$, m and m' are positive integers, $m - l$ is divisible by 3, and $t_i, t'_j > 0$ for $1 \le i \le m$ and $1 \le j \le m'$. (Otherwise, the input is rejected deterministically.)

The verifier splits the computation into four paths with equal probabilities. The protocol is one-way, and the verifier asks the prover to send $y = w$ in each path. Let $r > 8$ be the integer determining the error bound.

In the first path, the verifier checks whether $w = y$. If not, the input is rejected. Otherwise, the input is accepted with probability $\frac{1}{r}$ and the computation is restarted with probability $\frac{r-1}{r}$.

In the following part, we assume that $y = w$. In the second path, the verifier reads w and y in parallel and makes the following comparisons:

- for each $i \in \{1, \ldots, m-1\}$, whether $2t_i = t_{i+1}$,
- whether $2t_m = t'_1$, and,

- for each $j \in \{1, \ldots, m' - 1\}$, whether $t'_j = t'_{j+1}$.

If one of the comparisons is not successful, then the input is rejected. If all of them are successful, then we know that the input is of the form

$$w = 0^{2^0}10^{2^1}10^{2^2}1 \cdots 10^{2^{3k+l-1}}11(0^{2^{3k+l}}1)^{m'},$$

and the input is accepted with probability $\frac{1}{r}$ and the computation is restarted with probability $\frac{r-1}{r}$.

In the third path, the verifier checks whether $m' = 2^{3k}$ or not, i.e., the verifier compares $\frac{t'_1}{2^l}$ from y with the number of 1's appearing after "11" in w. If not, the input is rejected. Otherwise, it is accepted with probability $\frac{1}{r}$ and the computation is restarted with probability $\frac{r-1}{r}$.

In the fourth path the verifier assumes that

$$w = w_k = y = 0^{2^0}10^{2^1}10^{2^2}1 \cdots 10^{2^{3k+l-1}}11(0^{2^{3k+l}}1)^{2^{3k}}.$$

The verifier reads w and y in parallel and places both reading heads on "11". Then, the verifier reads w to toss $\mathtt{coin}_I$ $64^k \cdot 2^l$ times. For counting the heads, the verifier uses the part of y after symbols "11" as a read-only counter, which is composed of 8^k blocks of 0's and length of each block is $8^k \cdot 2^l$. Let h be the total number of heads:

$$h = i \cdot 8^{k+1} \cdot 2^l + j \cdot 8^k \cdot 2^l + q = (8i + j)8^k \cdot 2^l + q,$$

where $i \geq 0$, $j \in \{0, \ldots, 7\}$, and $q < 8^k \cdot 2^l$. Due to Lemma 2, x_k is the $(3k + l + 3)$-th digit of $bin(h)$ with probability at least $1 - \frac{1}{4 \cdot 2^l}$. The analysis of the value j is similar to one presented in the proof of Theorem 16. If x_k is guessed as 0, the input is rejected. If x_k is guessed as 1, the input is accepted with probability $\frac{1}{r}$ and the computation is restarted with probability $\frac{r-1}{r}$.

Let $a(w)$ and $r(w)$ be the probabilities to accept and reject the input in a single run, respectively.

If $w \in \mathtt{DIMA2}_l(I)$, the input can only be rejected in the fourth path with probability not exceeding $\frac{1}{4 \cdot 2^l}$. Therefore, $r(w) \leq \frac{1}{16 \cdot 2^l}$, while $a(w) \geq \frac{3}{4 \cdot r}$, and hence

$$\frac{a(w)}{r(w)} \geq \frac{3 \cdot 16 \cdot 2^l}{4 \cdot r} = \frac{12 \cdot 2^l}{r}.$$

This means that if $w \in \mathtt{DIMA2}_l(I)$, then the total probability to accept the input is at least

$$\frac{\frac{12 \cdot 2^l}{r}}{1 + \frac{12 \cdot 2^l}{r}} = \frac{1}{\frac{r}{12 \cdot 2^l} + 1} = \frac{1}{\frac{r + 12 \cdot 2^l}{12 \cdot 2^l}} = \frac{12 \cdot 2^l}{r + 12 \cdot 2^l}.$$

If $w \notin \texttt{DIMA2}_l$, the input is rejected in at least one of four paths with probability 1 in a single run. Therefore, $r(w) \geq \frac{1}{4}$ and $a(w) \leq \frac{3}{4 \cdot r}$, and hence

$$\frac{r(w)}{a(w)} \geq \frac{4 \cdot r}{3 \cdot 4} = \frac{r}{3}.$$

This means that if $w \notin \texttt{DIMA2}_l$, then the total probability to reject the input is at least

$$\frac{\frac{r}{3}}{1 + \frac{r}{3}} = \frac{1}{\frac{3}{r} + 1} = \frac{1}{\frac{3+r}{r}} = \frac{r}{3+r}.$$

If $w \in \texttt{DIMA2}_l$ and $w \notin \texttt{DIMA2}_l(I)$, the input is rejected with probability at least $1 - \frac{1}{4 \cdot 2^l}$ in the fourth path in a single run. Therefore, $r(w) \geq \frac{4 \cdot 2^l - 1}{4 \cdot 4 \cdot 2^l}$ and $a(w) \leq \frac{1}{r}$, and hence

$$\frac{r(w)}{a(w)} \geq \frac{(4 \cdot 2^l - 1) \cdot r}{16 \cdot 2^l} \geq \frac{2 \cdot 2^l \cdot r}{16 \cdot 2^l} = \frac{r}{8}.$$

This means that if $w \in \texttt{DIMA2}_l$ and $w \notin \texttt{DIMA2}_l(I)$, then the total probability to reject the input is at least

$$\frac{\frac{r}{8}}{1 + \frac{r}{8}} = \frac{1}{\frac{8}{r} + 1} = \frac{1}{\frac{8+r}{r}} = \frac{r}{8+r}.$$

Thus, the error bound ϵ depends on r and l,

$$\epsilon = \max\left(1 - \frac{r}{8+r}, 1 - \frac{12 \cdot 2^l}{12 \cdot 2^l + r}\right) = \left(\frac{8}{8+r}, \frac{r}{12 \cdot 2^l + r}\right).$$

We remark that if $\frac{8}{8+r} > \frac{r}{12 \cdot 2^l + r}$, then $\epsilon = \frac{8}{8+r}$. Therefore, if $96 \cdot 2^l > r^2$, then $\epsilon \to 0$ when $r \to \infty$. Therefore, for any $\epsilon_0 > 0$ there exist large enough r and l such that $\epsilon \leq \epsilon_0$ and for any $I \in \mathcal{I}$ the machine can verify $\texttt{DIMA2}_l(I)$ with error bound ϵ.

Each run of the computation finishes with probability at least $\frac{1}{r}$, therefore, the expected running time is linear. □

4.2 Realtime probabilistic Turing machines

In this section, we focus on PTMs that operate in realtime reading mode and we present our space-bound results for unary and binary languages.

In [50], it was shown that realtime 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 22. *Realtime unary PTMs can recognize uncountably many languages in $O(\log n)$ space with any error bound.*

Proof. We start with defining a sequence $(k_i)_{i\in\mathbb{N}}$ such that $k_1 = 64 \cdot 28$ and $k_i = k_{i-1} + 64^i \cdot (18i + 10)$ for $i > 1$. Let $\texttt{ULOG} = \{a^{k_i}\}$ be an unary language corresponding to the sequence $(k_i)_{i\in\mathbb{N}}$. Since it is not a periodic language, ULOG is nonregular. Then, for any $I \in \mathcal{I}$, we define the following language:

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

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

The PTM M_I uses $\mathfrak{coin}_I$. The aim of M_I is iteratively finding the values of $x_1, x_2, \ldots$ of p_I 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, M_I uses two binary counters on the work tape and it uses an iterative algorithm.

At the beginning, the iteration number is one, say $i = 1$. The machine initializes the work tape as

$$\#000000\#000000\#$$

by reading $9i + 3 + 2$ (=14) symbols from the input (after 14-th symbol the work tape head is placed on the 15-th tape square from the left). We name the separator symbols #'s for the counters as the first, second, and third ones from left to right. Moreover, the first (second) counter is kept between the last (first) two #'s as shown below:

$$\underbrace{\#}_{\substack{\text{1st}\\ \text{separator}}} \quad \underbrace{000000}_{\substack{\text{2nd counter}\\ \text{of length 3i+3}}} \quad \underbrace{\#}_{\substack{\text{2nd}\\ \text{separator}}} \quad \underbrace{000000}_{\substack{\text{1st counter}\\ \text{of length 6i}}} \quad \underbrace{\#}_{\substack{\text{3rd}\\ \text{separator}}} .$$

By using the first counter, the machine counts up to 64^i and meanwhile also tosses $\mathfrak{coin}_I$ 64^i times. By using the second counter, it counts the number of heads (modulo $2^{3i+3}(= 8^{i+1})$). The value of each counter can be easily increased by 1 when the work tape head passes on the counters from right to left once. Thus, when the work tape head is on the third #, it goes to the first #, and meanwhile increases the value of the first counter by

1, then tosses $\mathfrak{coin}_I$, and, if it is heads, it also increases the value of the second counter. After tossing $\mathfrak{coin}_I$ 64^i times, the machine uses the leftmost value of the second counter as its answer for x_i. (Remember the Method 2 of application of Lemma 1.) Once this decision is read from the work tape and immediately after the work tape head is placed on the first #, the current iteration is finished. If (1) an iteration is finished, (2) there are no more symbols remaining to be read from the input, and (3) 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 unless the input is finished.

Since the input is read in realtime mode, the number of 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 M_I.

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

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

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

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

by shifting the second and third #'s to 3 and 9 amounts of cells to the right (after initialization the work head is placed on the third #).

After the initialization of the counters, the work 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 $\mathfrak{coin}_I$ is flipped, and then the second counter is increased by 1 if the result is heads. When all digits of the first counter are 1, which means the number of passes reaches $64^i - 1$, the work tape head makes its last pass from the third # to the first #. During the last pass, M_I flips the $\mathfrak{coin}_I$ 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, M_I makes 64^i passes starting from the first #. Therefore, 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, M_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, due to Lemma 1, we can conclude that M_I recognizes language $\texttt{ULOG}(\texttt{I})$ with error bound $\frac{1}{4}$.

We can change the algorithm of M_I to perform the computation of the values of $x_1, x_2, \ldots$ of p_I multiple times. For this purpose, we define a new sequence $(k'_i)_{i\in\mathbb{N}}$ such that $k'_i = m \cdot k_i$ for $i \geq 1$, and a new corresponding unary language $\texttt{ULOG}_m = \{a^{k'_i}\}$, where $m > 1$. For any $I \in \mathcal{I}$ and $m > 1$, we can define the following language:

$$\texttt{ULOG}_\texttt{m}(\texttt{I}) = \{a^{k'_i} \mid a^{k_i} \in \texttt{ULOG}_m \text{ for } i \geq 1 \text{ and } i \in I\}.$$

To recognize $\texttt{ULOG}_\texttt{m}(\texttt{I})$, we make the following modification in the algorithm of M_I: For each $i > 0$, M_I repeats i-th iteration m times, and, at the end of each iteration, it remembers the calculated value x_i. If the input finishes before the end of m repetitions of i-th iteration, the input is rejected. If the input is finished after m-th repetition, M_I picks the most frequent value of x_i to decide whether the input should be accepted. If the input is not finished after the m-th repetition of the i-th iteration, M_I continues with $(i + 1)$-th iteration. The machine can recognize $\texttt{ULOG}_\texttt{m}(\texttt{I})$ with arbitrarily small error bound for sufficiently large m. □

Now, we focus on non-unary alphabets and establish our result for double logarithmic space. For this purpose, we use the Fact 1 given by Freivalds in [19]. This fact was derived from the Prime Number Theorem, which also takes place in our result.

Fact 5 (cf. [9])**.** *Denote by $\pi(x)$ the number of primes not exceeding x. The Prime Number Theorem states that* $\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$.

The language LOGLOG is composed by the strings

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

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

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 [50]: For each original "wait" move on a symbol from Σ, the machine expects to read symbol σ. If it reads something else or there are no more input symbols, 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 the realtime machine are no more than these of the one-way machine.

A DTM follows a single path during its computation, and hence 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. If PTM guarantees that each computation path uses the same amount of time steps on the same input, we can apply a similar idea for one-way PTM. We use this idea in the last part of the following proof.

Theorem 23. *Realtime PTMs can recognize uncountably many languages in* $O(\log \log n)$ *space with any error bound.*

Proof. By modifying the one-way algorithm given in [19], we present a PTM, say $M_{c,I}$, shortly M, for language $\texttt{LOGLOG}(\texttt{I})$ for $I \in \mathcal{I}$ and for a specific c that determines the error bound.

Each member of $\texttt{LOGLOG}(\texttt{I})$ has parts $bin(i)$ at least up to $bin(2^{63})$. The PTM M deterministically checks the input up to $bin(2^{63})$ and prepares the work tape. After reading $bin(2^{63})$, M begins to process $bin(i)$'s as described below.

PTM M keeps values $m = |bin(i)|$ and $m_0 = |bin(i-1)|$ on work tape. For each i, after reading $bin(i)$ M checks deterministically: If $m = m_0$ or ($m = m_0 + 1$ and $bin(i-1)$ contained only ones), then M continues. Otherwise, M rejects the input.

Before reading each $bin(i)$, M generates a prime number. First, M generates number r 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 $2^{|m|\cdot c} - 1$, except r. Each candidate natural number is denoted by d below. If r is not divisible by any of these d's, then M concludes that r is a prime number. Notice that the number of d's does not depend on r, and hence for any candidate prime number, the primality test procedure takes the same number of steps.

To store and work with prime number, M uses a register on the work tape that has $|m| \cdot c \cdot 2$ bits. M stores $r = r_1 r_2 r_3 \cdots$ and additional auxiliary number $q = q_1 q_2 q_3 \cdots$ as $r_1 q_1 r_2 q_2 r_3 q_3 \cdots$. M uses in total four different such registers, further in text denoted by p_1, p_2, p_3 and p_4. For $bin(i)$, M uses register $p_{(3-2\cdot(i \mod 2))}$. To check the divisibility of a number by d, M uses a register on the work tape that has $|m| \cdot c \cdot 2$ bits to keep number d and additional auxiliary number h. If $d = d_1 d_2 d_3 \cdots$ and $h = h_1 h_2 h_3 \cdots$, then the register keeps them as $d_1 h_1 d_2 h_2 d_3 h_3 \cdots$. M uses one more register on the work tape (further in text u) to keep track of total number of subtractions performed while checking the divisibility of r by d. It has $|m| \cdot c$ bits.

To begin the check of divisibility of r by d, (1) the value of r is copied on q, (2) the value of d is copied on h, and (3) the counter u is initialized with zeros. Then, $2^{|m|\cdot c}$ iterations are performed. In each iteration, the values

of q and h are decreased by 1, the value of u is increased by 1. If only h reaches zero, d is again copied on 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 (except the case when h reaches zero for the first time — in this case $d = r$), otherwise, r is not divisible by d. After that, M continues to perform the iterations but without changing q and checks of value of q until the value of u reaches $2^{|m|\cdot c}$. Then, M repeats the procedure for the next d.

PTM M uses additional register on the work tape (further in text v) that counts the number of attempts to generate a prime number. It is initialized with zeros and is increased by one after each try. If M finds a prime number before v reaches $2^{|m|\cdot c}$, M continues performing the algorithm until v 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 M fails to generate a prime number, M continues with the last generated r. Thus, M performs the same number of steps for trying to generate a prime number. For any $bin(i)$, M performs exactly $2^{|m|\cdot c}$ such operations.

Next, the machine copies r into $p_{(4-2\cdot(i \mod 2))}$ 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.

After that, M reads $bin(i)$ and 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 the left by one position, and the machine puts value i_j in rightmost bit. After this operation, M performs one pass through registers, and if $q \geq r$, then r is subtracted from q during this one pass, therefore, each iteration for i_j is performed in equal number of steps. The machine performs the calculation while reading $bin(i)$ for the registers $p_{(2-i \mod 2)}$ and $p_{(3+i \mod 2)}$.

After these, the machine compares the values of the modules from the registers $p_{(1+2\cdot(i \mod 2))}$ and $p_{(2+2\cdot(i \mod 2))}$. 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 $(bin(i-1) \mod r) + 1 \neq bin(i) \mod r$, M rejects the input. Otherwise, the computation continues.

After reading "4", M checks whether $m = 64^k$ for some integer $k > 0$. If $m \neq 64^k$, then the input is rejected, otherwise, m is written on the tape as

$$1(000000)^k.$$

Then, M tosses $\mathtt{coin}_I$ 64^k times and meanwhile calculates the number of heads $\mod (8 \cdot 8^k)$, say t. If after all coin tosses, the leftmost bit of t is 1,

then the input is accepted, otherwise it is rejected.

Due to Fact 5, we can conclude that the probability of picking a prime number of $|m| \cdot c$ bits in one attempt is $\theta(\frac{1}{|m| \cdot c})$. Therefore, the probability not to generate a prime number of $|m| \cdot c$ random bits in $2^{|m| \cdot c}$ attempts does not exceed $(1 - \frac{1}{|m| \cdot c})^{2^{|m| \cdot c}}$. Note that $\lim_{m \to \infty} (1 - \frac{1}{m})^m = \frac{1}{e}$, 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 consistency 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 1. Since M can fail to generate a prime number, this probability is increased to at most

$$\epsilon + \epsilon_0 - \epsilon \cdot \epsilon_0 = \epsilon_1.$$

If the input belongs to $\texttt{LOGLOG}(\texttt{I})$, M is guaranteed not to reject the input before reaching "4" on the input tape. If at least one pair $bin(i)$ and $bin(i+1)$ is inconsistent, then M rejects the input right after checking this pair with probability at least $1 - \epsilon_1$. Due to Lemma 1 the membership of $k \in I$ for $\texttt{LOGLOG}(\texttt{I})$ is computed correctly with probability at least $\frac{3}{4}$. By repeating the procedure of coin tosses l times, this probability can be increased to $1 - \epsilon_l$, which can be arbitrarily close to 1. Therefore, the language $\texttt{LOGLOG}(\texttt{I})$ is recognized with probability at least $(1 - \epsilon_1) \cdot (1 - \epsilon_l)$, which can be arbitrarily close to 1 by picking suitable c and l.

We can execute M in realtime by using aforementioned technique borrowed from [50]. Let $\texttt{LOGLOG}(\texttt{I})'$ be the language recognized by M in realtime. Then, the language $\texttt{LOGLOG}(\texttt{I})'$ differs from the language $\texttt{LOGLOG}(\texttt{I})$ with the presence of symbols "3": Instead of "wait" move on "0", "1", "2" or "4", M expects to read one symbol of "3". If M fails to read a symbol "3" when it is expected, the input is rejected. It is easy to see that there is a bijection between $\texttt{LOGLOG}(\texttt{I})$ and $\texttt{LOGLOG}(\texttt{I})'$.

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

In [19] Freivalds has proven that only regular languages can be recog-

nized with one-way PTM in $o(\log\log n)$ space and with probability $p > \frac{1}{2}$. Therefore, the presented space bound is tight.

4.3 Probabilistic counter automata

We begin this section with the recognition of uncountably many unary languages with two counters in polynomial time.

Theorem 24. *Unary 2P2CAs can recognize uncountably many languages in $O(\log n)$-space and $O(n \log n)$-time with any error bound.*

Proof. We remind the definition of the language $\texttt{AM75}'(I)$, which is considered in the proof of Theorem 6:

$$\texttt{AM75}'(\texttt{I}) = \{a^n \mid a^n \in \texttt{AM75}' \text{ and } \Sigma^*(\log_{64} F(n)) \in I\}, \text{ where}$$

$$\texttt{AM75}' = \{a^n \mid n > 0 \text{ and } F(n) \text{ is a power of } 64\} \text{ and } I \in \mathcal{I}.$$

Let M be the 2P2CA that recognizes $\texttt{AM75}'(\texttt{I})$ for some $I \in \mathcal{I}$. We represent the values of the first and second counters as c_1 and c_2, respectively, and the position of the head on the input tape as b, and $b = 0$ when the input head is on the left end-marker.

Let $w = a^n$ be the input for some $n > 0$. M first checks whether $w \in \texttt{AM75}'$ like in the proof of Theorem 6. M uses both counters to compute the value $F(n)$. In order to compute $F(n)$, M checks each $l = 2, 3, \ldots$ by reading w in sweeping mode to determine the lowest value l such that $n \mod l \neq 0$.

In order to check whether $n \mod l \neq 0$, M moves the input head on the left end-marker and sets $c_1 = l$, $c_2 = 0$. After that, M reads the input from the left to the right and for each a 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_2 is zero at the beginning, the first strategy is decreasing the value of c_1 and increasing the value of c_2. When M reads the right end-marker, if $c_1 = 0$ or $c_2 = 0$, then $n \mod l = 0$, and if the values of both counters are not equal to zero, then $n \mod l \neq 0$.

If $n \mod l = 0$, M sets $c_1 = l + 1$ and $c_2 = 0$ and proceeds with the next check, i.e., to determine whether $n \mod (l + 1) \neq 0$. If $n \mod l \neq 0$, then $F(n) = l$.

After computing $F(n)$, M sets $c_1 = l$ and $c_2 = 0$, and then places the input head on the left end-marker. Then, M checks whether $l = 64^k$ for some $k > 0$. To make this check, M performs the following steps:

- with the help of 2nd counter M divides the value c_1 by 64,

- M moves the input head to the next symbol on the right.

The steps are repeated until $c_1 = 1$. If M cannot divide c_1 by 64, then $l \neq 64^k$ and M rejects the input. If M successfully performs the steps and $c_1 = 1$, then $w \in \texttt{AM75}'$, $l = 64^k$ for some $k > 0$, and M continues with the following configuration: $c_1 = 1$, $c_2 = 0$, $b = k$.

After that, M performs the following steps in a loop:

- in a loop until c_1 reaches zero M decreases c_1 by 1 and increases c_2 by 64,
- then M swaps the values of c_2 and c_1,
- M moves the input head on the next symbol to the left.

The loop finishes when the input head reads the left end-marker. In this case $b = 0$, $c_1 = 64^k$, $c_2 = 0$.

After that, by using the value of the first counter, M sets $b = 64^k$, $c_2 = 64^k$, and $c_1 = 0$. Then, M performs the following loop: M decreases b by one, tosses the $\texttt{coin}_I$, and if the result is heads, the machine increases c_1 by one. Let h be the number of heads after 64^k tosses of $\texttt{coin}_I$. At the end of the loop, we have $c_1 = h$, $c_2 = 64^k$, and $b = 0$.

Then, M performs the following operations in a loop:

- M divides c_2 by 64 with the help of the position of input head (by using b as a counter),
- M divides c_1 by 8 with the help of b as a counter (the division is performed with rounding down).

The loop finishes when $c_2 = 1$. At this step $c_1 = \lfloor \frac{h}{8^k} \rfloor$. After that, M divides c_1 by 4 with the help of the second counter, and hence $c_1 = \lfloor \frac{h}{4 \cdot 8^k} \rfloor$. Let $t = c_1 = \lfloor \frac{h}{4 \cdot 8^k} \rfloor$. Then the following relation is true:

$$t \cdot 4 \cdot 8^k \leq h < (t+1) \cdot 4 \cdot 8^k.$$

To guess the value x_k, M checks whether t is odd or even by decreasing the value c_1 until zero. If t is odd, then x_k is guessed as 1 and M accepts the input, otherwise, x_k is guessed as 0 and M rejects the input.

Due to Lemma 1, M computes x_k with error bound $\frac{1}{4}$. Therefore, M recognizes $\texttt{AM75}'(I)$ with bounded error for any $I \in \mathcal{I}$. There are uncountably many different I, therefore, unary 2P2CAs can recognize uncountably many languages. It is known that $F(n) < c \cdot \log n$ for some constant c. The values of the counters do not exceed $F(n)$ during the computation, therefore, the

space complexity is $O(\log n)$. M performs each computational loop at most in $F(n)$ iterations, and each iteration takes at most n steps. Therefore, the total computational time does not exceed $O(n \log n)$.

M can repeat the whole algorithm multiple times and choose the most frequent value of x_k for decision to reduce the error bound arbitrarily close to zero. □

It is still open question whether 2PCAs can recognize uncountably many unary languages with bounded error. Therefore, we continue with a two-way PCAs on binary languages. We remark that any $s(n)$-space counter can be simulated by $\log(s(n))$-space work tape.

It is easy for a two-way PCA to check whether any specific part of the input has length of 64^k for some $k > 0$, and hence, they can easily toss a biased coin for 64^k times and then count the number of heads on the counter. However, it is not trivial to read some certain digits of the result on the counter, and so we use a clever trick here.

Theorem 25. *Linear-time (linear-space) 2PCAs can recognize uncountably many languages with any error bound.*

Proof. We start with the definition of the following language:

$$\texttt{DIMA} = \{0^{2^0}10^{2^1}10^{2^2}1\cdots10^{2^{3k+1}}110^{2^{3k+2}}110^{2^{3k+3}}1\cdots10^{2^{6k}} \mid k > 0\}.$$

We remark that each member is composed by $(6k+1)$ zero-blocks separated by single 1's except two special separators "11" that are used as the markers to indicate the $(3k+3)$-th block, the length of which is 2^{3k+2}.

The language DIMA can be recognized by a two-way deterministic counter automaton, say D. First, it checks that the input starts with a single 0 and ends with some 0's, all separators are 1's except two of them, which are "11" and consecutive, and the number of zero-blocks is $6k+1$ for some $k > 0$. For all these checks, D can use only its internal states. And then, by using its counter, it can check whether the length of each zero-block (except the first one) is double of the length of previous block. Similarly, it can check the equality of the number of zero-blocks before the first "11" and the number of zero-blocks after the first "11" plus 3, i.e., $3k+2$ versus $(3k-1)+3$. If one of these checks fails, then the input is rejected immediately. Otherwise, it is accepted. Notice that D can finish its computation in linear time and the counter value never exceeds the input length.

For any $I \in \mathcal{I}$, we define a new corresponding language:

$$\texttt{DIMA(I)} = \{w \in \{0,1\}^*10^m \mid m > 0, w \in \texttt{DIMA}, \text{ and } \Sigma^*(\log_{64} m) \in I\}.$$

For any such I, we can construct a two-way PCA recognizing $\texttt{DIMA}(\texttt{I})$, say M_I, as desired. The machine M_I checks whether any given input, say w, is in DIMA deterministically by using D. If the input is not rejected by D, we continue with a probabilistic procedure. Since the last zero-block has the length of $m = 64^k$, by reading this block M_I can toss 64^k biased coins that land on head with probability p_I ($\mathtt{coin}_I$). The number of heads are counted on the counter. Similarly to the proof of Theorem 6, the only remaining task is to determine the $(3k+3)$-th bit of the binary value of the counter, which is x_k. The bit x_k in $E[X] = p_I \cdot 64^k$ is followed by $3k+2$ bits.

The number of heads on the counter, say h, can be written as a binary number as follows:

$$h = \sum_{i=0}^{6k} a_i 2^i = a_{6k}2^{6k} + \cdots + a_{3k+2}2^{3k+2} + a_{3k+1}2^{3k+1} + \cdots + a_1 2 + a_0,$$

where each $a_i \in \{0,1\}$. We remark that x_k is corresponding to a_{3k+2}, i.e., $3k+2 = 6k-(3k-1)+1$. We can rewrite h as

$$h = b_1 2^{3k+3} + a_{3k+2}2^{3k+2} + b_0 = b_1 2^{3k+3} + h',$$

where b_0 and b_1 are integers, $b_0 < 2^{3k+2}$, and $h' = a_{3k+2}2^{3k+2} + b_0$.

After tossing-coin part, M_I moves its head to the second symbol of the first "11", and then the automaton enters a loop. In each iteration, the head moves to the next separator on the right by reading 2^{3k+2} 0's, and then comes back by reading the same amount of 0's. In each iteration, M_I tries to subtract 2^{3k+2} twice (i.e., 2^{3k+3}, the approach is similar to the Method 2 of application of Lemma 1).

If $h' = 0$, then M_I hits to the zero value on the counter when the head is at the starting position of the loop. This means $a_{3k+2} = x_k = 0$, and hence the input is rejected by the automaton M_I. If $h' \neq 0$, then M_I hits to the zero value on the counter, say in the j-th iteration $j = 0, 1, \ldots$, when the head is not at the starting position of the loop. The value of the counter is h' before starting the j-th iteration and there are two cases, $x_k = 1$ or $x_k = 0$. If $x_k = 1$, M_I hits to the zero value on the counter only after reading the first 2^{3k+2} 0's. In this case, the input is accepted. Otherwise, M_I hits to the zero value on the counter before finishing to read the first 2^{3k+2} 0's. Then, the input is rejected.

It is clear that the value of the counter never exceeds the length of the input. Moreover, both deterministic and probabilistic parts finish in linear time. By repeating the procedure of coin tosses and calculation of the value of x_k, the success probability is increased arbitrarily close to 1. □

We can also obtain polynomial-time result for sweeping PCAs.

Theorem 26. *Linear-space sweeping PCAs can recognize uncountably many languages in subquadratic time with any error bound.*

Proof. We modify the algorithms given in the proof of Theorem 25. Notice that the algorithms given there run in linear time. Here the algorithms run in super-linear time but still subquadratic. First, we show how to deterministically recognize the language DIMA in sweeping reading mode, i.e.,

$$\texttt{DIMA} = \{0^{2^0}10^{2^1}10^{2^2}1\cdots 10^{2^{3k+1}}110^{2^{3k+2}}110^{2^{3k+3}}1\cdots 10^{2^{6k}} \mid k > 0\}.$$

With one pass (reading the input from the left end-marker to the right end-marker), the input is checked without using counter whether having the following form

$$01(0^+1)^+110^+11(0^+1)^+0^+$$

and the number of 0-blocks are $6k+1$ for some $k > 0$. Moreover, for a member, the number of 0-blocks before the first "11" is $3k+2$ and the number of 0-blocks after the second "11" is $3k-2$. Therefore, by using the counter, we can check that the number of 0-blocks before the first "11" is 4 more than the number of 0-blocks after the second "11". If any of these checks fails, then the input is immediately rejected.

In the second pass (reading input from the right end-marker to the left end-marker), it is checked that, for each $0 < i \leq 3k$, $(2i+1)$-th 0-block has twice more zeros than $(2i)$-th 0-block.

In the third pass (reading input from the left end-marker to the right end-marker), it is checked that, for each $0 < i \leq 3k$, $(2i-1)$-th 0-block has twice less zeros than $(2i)$-th 0-block.

Thus, in three passes, DIMA can be recognized by a sweeping PCA.

Then, as in the proof of Theorem 25, for any $I \in \mathcal{I}$, we consider the language:

$$\texttt{DIMA(I)} = \{w \in \{0,1\}^*10^m \mid m > 0, w \in \texttt{DIMA}, \text{ and } \Sigma^*(\log_{64} m) \in I\}.$$

If the given input is in DIMA, then we continue with the probabilistic procedure. (Otherwise, the input is rejected.) We perform the same walk as in the proof of Theorem 25, but due to sweeping reading mode, each walk can be done from one end-marker to the other end-marker. But the presence of symbols "11" allows us to follow the same procedure only with slowdown. The running time is

$$O(2^{3k})O(2^{6k}) = O(2^{9k})$$

and it is super-linear and subquadratic in the length of input. To be more precise, the running time is $O(n\sqrt{n})$ (where n is the length of the input). By repeating the procedure of coin tosses and calculation of the value of x_k, the success probability is increased arbitrarily close to 1. □

Now we restrict the input head even more and continue with realtime counter automata and begin with one counter.

We use the following nonregular binary language, a modified version of DIMA:

$$\texttt{DIMA3}_l = \{0^{2^0}10^{2^1}10^{2^2}1\cdots10^{2^{6k+l-2}}110^{2^{6k+l-1}}11^{2^{6k+l}}(0^{2^{3k+l}-1}1)^{2^{3k}} \mid k > 0\},$$

where $l \geq 0$ is an even constant that influences the error bound.

Theorem 27. *Realtime probabilistic automata with one counter can recognize uncountably many languages with error bound* $\frac{4}{9} + \frac{1}{20\cdot 2^l}$ *for any even integer* $l \geq 0$.

Proof. Let w_k be the k-th shortest member of $\texttt{DIMA3}_l$ for $k > 0$. For any $I \in \mathcal{I}$, we define the following language:

$$\texttt{DIMA3}_l(I) = \{w_k \mid k > 0 \text{ and } k \in I\}.$$

Now we proceed with the recognition of $\texttt{DIMA3}_l(I)$ for any $I \in \mathcal{I}$. Let M be the PCA and w be the given input of the form

$$w = 0^{t_1}10^{t_2}1\cdots10^{t_{m-1}}110^{t_m}11^{t'_0}0^{t'_1}10^{t'_2}1\cdots10^{t'_{m'}}1,$$

where $t_1 = 1$, m and m' are positive integers, $m - l$ is divisible by 6, and $t_i, t'_j > 0$ for $1 \leq i \leq m$ and $0 \leq j \leq m'$. (Otherwise, the input is rejected deterministically.)

M splits the computation into five paths with equal probabilities. In the first path, with the help of the counter, M makes the following comparisons:

- for each $i \in \{1, \ldots, \frac{m}{2}\}$, whether $2t_{2i-1} = t_{2i}$,
- for each $j \in \{1, \ldots, \frac{m'}{2}\}$, whether $t'_{2j-1} = t'_{2j}$.

In the second path, with the help of the counter, M makes the following comparisons:

- for each $i \in \{1, \ldots, \frac{m}{2} - 1\}$, whether $2t_{2i} = t_{2i+1}$,
- whether $2t_m = t'_0$ (this also helps to set the counter to 0 for the upcoming comparisons),

- for each $j \in \{1, \ldots, \frac{m'}{2} - 1\}$, whether $t'_{2j} = t'_{2j+1}$.

In the third path, M checks whether $1 + \sum_{i=1}^{m} t_i = m' + \sum_{j=1}^{m'} t'_j$. In the fourth path M checks, whether $\frac{t'_1+1}{2^l} = m'$.

It is easy to see that all comparisons are successful if and only if $w \in \texttt{DIMA3}_l$. If check in the path is not successful, the input is rejected. Otherwise, M finishes to read the input and accepts it with probability $\frac{5}{9}$, and rejects it with the remaining probability $\frac{4}{9}$.

In the fifth path, M tosses $\texttt{coin}_I$ $c = 1 + \sum_{i=1}^{m} t_i$ times by reading the part of the input

$$0^{t_1}10^{t_2}1 \cdots 10^{t_{m-1}}110^{t_m}1.$$

We remark that if $w \in \texttt{DIMA3}_l$, c is $64^k \cdot 2^l$ for some $k > 0$. After each coin toss, if the result is heads, M increases the value of the counter by one. Let h be the total number of heads, therefore, the value of the counter is h. Then, M reads h symbols from the part

$$w'_k = (0^{2^{3k+l}-1}1)^{2^{3k}}$$

with the help of the counter. During attempt to read h symbols, if the input is finished, then M rejects the input in this path. Otherwise, M guesses the value x_k with probability at least $1 - \frac{1}{4 \cdot 2^l}$. If the guess is 1, M accepts the input with probability $\frac{5}{9}$, and rejects the input with probability $\frac{4}{9}$. If the guess is 0, M rejects the input.

When M reads h symbols from the part $w'_k = (0^{2^{3k+l}-1}1)^{2^{3k}}$, it guesses the value x_k. Here we use the analysis similar to one presented in the proof of Theorem 16. We can write h as

$$h = i \cdot 8^{k+1} \cdot 2^l + j \cdot 8^k \cdot 2^l + q = (8i + j)8^k \cdot 2^l + q,$$

where $i \geq 0$, $j \in \{0, \ldots, 7\}$, and $q < 8^k \cdot 2^l$.

Due to Lemma 2, x_k is the $(3k + l + 3)$-th digit of $bin(h)$ with probability at least $1 - \frac{1}{4 \cdot 2^l}$. In other words, x_k is guessed as 1 if $j \in \{4, \ldots, 7\}$, and as 0, otherwise. M sets $j = 0$ at the beginning. We can say that for each heads, it consumes a symbol from w'_k. After reading $8^k \cdot 2^l$ symbols, it updates j as $(j + 1) \mod 8$. When the value of the counter reaches zero, M guesses x_k by checking the value of j.

If $w \in \texttt{DIMA3}_l(I)$, then the input is accepted with probability at least

$$4 \cdot \frac{1}{5} \cdot \frac{5}{9} + \frac{1}{5} \cdot \frac{5}{9} \cdot (1 - \frac{1}{4 \cdot 2^l}) = \frac{5}{9} - \frac{1}{36 \cdot 2^l}.$$

If $w \notin \texttt{DIMA3}_l$, the input is rejected with probability at least

$$\frac{1}{5} + 4 \cdot \frac{1}{5} \cdot \frac{4}{9} = \frac{5}{9}.$$

If $w \in \texttt{DIMA3}_l$ and $w \notin \texttt{DIMA3}_l(I)$, the input is rejected with probability at least

$$4 \cdot \frac{1}{5} \cdot \frac{4}{9} + \frac{1}{5} \cdot (1 - \frac{1}{4 \cdot 2^l}) = \frac{5}{9} - \frac{1}{20 \cdot 2^l}.$$

Therefore, the input is recognized with error bound $\epsilon = \frac{4}{9} + \frac{1}{20 \cdot 2^l}$, where $\epsilon < \frac{1}{2}$ for any even $l \geq 0$, and ϵ can be arbitrarily close to $\frac{4}{9}$ for sufficiently large l. Since the cardinality of set $\{I \mid I \in \mathcal{I}\}$ is uncountable, there are uncountably many languages in $\{\texttt{DIMA3}_l(I) \mid I \in \mathcal{I}\}$, each of which is recognized by a bounded-error realtime PCA. □

It is a known fact that bounded-error probabilistic automata can recognize only regular languages in polynomial time [14]. Therefore, realtime PFAs without additional memory cannot recognize uncountably many languages, and hence one is the minimal number of counters required to recognize uncountably many languages in realtime.

We continue with the recognition of unary languages. We start with four counters.

Theorem 28. *Realtime unary P4CAs can recognize uncountably many languages with any error bound.*

Proof. We describe a realtime P4CA, say M_I, that can use a $\texttt{coin}_I$ landing on head with probability p_I for an $I \in \mathcal{I}$. Let c_j $(1 \leq j \leq 4)$ represent the value of j-th counter.

Let $l > 0$ be an integer that determines the error bound ϵ, such that $\epsilon \geq \frac{1}{4 \cdot 2^l}$. The automaton M_I executes an iterative algorithm. We use i to denote the iteration steps. At the beginning, $i = 1$. In each iteration, $64^i \cdot 2^l$ tosses of $\texttt{coin}_I$ are performed. The details are as follows:

- Set $c_1 = 64^i \cdot 2^l$ and $c_2 = 4 \cdot 8^i \cdot 2^l$. (The details are given later.)
- Perform c_1 flips of $\texttt{coin}_I$ and meanwhile increase/decrease the values of c_2 and c_3 by 1. If the coin flip result is heads, one of the counters is increased by 1 and the other one is decreased by 1. When one of them hits zero, the 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 $4 \cdot 8^i \cdot 2^l$ heads, the update strategy on the counters is changed.

- When c_1 hits zero, c_2 and c_3 are equal to t and $4 \cdot 8^i \cdot 2^l - t$, and, the automaton makes its decision on x_i. If the latest strategy is decreasing the value of c_3 or $c_2 = 0$, then x_i 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 22. Here changing the update strategy between c_2 and c_3 refers to the change of bit x_i, which is changed after each $4 \cdot 8^i \cdot 2^l$ heads: It is 0 initially, and then alternates between 1 and 0 $(1, 0, 1, 0, \ldots)$.

At the end of the i-th iteration, we have $c_1 = 0$, $c_2 = t$, and $c_3 = 4 \cdot 8^i \cdot 2^l - t$. We initialize $(i+1)$-th iteration as follows:

- By using c_2 and c_3, we can set $c_1 = 2t + 2(4 \cdot 8^i \cdot 2^l - t) = 8^{i+1} \cdot 2^l$. Now $c_2 = c_3 = c_4 = 0$.
- Set $c_2 = c_3 = 8^{i+1} \cdot 2^l$ by setting $c_1 = 0$. Then, in a loop, until c_2 hits zero: Decrease value of c_2 by 2^l, 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 $8^{i+1} \cdot 2^l$ to c_1.
- $c_1 = 8^{i+1}(8^{i+1} \cdot 2^l) = 64^{i+1} \cdot 2^l$, $c_2 = 0$, $c_3 = 8^{i+1} \cdot 2^l$, $c_4 = 0$. Then, set $c_2 = 4 \cdot 8^{i+1} \cdot 2^l$ 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 are no more input symbols to be read exactly at the end of an iteration, say i-th, and x_i is guessed as 1. Otherwise, the input is always rejected.

The coin tosses part is performed in $64^i \cdot 2^l$ steps. The initialization part for i-th iteration is performed in

$$8^i \cdot 2^l + 8^i \cdot 2^l + 64^i \cdot 2^l + 4 \cdot 8^i \cdot 2^l = 64^i \cdot 2^l + 6 \cdot 8^i \cdot 2^l$$

steps, where $i > 1$. The initialization part for $i = 1$ is performed in $64 \cdot 2^l$ steps.

Based on this analysis, we can easily define the language recognized by M_I. Let $(k_i)_{i \in \mathbb{N}}$ be the sequence such that $k_1 = 128 \cdot 2^l$ and $k_i = k_{i-1} + (6 \cdot 8^i + 2 \cdot 64^i) \cdot 2^l$ for $i > 1$; and, let $\texttt{UP4CA}_1 = \{a^{k_i}\}$ be an unary language corresponding to the sequence $(k_i)_{i \in \mathbb{N}}$. Then, for any $I \in \mathcal{I}$, the realtime P4CA M_I can recognize the language

$$\texttt{UP4CA}_1(\texttt{I}) = \{a^{k_i} \mid a^{k_i} \in \texttt{U4PCA}_1 \text{ for } i \geq 1 \text{ and } i \in I\}$$

with bounded error. The automaton M_I iteratively determines the values of $x_1, x_2, \ldots$ with probability at least $1 - \frac{1}{4 \cdot 2^l}$ due to Lemma 2 and the number of steps for each iteration corresponds with the members of $\mathtt{U4PCA_1}$.

Since $\mathcal{I}$ is an uncountable set and there is a bijection between $I \in \mathcal{I}$ and $\mathtt{UP4CA_1(I)}$, 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 use the well-known simulation technique of k counters by 2 counters.

Theorem 29. *Unary realtime P2CAs can recognize uncountably many languages with any error bound.*

Proof. Let M_I be the realtime P4CA described above and $\mathtt{UP4CA_1(I)}$ be the language recognized by M_I for some $l > 0$. Due to the realtime reading mode, the unary inputs to M_I can also be seen as the time steps. For example, M_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 a huge slowdown [28]. 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, the status of each simulated counter can be easily detected and stored, and then all updates on the simulated counters are applied one by one.

Thus, by fixing the above simulation, we can easily simulate M_I by a P2CA, say M_I'. Then, M_I' recognizes a language with bounded error, say $\mathtt{UP2CA_1(I)}$.

It is easy to see that there is a bijection between

$$\{\mathtt{UP4CA_1(I)} \mid I \in \mathcal{I}\} \text{ and } \{\mathtt{UP2CA_1(I)} \mid I \in \mathcal{I}\},$$

and hence realtime P2CAs also recognize uncountably many languages with bounded error. We remark that for each member of $\mathtt{UP4CA_1(I)}$, the corresponding member of $\mathtt{UP2CA_1(I)}$ is much longer. □

Unary realtime probabilistic automata with one counter can recognize only regular languages with bounded error [24]. Therefore, with less than two

counters it is not possible to recognize uncountably many unary languages with bounded error in realtime.

At this point, we can ask whether we can obtain the same results by using sublinear counter values. For this purpose, we modify the algorithms given above.

Theorem 30. *Realtime unary P4CAs can recognize uncountably many languages in $O(\sqrt{n})$ space on the counters with any error bound.*

Proof. We modify the algorithm given in the proof of Theorem 28 by putting certain amount of delays between two coin tosses. Remember that before starting coin tosses $c_1 = 64^i \cdot 2^l$ and $c_4 = 0$. In the new algorithm, before each coin toss, (1) the automaton decreases the value of c_1 and increases the value of c_4 by one until c_1 reaches zero, and then (2) it decreases the value of c_4 and increases the value of c_1 by one until c_4 reaches zero. After such operation automaton performs the coin toss and decreases the value of c_1 by one. Thus each coin toss can be done in $2 \cdot c_1 + 1$ steps instead of a single step. Therefore, $64^i \cdot 2^l$ coin tosses can be done in

$$64^i \cdot 2^l + 2 \cdot \sum_{j=1}^{64^i \cdot 2^l} j = 64^i \cdot 2^l + 2 \cdot \frac{(64^i \cdot 2^l) \cdot (64^i \cdot 2^l + 1)}{2} = 64^{2i} \cdot 4^l + 2 \cdot 64^i \cdot 2^l$$

steps.

The rest of the algorithm is the same. Then, the modified algorithm recognizes the following languages:

$$\mathtt{UP4CA}'_1(\mathtt{I}) = \{a^{k'_i} \mid a^{k'_i} \in \mathtt{UP4CA}'_1 \text{ for } i \geq 1 \text{ and } i \in I\},$$

where $\mathtt{UP4CA}'_1 = \{a^{k'_i}\}$, $(k'_i)_{i \in \mathbb{N}}$ is the sequence such that $k'_1 = 64^2 \cdot 4^l + 192 \cdot 2^l$ and $k'_i = k'_{i-1} + 6 \cdot 8^i \cdot 2^l + 3 \cdot 64^i \cdot 2^l + 64^{2i} \cdot 4^l$ for $i > 1$. During testing of each potential member $a^{k'_i}$, the input has more than $64^{2i} \cdot 4^l$ symbols, while the space used on counters does not exceed $64^i \cdot 2^l$. Therefore, the modified automaton has $O(\sqrt{n})$ space complexity if the input length is n. □

By using a similar trick and three counters, we can also modify the algorithm given in the proof of Theorem 29 and use $O(\sqrt{n})$ space on the counters.

Theorem 31. *Realtime unary P3CAs can recognize uncountably many languages with any error bound in $O(\sqrt{n})$ space on the counters.*

Proof. In the proof of Theorem 29, two counters simulate four counters. During the simulation, the values of c_1, c_2, c_3 and c_4 are represented with the number $\prod_{j=1}^{4} {p_j}^{c_j}$, which is stored in one counter. We can pick $p_1 = 7$,

$p_2 = 5$, $p_3 = 3$, and $p_4 = 2$. Then, at the i-th iteration, $\prod_{j=1}^{4} {p_j}^{c_j}$ takes the maximum value just before the coin-flip procedure, which is $t_i = 7^{64^i}5^{4\cdot 8^i}$.

By using an extra counter, the algorithm can easily spend t_i^2 dummy steps before starting coin-flip procedure. That is, the value of the third counter is set to t_i, and then the values of the first and second counters are transferred to each other t_i times. Thus, the original algorithm is paused for t_i^2 steps, and after this pause, the algorithm continues with coin-flip procedure.

Therefore, when the space used on the counters does not exceed t_i, the algorithm reads a string longer than t_i^2. Therefore, this new realtime unary P3CA uses $O(\sqrt{n})$ space on its counters when reading a string with length n. □

In the above algorithm, by using an extra counter, the algorithm is paused for t_i^2 steps at the i-th iteration. If we use $(k-2)$ extra counters (for $k > 3$), then the algorithm can be paused for t_i^{k-1} steps at the i-th iteration, and hence the space usage on the counters can be reduced to $O(\sqrt[k-1]{n})$.

Corollary 11. *Realtime unary PkCAs can recognize uncountably many languages with any error bound in $O(\sqrt[k-1]{n})$ space on the counters for any $k > 2$.*

We continue to investigate sublinear space complexity with languages defined over non-unary alphabets. Because only regular languages can be recognized with one-way PTM in $o(\log\log n)$ space with bounded error [19], we can apply the space bound also for counter automata.

Corollary 12. *Bounded-error one-way PCAs cannot recognize uncountably many languages in $o(\sqrt[k]{\log n})$ space for any $k \geq 1$.*

Proof. Suppose that bounded-error one-way PCA can recognize uncountably many languages in $o(\sqrt[k]{\log n})$ space. Then the counters of PCA can be stored on a work tape using $o(\log\log n)$ space, therefore, a $o(\log\log n)$-space PTM can recognize uncountably many languages with bounded error. This makes contradiction with the results from [19], since bounded-error one-way PTMs in $o(\log\log n)$ space can recognize only regular languages. □

Since realtime automata are more restricted version of one-way automata, bounded-error realtime PCAs cannot recognize uncountably many languages in $o(\sqrt[k]{\log n})$ space for some $k \geq 1$. We show that this space bound can be obtained. We first provide P18CA with polylogarithmic space bound.

Corollary 13. *Realtime P18CAs can recognize uncountably many languages with any error bound in polylogarithmic space, i.e., $O(\log^c n)$ for some positive integers c.*

Proof. We can replace the registers of $M_{c,I}$ given in the proof of Theorem 23 with counters, most of which will perform similar functions as registers:

- 2 counters store the values of m and m_0.
- 2 counters store the value of $|m|$ and perform operations with this value.
- 1 counter keeps the number of attempts for generating a prime number.
- 2 counters store two different prime numbers.
- 4 counters store the modules of prime numbers like in p_1, p_2, p_3 and p_4 of M.
- 1 counter keeps track of total number of subtractions.
- 2 counters perform the subtractions.
- 2 counters perform the operations with the values. We will refer them as operational counters.
- 2 counters are used to bound the number of steps for operations. We will refer them as step counters.

The algorithm for P18CA, say $M'_{c,I}$, shortly M', is almost the same as the algorithm of $M_{c,I}$. In order to compare the values of two counters, we can decrease their values. In this way, the number of steps for checking and initializing the values of the counters can be different. On the other hand, our algorithm is designed to spend the same number of steps for many tasks. For this purpose, we use step counters. More precisely, after reading each $bin(i)$, one of the step counters is initialized with the value $2^{c\cdot|m|}$ since this is the maximum value for this part of the algorithm. Thus, whenever is needed, we can easily count (up) to $2^{c\cdot|m|}$ by exchanging the values of step counters.

To generate, initialize, or check the value of $c \cdot |m|$ bits, the counters holding the value $|m|$ are used, i.e., by help of internal states, $c \cdot |m|$ bits can be easily processed.

Two counters (operational) are enough to perform any single operation (generating and initializing certain values and storing temporary values) of M'. We remark that the other counters are used to store the values to be used later.

In order to check the specific bit of a number stored in a counter, we use the technique given in the proof of Theorem 28: One of the counters is initialized with the value m, which should be equal to 64^k for some k, i.e., M' can copy m to a different counter and by help of another counter the

stored value is iteratively divided by 64. Another counter is initialized with the value $2^{\frac{|m|-1}{2}+2}$, which should be equal to $4 \cdot 8^k$. (Remember algorithm from the proof of Theorem 28, where value of c_2 is initialized with $4 \cdot 8^i \cdot 2^l$.) To initialize this counter, its value is set to 1, and then with the help of additional counter and counters that store and work with value $|m|$, the value is iteratively multiplied by 2 for $\frac{|m|-1}{2}+2$ times. Then, we can perform 64^k coin-flips and easily determine whether the result (integer part) of the division of the number of heads by $4 \cdot 8^k$ is odd or even.

At any step of M', the value of any counter does not exceed $2^{c \cdot |m|}$. Since $m = |bin(i)|$ and the length of $bin(i)$ is logarithmic to the length of the input string, space complexity for any counter does not exceed $O(\log^c n)$. □

Corollary 14. *For any $k \geq 1$ there exists a natural number l, such that realtime PlCAs can recognize uncountably many languages in $O(\sqrt[k]{\log n})$ space with any error bound.*

Proof. The values of the counters of P18CA $M'_{c,I}$ (shortly M') given in the proof of Corollary 13 do not exceed $2^{c \cdot |m|}$. We modify M' with additional counters as follows: Each of the counters of M' (except two counters that store and work with value $|m|$) is replaced with $c \cdot k$ counters in a way that their values do not exceed $2^{\frac{|m|}{k}}$. Therefore, for each of the mentioned 16 counters $1 \leq i \leq 16$, the value c_i is distributed among $c \cdot k$ counters to satisfy the following condition:

$$c_i = \sum_{j=1}^{c \cdot k} c_{ij} \cdot 2^{\frac{|m| \cdot (j-1)}{k}},$$

i.e., the value of each counter never exceeds $2^{\frac{|m|}{k}}$. Here we also use two additional counters to help counting up to $2^{\frac{|m|}{k}}$ when necessary.

Thus, we obtain a realtime algorithm similar to M' with $l = 2+16 \cdot c \cdot k+2$ counters. For $t \geq 2^{63}$: $\sum_{i=1}^{t} |bin(i)| > 2^{|bin(t)|}$, therefore, $m = |bin(t)| < \log n$. Since the value of each counter never exceeds $2^{\frac{|m|}{k}}$ and $2^{|m|} \leq 2 \cdot m \leq 2 \cdot \log n$, the total space complexity is $O(\sqrt[k]{\log n})$. □

We finish this section by providing a sublinear space for realtime PCAs having only two counters.

Theorem 32. *Realtime P2CAs can recognize uncountably many languages in $O(\sqrt[k]{n})$ space for any integer $k > 1$ with any error bound.*

Proof. Here we use the PlCA (shortly M'_l) for space complexity $O(\log n)$ given in the proof of Corollary 14, therefore,

$$l = 2 + 16 \cdot c + 2.$$

We also use the idea of simulating any number of counters by two counters. Let k be any integer greater than 1 and $j = l \cdot k$.

As described in the proof of Theorem 29, a P2CA, say M'_2, can simulate M'_l and the maximum value of each counter is $p_1^{c_1} \cdot p_2^{c_2} \cdot \cdots \cdot p_l^{c_l}$, where p_i are distinct prime numbers and c_i is the maximum value of the i-th counter of M'_l $(1 \leq i \leq l)$. By using additional internal states, we can easily modify M'_l, say M'', such that the i-th counter of M'' stores $\frac{t}{2 \cdot \log_2 p_i \cdot j}$ when its original value is t.

Remember that in any step of computation the value of each counter of M'_l does not exceed $2^{|m|} \leq 2 \cdot m \leq 2 \cdot \log n$. Thus, the value of the i-th counter of M'' does not exceed

$$\frac{2 \cdot \log n}{2 \cdot \log_2 p_i \cdot j} = \frac{\log_{p_i} n}{j}.$$

Let M''_2 be the P2CA simulating M''. Then, the maximum value of each counter of M''_2 is calculated as

$$\prod_{i=1}^{l} p_i^{\frac{\log_{p_i} n}{j}} = \prod_{i=1}^{l} n^{\frac{1}{j}} = n^{\frac{l}{j}} = n^{\frac{l}{l \cdot k}} = \sqrt[k]{n}.$$

□

Chapter 5

Related results on quantum and ultrametric automata

In this chapter, we present our results on quantum and ultrametric automata that recognize uncountably many languages. We first give the definitions and then present our results.

Say and Yakaryılmaz [36] proved that bounded-error polynomial-time two-way finite automata with quantum and classical states (2QCFAs) can recognize uncountably many languages. We prove the same result for restarting realtime QCFAs. In [7], it was shown that 2QCFAs with a counter (2QCCAs) can recognize nonregular unary language with bounded error in middle logarithmic space. We show that bounded-error unary 2QCCAs can recognize uncountably many languages in middle logarithmic space. Shur and Yakaryılmaz [40] proved that 2-state unary quantum finite automata (QFAs) can recognize uncountably many languages with cutpoints. We present the same result by fixing a single cutpoint. In [20], Freivalds proved that ultrametric automata with 3 states can recognize uncountably many languages, and we improve this result by proving the same statement for 2-state ultrametric automata.

5.1 Definitions

We refer the reader to [30] for a complete reference on quantum computation, to [35] for a pedagogical introduction to quantum finite automata (QFAs), and to [5] for a comprehensive chapter on QFAs.

An m-state ($Q = \{q_1, \ldots, q_m\}$) quantum system forms an m-dimensional Hilbert space ($\mathcal{H}^m$), complex vector space with inner product, spanned by $\{|q_1\rangle, \ldots, |q_m\rangle\}$, where $|q_j\rangle$ is a column vector with zero entries except the

j-th entry that is 1. A quantum state of the system is a norm-1 vector in $\mathcal{H}^m$:

$$|v\rangle = \alpha_1|q_1\rangle + \cdots + \alpha_m|q_m\rangle, \quad \sum_{j=1}^{m} |\alpha_j|^2 = 1,$$

where α_j is a complex number and represents the amplitude of the system being in $|q_j\rangle$, and the probability of system being in $|q_j\rangle$ is given by $|\alpha_j|^2$.

The quantum system evolves by unitary operators, also known as norm preserving operators, represented by unitary matrices. Let U be a unitary operator (matrix). Then its (i, j)-th entry represents the transition amplitude from $|q_j\rangle$ to $|q_i\rangle$, where $1 \leq j, i \leq n$. After applying U, the new state is

$$|v'\rangle = U|v\rangle = \alpha'_1|q_1\rangle + \cdots + \alpha'_m|q_m\rangle, \quad \sum_{j=1}^{m} |\alpha'_j|^2 = 1.$$

In order to retrieve information from the system, measurement operators are applied. We use a simple one called projective measurement, say P. Formally P is composed by $k \geq 1$ elements $\{P_1, \ldots, P_k\}$. Each P_i is a zero-one diagonal (nonzero) matrix and $P_1 + \cdots + P_k = I$. Therefore, P is designed to decompose $\mathcal{H}^m$ into k orthogonal subspaces and P_j projects any vector to its subspace, where $1 \leq j \leq k$. After applying P to the system when in $|v\rangle$, the system collapses to one of the subspaces, and hence the new quantum state lies only in this subspace. The vector

$$|\widetilde{v_j}\rangle = P_j|v\rangle$$

is the projection of the quantum state to the j-th space, and hence the probability of observing the system in this subspace is given by $p_j = |||\widetilde{v_j}\rangle||^2$. If this happens ($p_j > 0$), the new quantum state is

$$|v_j\rangle = \frac{|\widetilde{v_j}\rangle}{\sqrt{p_j}}.$$

The vector $|\widetilde{v_j}\rangle$ is called unnormalized state vector and tracing quantum systems by such vectors can make the calculations simpler.

Now we give the definition of two-way quantum finite automaton with classical head, known as two-way finite automaton with quantum and classical states (2QCFA) [4], which can use unitary operators and projective measurements on the quantum part. Formally, a 2QCFA M is a 8-tuple

$$M = (S, Q, \Sigma, \delta, s_{init}, q_{init}, s_{acc}, s_{rej}),$$

where

- S is the finite set of classical states,
- Q is the finite set of quantum states,
- Σ is the input alphabet,
- δ is the transition function and is composed by δ_q governing the quantum part and δ_r governing the classical part,
- $s_{init} \in S$ is the initial classical state,
- $q_{init} \in Q$ is the initial quantum state,
- $s_{acc} \in S$ is the accepting classical state, and,
- $s_{rej} \in S$ is the rejecting classical state.

The computational process of 2QCFA M is governed classically. At the beginning of the computation, the classical part is initialized and the state of quantum part is set to $|q_{init}\rangle$. In each step, the current classical state and scanned symbol determines a quantum operator, either a unitary operator or a projective measurement, which is applied to the quantum register. After getting the new quantum state, the classical part is updated. If the quantum operator is unitary, then the classical part is updated like a two-way deterministic finite automaton. If the quantum operator is a measurement, then the outcome is processed classically, that is, the next state and head movement is determined by the current classical state, the measurement outcome, and the scanned symbol. When entering s_{acc} (s_{rej}), the computation halts and the input is accepted (rejected).

Restarting realtime QCFA is defined similarly to restarting realtime PFA (see Chapter 1 for the definition of restarting realtime PFA) — the automaton reads the input in realtime mode, after reading the right end-marker it accepts or rejects the input or restarts the computation from the initial configuration.

A 2QCFA with a counter (2QCCA) is a 2QCFA augmented with a classical counter, where the classical part can access the counter.

We also consider realtime QFAs without classical states. Formally, a realtime QFA M is a 5-tuple

$$M = (Q, \Sigma, \delta, q_{init}, Q_{acc}),$$

where

- Q is the finite set of quantum states,
- Σ is the input alphabet,
- δ is transition function,
- q_{init} is the initial quantum state, and,
- $Q_{acc} \subseteq Q$ is the set of accepting states.

The transition function δ is governing transition of M in the following way: When the automaton reads $\sigma \in \tilde{\Sigma}$ it applies unitary transformation U_σ (which depends on σ) to the current quantum state $|v\rangle$ and changes the quantum state to $|v'\rangle = U_\sigma|v\rangle$.

The input w is placed on the input tape as $\tilde{w}$, and the input head of QFA M is placed on the left end-marker in the beginning of the computation. M reads the input symbol by symbol without pauses. After reading the right end-marker and applying the last unitary transformation, M performs the measurement on the resulting quantum state, and accepts (resp., rejects) the input if the outcome of measurement is $q \in Q_{acc}$ (resp., $q \notin Q_{acc}$).

Language $L \subseteq \Sigma^*$ is said to be recognized by a machine M with cutpoint λ if

- any member is accepted by M with probability greater than λ, and,
- any non-member is accepted by M with probability at most λ.

In this chapter we also consider ultrametric automata that use p-adic numbers in their definition. Therefore, we continue with the definition of p-adic numbers.

Different sciences use p-adic numbers, including chemistry and physics [26, 43], and p-adic numbers are also used in the definitions of ultrametric automata and ultrametric Turing machines [20]. A p-adic digit is a natural number between 0 and $p-1$ where p is an arbitrary prime number. A p-adic integer $(a_i)_{i \in N}$ is an infinite sequence of p-adic digits written from right to left. A p-adic integer can be written like a decimal number $...a_i...a_3a_2a_1$.

For each natural number, there exists its p-adic representation and only a finite number of p-adic digits are not zeros. Negative integers have a different representation in p-adic numbers, namely, they have an infinite sequence of digits $p-1$ to the left. If all digits of a p-adic integer are $p-1$, then we have the p-adic number -1. We can add, subtract and multiply p-adic integers in the same way as natural numbers in base p. The only division that is not possible in p-adic integers is division by p. For example, if we want

to have a p-adic integer $\frac{1}{p}$, equation $p \cdot x = 1$ should have a solution for x, but multiplication by p-adic integer p gives zero in the right-most p-adic digit. Nonetheless, p-adic integers can represent any integer and most of the rational numbers, except for those having a positive integral power of p in the divisor.

There also exist p-adic float numbers, which can have a decimal point and are infinite to the left side but finite to the right side. For example, p-adic number $\frac{1}{p}$ can be written as

$$...0000.1.$$

The field of p-adic numbers is denoted by $\mathbb{Q}_p$. For the curious reader, David A. Madore has written extensively about p-adic numbers and further information on the subject can be found in [27].

To measure a p-adic number, we require the absolute value of a p-adic number. If p is a prime number, then the p-adic ordinal of the rational number a, denoted by $ord_p a$, is the largest m such that p^m divides a.

For any rational number x its p-norm (p-adic absolute value) is

$$\|x\|_p = \begin{cases} \frac{1}{p^{ord_p x}}, & \text{if } x \neq 0 \\ 0, & \text{if } x = 0. \end{cases}$$

For example, if $p = 5$, $\|50\|_5 = \|2 \cdot 5^2\|_5 = \|5^2\|_5 = \frac{1}{5^2} = \frac{1}{25}$, if $p = 2$, $\|50\|_2 = \|2\|_2 = \frac{1}{2}$, but for any other prime number p, $\|50\|_p = 1$.

Ultrametric automata are defined similarly to probabilistic automata. A realtime p-ultrametric finite automaton is a 6-tuple

$$(S, \Sigma, \delta, q_0, F, \Lambda),$$

where

- S is the finite set of states,
- Σ is the input alphabet,
- $\delta \colon S \times \Sigma \times S \to \mathbb{Q}_p$ is the transition function,
- $q_0 \colon S \to \mathbb{Q}_p$ is the initial amplitude distribution,
- $F \subseteq S$ is the set of accepting states, and,
- $\Lambda = (\lambda, \Diamond)$ is the acceptance condition where $\lambda \in \mathbb{R}$ is the acceptance threshold and $\Diamond \in \{\geq, \leq\}$.

A probabilistic automaton has transition probabilities that are real numbers. In the case of p-ultrametric automaton the transitions have amplitudes, which are p-adic numbers. Therefore, we can assume that, for a p-ultrametric automaton, prime number p is also a parameter. Probabilistic automata have their beginning distribution of probabilities among the states and transitions are performed with probabilities. In ultrametric automata every state has a beginning amplitude, and, by reading the input string, transitions are done with amplitudes. This means that final amplitudes of the states are calculated in the same way as probabilities in probabilistic automata. Realtime p-ultrametric automaton reads the input symbol by symbol without pauses and after reading the symbol changes the amplitudes of the states according to δ. To get the result after reading the input string, after reading the right end-marker the amplitude of every accepting state is transformed into p-norm and the string is accepted if and only if p-norm sum of accepting states satisfies the acceptance condition.

All possible p-adic numbers are allowed to be used in p-ultrametric automata. This was allowed in the first definition of ultrametric automata because Paavo Turakainen defined generalized finite automata where the "probabilities" can be arbitrary real numbers and he has proven that languages recognizable by these generalized finite automata are the same as for ordinary probabilistic finite automata [42]. Ultrametric automata defined in this way have great capabilities, for example, they are able to recognize nonrecursive languages [20].

5.2 Results

Quantumly, the membership of each positive integer in any $I \in \mathcal{I}$ can be represented as a single rotation on $\mathbb{R}^2$ with the angle, originally given in [2]:

$$\theta_I = 2\pi \sum_{i=1}^{\infty} \left(\frac{x_i}{8^{i+1}} \right), \qquad \begin{array}{rl} x_i = & 1, \text{ if } i \in I \\ x_i = & -1, \text{ if } i \notin I \end{array}.$$

For any $I \in \mathcal{I}$, we can compute the membership of the positive integer j in I by using the technique described in [2, 36]. We call it Procedure ADH.

The qubit spanned by $\{|q_1\rangle, |q_2\rangle\}$ is set to $|q_1\rangle$. Then, it is rotated with angle θ_I 8^j times, which leaves the quantum state having angle

$$8^j \cdot 2\pi \sum_{i=1}^{\infty} \left(\frac{x_i}{8^{i+1}} \right) = \pi \left(\frac{x_j}{4} \right) + 2\pi \sum_{i=j+1}^{\infty} \left(\frac{x_i}{8^{i-j+1}} \right)$$

from the initial position. After an additional rotation by $\frac{\pi}{4}$ counter clockwise, the final angle from $|q_1\rangle$ is $\frac{\pi}{2}+d$ if $x_j = 1$ ($j \in I$) and it is d if $x_j = -1$ ($j \notin I$), where d is sufficiently small such that the probability of the qubit being in $|q_2\rangle$ ($|q_1\rangle$) is bigger than 0.98 if $j \in I$ ($j \notin I$). The success probability can be increased arbitrarily close to 1 by tensoring multiple qubits with implemented Procedure ADH and choosing the most frequent value of the measurement as the decision.

Say and Yakaryılmaz [36] presented a bounded-error polynomial-time 2QCFA algorithm, say M, for

$$\texttt{POWER-EQ} = \{aba^7ba^{7\cdot 8}ba^{7\cdot 8^2}ba^{7\cdot 8^3}b\ldots ba^{7\cdot 8^m} \mid m \geq 0\}.$$

We remark that every member has 8^{m+1} a's for some $m \geq 0$. It is clear that for any member of POWER-EQ having 8^{m+1} a's and for any $I \in \mathcal{I}$, M can be modified, say M_I, in order to determine whether m is in I or not by using Procedure ADH with high probability. Therefore, M_I can recognize the following language with bounded error [36]:

$$\texttt{POWER-EQ(I)} = \{w \in \{a,b\}^* \mid w \in \texttt{POWER-EQ} \text{ and } \log_8(|w|_a) \in I\}.$$

Then, we can conclude that 2QCFAs can recognize uncountably many languages with bounded error.

Procedure ADH can be trivially implemented by a restarting realtime QCFA having a single qubit (multiple qubits for better success probability), say M_I for $I \in \mathcal{I}$. Therefore, if we show that POWER-EQ is recognized by a restarting realtime QCFA, say M, then, we can conclude that restarting realtime QCFAs can recognize uncountably many languages. Since M can execute M_I in parallel to its original algorithm, i.e., M and M_I are tensored such that if M is in the restarting state, then all computation is restarted; otherwise, the input is accepted if and only if both M and M_I give the decision of "accepting". The obtained restarting realtime QCFA gives its decisions with bounded error. We refer the reader to [47] for the technical details how to obtain a bounded-error restarting realtime QCFA by tensoring two bounded-error restarting realtime QCFAs, where the results are given for general realtime QFA models, but it can be obtained for realtime QCFAs in the same way since general realtime QFA models and realtime QCFAs can simulate each other exactly.

Theorem 33. *The language* POWER-EQ *can be recognized by a restarting realtime QCFA M with any error bound.*

Proof. The quantum part of M has 9 states ($\{q_1,\ldots,q_9\}$), but only the first three of them are used to process useful information. Before each unitary

operation, the quantum state has always zeros after these significant first three entries

$$(\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 0 \quad \cdots \quad 0)^T .$$

After we apply a unitary operator U, we obtain a new quantum state. Then, we make some measurements such that if the system is in $span\{|q_4\rangle, \ldots, |q_9\rangle\}$, then the computation is always restarted. Therefore, if the computation is not restarted, the new quantum state has always zeros for the last six entries.

$$(\alpha'_1 \quad \alpha'_2 \quad \alpha'_3 \quad 0 \quad \cdots \quad 0)^T .$$

Sometimes the measurement operator can also affect the first three states that will be specified later.

Each unitary operator is a (9×9)-dimensional unitary matrix. However, the significant parts are the top-left (3×3)-dimensional matrices due to the measurement operators. Therefore, we can trace the computation only by a 3-dimensional vector and (3×3)-dimensional matrices.

We describe our algorithm with integer matrices with a real coefficient $0 < l < 1$:

$$lB = l \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} .$$

We remark that lB is the top-left corner of a unitary matrix and all the other entries can be filled arbitrarily providing that the matrix is unitary. For our purpose, first, we define our (3×3)-dimensional matrices B's, which do the main tasks, and then complete the missing parts of unitary matrices by picking a fixed l for all B's. We refer the reader to [45, 48] for the details of how to pick nonnegative real $l < 1$ and fill the missing parts of unitary matrices.

After a measurement operator, we can obtain more than one unnormalized state vector having norm less than 1. Depending on the measurement outcome, the system collapses into one of them, and then the corresponding unnormalized state vector is normalized (norm-1 vector). On the other hand, since the probabilities can be calculated directly from the entries of unnormalized state vectors, we trace the computation with unnormalized state vectors.

After all these technical descriptions, we can give the details of our quantum algorithm. (QFA algorithms based on such assumptions have been presented before (e.g., see [44]).)

The quantum part is in state $|v_0\rangle = (1 \quad 0 \quad 0)^T$ at the beginning. If the input does not start with aba^7b, then it is rejected deterministically. Otherwise,

by reading 7 a's, the quantum part is set to

$$|\widetilde{v_7}\rangle = l^7 \begin{pmatrix} 1 \\ 7 \cdot 56 \\ 0 \end{pmatrix} = \left(l \begin{pmatrix} 1 & 0 & 0 \\ 56 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right)^7 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

If the input is aba^7b, then it is accepted deterministically.

In the remaining part, we assume that the input is of the form

$$aba^7b(a^+b)^+.$$

Otherwise, the input is rejected deterministically. We remark that, after each quantum step, the computation is restarted with some probability, and hence the computation in a single round can reach to the end-marker only with a very small (exponentially small in the input length) probability.

At the beginning of each block of a's, say the i-th block, the quantum state is

$$|\widetilde{v_{t_{i-1}}}\rangle = l^{t_{i-1}} \begin{pmatrix} 1 \\ 8k_{i-1} \\ 0 \end{pmatrix},$$

where

- $|\widetilde{v_{t_{i-1}}}\rangle$ is the non-halting unnormalized quantum state vector,
- the first block ($i = 1$) refers the first a's after aba^7b,
- k_{i-1} is the number of a's in the previous block with $k_0 = 7$, and
- $t_{i-1} = k_0 + k_1 + \cdots + k_{i-1} + i - 1$ is the number of unitary operators applied until that step, i.e., 7 unitary operators are applied on the input aba^7b (the other quantum operators on aba^7 can be assumed as identity operator), and then for each symbol after aba^7b, a unitary operator is applied ($i - 1$ refers the number of b's).

Notice that the expected number of a's in the i-th block is already written as the amplitude of the $|q_2\rangle$, i.e., for any member, the number of a's in the i-th block is 8 times of the number of a's in the previous block.

During reading the i-th block, the number of a's is counted and kept as the amplitude of $|q_3\rangle$:

$$l^{t_{i-1}+j} \begin{pmatrix} 1 \\ 8k_{i-1} \\ j \end{pmatrix} = l \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} l^{t_{i-1}+j-1} \begin{pmatrix} 1 \\ 8k_{i-1} \\ j-1 \end{pmatrix}.$$

After reading all a's in the block and just before reading b, the quantum state is

$$l^{t_i-1}\begin{pmatrix} 1 \\ 8k_{i-1} \\ k_i \end{pmatrix}.$$

Then, we obtain the following vector after reading b and just before the measurement:

$$l^{t_i}\begin{pmatrix} 1 \\ 8\cdot k_i \\ k_i - 8k_{i-1} \end{pmatrix} = l\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 8 \\ 0 & -1 & 1 \end{pmatrix} l^{t_i-1}\begin{pmatrix} 1 \\ 8k_{i-1} \\ k_i \end{pmatrix}.$$

After this, the measurement operator, in addition to the previously described standard behavior, also checks whether the system is in $span\{|q_1\rangle, |q_2\rangle\}$ or $span\{|q_3\rangle\}$. In the latter case, the input is rejected, and the computation continues, otherwise. Here we have two cases:

- If $k_i \neq 8k_{i-1}$, then $k_i - 8k_{i-1}$ is a nonzero integer, and hence, the input is rejected with probability at least l^{2t_i}.
- If $k_i = 8k_{i-1}$, then the input is rejected with zero probability.

That is, for any non-member, the input is rejected after a block with some nonzero probability. For each member, on the other hand, the input is never rejected until the end of the computation.

At the end of the computation, the state before reading the right end-marker is

$$l^{t_m}\begin{pmatrix} 1 \\ 8\cdot k_m \\ 0 \end{pmatrix}$$

if there are m blocks of a's. Then, we obtain the following quantum state after reading the right end-marker

$$l^{t_m+1}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = l\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} l^{t_m}\begin{pmatrix} 1 \\ 8\cdot k_m \\ 0 \end{pmatrix}$$

and the input is accepted if $|q_1\rangle$ is observed. Then, the input is accepted with probability l^{2t_m+2}, which is clearly at least l^2 times of any possible rejecting probability before.

Now, we can analyze a single round of M, the period from the initial configuration to giving a decision or to restarting the computation, and then calculate the overall probabilities on the given input.

Any member is accepted with an exponentially small but non-zero probability (l^{2t_k+2}) and it is rejected with zero probability. Therefore, in exponential expected time, the input is accepted with probability 1.

Any non-member, on the other hand, is again accepted with a very small non-zero probability, but it is also rejected with a probability sufficiently bigger than the accepting probability. Therefore, in exponential expected time, the input is rejected with probability $\frac{R}{A+R}$ that is at least

$$\frac{R}{l^2R+R} = \frac{1}{1+l^2} > \frac{1}{2},$$

where A and R are the accepting and rejecting probabilities, respectively, in a single round (see [46] for the details of calculating the overall rejecting probability). We can pick l arbitrarily small, and hence the rejecting probability can be arbitrarily close to 1. □

Corollary 15. *Exponential expected time restarting realtime QCFAs can recognize uncountably many languages with any error bound.*

It is still open whether restarting realtime QCFAs can recognize a nonregular language with bounded error in polynomial time.

Some algorithms can be space sufficient only for the members. That is known as recognition with middle space [41]. The standard space usage is then called as recognition with strong space.

Now we will consider middle space complexity results. We know that 2QCCAs can recognize the following nonregular unary language with any error bound $\epsilon > 0$ in middle logarithmic space [7]:

$$\texttt{UPOWER} = \{a^{2^m} \mid m \geq 0\}.$$

Here the base-2 is not essential and it can be replaced with any integer bigger than 2. Therefore, 2QCCAs can also recognize

$$\texttt{UPOWER}_3 = \{a^{8^m} \mid m \geq 0\}$$

with bounded error in middle logarithmic space (by slightly modifying the algorithm for UPOWER). Moreover, for any $I \in \mathcal{I}$, 2QCCAs recognize the following language

$$\texttt{UPOWER}_3(\texttt{I}) = \{a^{8^m} \mid m-1 \in I\}$$

with bounded error in middle logarithmic space, i.e., we first determine whether the input is of the form a^{8^m} with high probability. If so, we call Pro-

cedure ADH, which does not use the counter, to determine whether $(m-1)$ is in I or not with high probability.

Theorem 34. *Unary middle logarithmic-space 2QCCAs can recognize uncountably many languages with any error bound.*

Now we switch our attention to finite automata that recognize languages with cutpoint. Rabin has proven that 2-state PFAs can recognize uncountably many languages [32]. The proof is based on a single PFA and the fact that there are uncountably many cutpoints. But the question whether 2-state PFAs can recognize uncountably many languages with fixed cutpoint has not been considered for long time up to our knowledge. Very recently, the following result was shown [29].

Fact 6. *Realtime 2-state PFAs can recognize uncountably many languages with cutpoint* $\frac{1}{2}$.

Shur and Yakaryılmaz have obtained similar result for 2-state unary QFAs, and the proof is based on a single QFA and the fact that there are uncountably many cutpoints, where each cutpoint defines different language for constructed QFA [39, 40]. Here we show how to recognize uncountably many unary languages with 2-state QFAs with fixed cutpoint.

We denote a rotation operator on a single qubit by U_θ, where θ is an angle of rotation on $\mathbb{R}^2$ counterclockwise. We define a unary 2-state rotation QFA: $M_\theta = (\{q_{init}, q_2\}, \{a\}, U_\theta, q_{init}, \{q_{init}\})$. Let $\theta_1 = 2\pi \cdot \alpha_1$ and $\theta_2 = 2\pi \cdot \alpha_2$, where $\alpha_1 = 0.a_1a_2a_3\cdots$ and $\alpha_2 = 0.b_1b_2b_3\cdots$ are binary representations of two irrational numbers in interval $[0, 1)$.

Theorem 35. *If* $a_i = b_i$ *for all* $i > 1$ *and* $a_1 \neq b_1$, *then* M_{θ_1} *and* M_{θ_2} *recognize the same language with cutpoint* $\frac{1}{2}$.

Proof. Without loss of generality, let $\alpha_1 = 0.0a_2a_3\cdots$ and $\alpha_2 = 0.1b_2b_3\cdots$. Then, for any $k > 0$, $k\theta_2 = k\pi + k\theta_1$. Therefore, after reading $k \geq 0$ input symbols both M_{θ_1} and M_{θ_2} have equal accepting probabilities, and hence both automata accept the same inputs. □

Theorem 36. *If* $a_2 \neq b_2$, *then there are cases in which* M_{θ_1} *and* M_{θ_2} *can recognize the same language with cutpoint* $\frac{1}{2}$.

Proof. If $a_1 = b_1 = 0$ and $\alpha_2 = 0.5 - \alpha_1$, then $k\theta_2 = k\pi - k\theta_1$ and so the string a^k $(k \geq 0)$ is accepted with equal probabilities by M_{θ_1} and M_{θ_2}.

If $a_1 = b_1 = 1$ and $\alpha_2 = 1.5 - \alpha_1$, then $k\theta_2 = 3k\pi - k\theta_1$ and so similarly the string a^k $(k \geq 0)$ is accepted with equal probabilities by M_{θ_1} and M_{θ_2}.

Remark that in both cases $a_2 \neq b_2$ because α_1 and α_2 are irrational numbers:

- It is given that $\alpha_1 + \alpha_2 = 0.5$. Then, $\alpha_1 \neq 0.25$ and $\alpha_2 \neq 0.25$ because they are not rational. Thus, either $\alpha_1 > 0.25$ and $\alpha_2 < 0.25$ or $\alpha_1 < 0.25$ and $\alpha_2 > 0.25$; or equivalently, either a_2 is 1 and b_2 is 0 or a_2 is 0 and b_2 is 1.

- It is given that $\alpha_1 + \alpha_2 = 1.5$. Then, $\alpha_1 \neq 0.75$ and $\alpha_2 \neq 0.75$ because they are not rational. Thus, either $\alpha_1 > 0.75$ and $0.5 < \alpha_2 < 0.75$ or $0.5 < \alpha_1 < 0.75$ and $\alpha_2 > 0.75$; or equivalently, either a_2 is 1 and b_2 is 0 or a_2 is 0 and b_2 is 1.

In the above cases, both M_{θ_1} and M_{θ_2} recognize the same language with any given cutpoint including $\frac{1}{2}$. □

Theorem 37. *If $a_2 = b_2$ and $a_i \neq b_i$ for some $i > 2$, then M_{θ_1} and M_{θ_2} recognize different languages with cutpoint $\frac{1}{2}$.*

Proof. Let $i > 2$ be the smallest i for which $a_i \neq b_i$. Without loss of generality, we assume that $a_i = 0$ and $b_i = 1$ (the analysis for case when $a_i = 1$ and $b_i = 0$ is similar). Then, we have

$$2^{i-3}\theta_1 = 2l_1\pi + (0.a_{i-2}a_{i-1}0) \cdot 2\pi + \theta_1'$$

and

$$2^{i-3}\theta_2 = 2l_2\pi + (0.a_{i-2}a_{i-1}1) \cdot 2\pi + \theta_2',$$

where $\theta_1', \theta_2' < \frac{\pi}{4}$ and $l_1, l_2 \geq 0$. Thus, if both M_{θ_1} and M_{θ_2} read 2^{i-3} input symbols, then:

- if $a_{i-1} = 0$, then M_{θ_1} accepts the input with probability greater than $\frac{1}{2}$, and M_{θ_2} accepts the input with probability less than $\frac{1}{2}$;

- if $a_{i-1} = 1$, then M_{θ_1} accepts the input with probability less than $\frac{1}{2}$, and M_{θ_2} accepts the input with probability greater than $\frac{1}{2}$.

Therefore, the languages recognized by M_{θ_1} and M_{θ_2} with cutpoint $\frac{1}{2}$ are different. □

From Theorem 37 we can conclude that unary 2-state realtime QFAs can recognize uncountably many languages with cutpoint $\frac{1}{2}$.

We close this chapter with our results on ultrametric finite automaton.

Theorem 38. *For every prime number p a realtime p-ultrametric automaton with two states can recognize uncountably many languages.*

Proof. Let p be an arbitrary prime number, $\alpha = \cdots a_3 a_2 a_1$ be an arbitrary p-adic integer where all $a_i \in \{0, 1\}$. We define a language L_α in the following way: A binary sequence belongs to L_α if and only if it is equal to the last digits of α, namely, $a_1 a_2 a_3 \cdots$. Now we construct an ultrametric automaton to recognize L_α.

The automaton has two states, s_1 and s_2, and $F = \{s_1\}$. The accepting state s_1 has a beginning amplitude α. Because α is a p-adic integer, the p-adic digits after the decimal point are zeros. State s_2 has a beginning amplitude $\frac{1}{p}$. We define the transition amplitudes with the following transition matrix for symbol "1"

$$\begin{pmatrix} \frac{1}{p} & -1 \\ 0 & 1 \end{pmatrix},$$

and the following transition matrix for symbol "0"

$$\begin{pmatrix} \frac{1}{p} & 0 \\ 0 & 1 \end{pmatrix},$$

where (i, j)-th entry represents the transition amplitude from s_j to s_i.

Assume that the input string is $w = w[1]w[2]\cdots w[n]$ and in step i automaton reads $w[i]$. If $w[i] = 1$, $\frac{1}{p}$ is subtracted from the amplitude of state s_1, otherwise nothing is subtracted $(0 \cdot \frac{1}{p})$. If $w[i] = a_i$, the remaining amplitude of s_1 has i-th p-adic digit equal to zero, and it will remain zero for all the time of the work of the automaton. If $w[i] \neq a_i$, s_1 has i-th p-adic digit different from zero (it is 1 if $a_i = 1$ and $w[i] = 0$, and it is $p - 1$ if $a_i = 0$ and $w[i] = 1$).

When the whole input string w is read, the accepting state s_1 has an amplitude followed by $b_n b_{n-1} ... b_1$ after the decimal point, where all the digits in $b_n b_{n-1} ... b_1$ are zeros if and only if the input string belongs to L_α. If the input string does not belong to L_α, the amplitude of the state s_1 will have at least one digit after the decimal point that is not zero. The p-norm of such number is at least p. If the p-adic number does not have nonzero digits after the decimal point, its p-norm is not greater than 1. An ultrametric automaton will accept the input string if the p-norm is less than or equal to 1. Otherwise, the input string is rejected.

If $\alpha_1 \neq \alpha_2$, then $L_{\alpha_1} \neq L_{\alpha_2}$. There are total uncountably many such numbers α, therefore, uncountably many different languages L_α. The constructed ultrametric automaton recognizes L_α for any prime number p and any p-adic integer α. Therefore, for any prime number p, two-state p-ultrametric automaton can recognize uncountably many languages. □

Chapter 6

Conclusion

The results presented in this thesis are original, and they are new contributions to the fields of computational complexity, probabilistic and quantum computation, and automata theory.

In this thesis, we consider different probabilistic computational models that recognize or verify uncountably many languages. We examine the models on different uncountable sets of unary and binary languages, as well as on the sets of all languages. We obtain many new results, and we also leave certain open problems for future work.

We show how to encode the membership information for uncountably many languages in the probabilistic binary value, and how to guess the membership information with probabilistic experiments correctly with bounded error. After that, we show how to extend the experiment for obtaining arbitrarily small error bound. We also investigate different encodings, and provide some evidence that our special encoding works fine and why trivial encoding does not work.

We obtain different upper bounds on space complexity for PTMs and probabilistic counter automata with different number of counters that recognize any language with bounded error. We summarize our results in Table 6.1. Note that in unary case each considered model uses exponentially less resources than in binary case because of smaller number of coin tosses. We also remark that for one-way probabilistic counter automata it is not enough to have one counter to recognize all languages with bounded error.

alphabet	machine	space	time
unary	PTM	$O(n)$	$O(2^n)$
binary	PTM	$O(2^n)$	$O(2^{2^n})$
unary	1P4CA	$O(2^n)$	$O(2^n)$
binary	1P4CA	$O(2^{2^n})$	$O(2^{2^n})$
unary	1P2CA	$O(2^{2^n})$	$O(2^{2^n})$
binary	1P2CA	$O(2^{2^{2^n}})$	$O(2^{2^{2^n}})$

All results are proven for any error bound $\epsilon > 0$.

Table 6.1: Recognition of any language.

After that, we consider the verification power of probabilistic machines that can verify any language. We obtain different results for two-way private-coin IPSs. We summarize our results in Table 6.2. We remark that in case of weak IPSs non-members may not be rejected with high probability, or, in other words, the computation may not halt with high probability on non-members.

alphabet	type of IPS	verifier	space	time
unary	one prover	PTM	$O(\log n)$	$O(2^n)$
binary	one prover	PTM	$O(n)$	$O(2^{2^n})$
binary	weak and one prover	sweeping PFA	$O(1)$	$O(2^{2^{2^n}})$
binary	two provers	1PFA	$O(1)$	$O(2^{2^n})$

All results are proven for any error bound $\epsilon > 0$.

Table 6.2: Verification of any language.

Then, we consider different probabilistic models that can recognize uncountably many languages with bounded error. We summarize our results in Table 6.3. Some bounds presented in the table are tight. Realtime counter automata require at least one counter in binary case and two counters in unary case. Realtime PTMs cannot have $o(\log \log n)$ space complexity. For any $k > 0$:

- Realtime multi-counter automata cannot have $o(\sqrt[k]{\log n})$ space complexity.
- There exists m, such that realtime PmCAs can have $O(\sqrt[k]{\log n})$ space complexity.
- Realtime P2CAs can have $O(\sqrt[k]{n})$ space complexity.

Bounded-error polynomial-time PTMs using $o(\log\log n)$ space can recognize only regular languages, and we show polynomial-time $O(\log\log n)$-space unary sweeping PTM for uncountably many languages for any error bound. Note that results obtained for restarting realtime models are also valid for postselecting realtime models.

In our proofs, we use unary languages `AM75`, `ULOG`, `UP2CA`, binary languages `LOG`, `LOGLOG`, `DIMA`, `DIMA3`, and their certain modifications.

alphabet	machine	space	time
unary	sweeping PTM	$O(\log\log n)$	$O(n\log n)$
unary	2P2CA	$O(\log n)$	$O(n\log n)$
binary	sweeping PCA	$\omega(1)$	$O(2^n)$
binary	restarting realtime PCA[1]	$\omega(1)$	$O(2^n)$
binary	2PCA	$O(n)$	$O(n)$
binary	sweeping PCA	$O(n)$	$O(n\sqrt{n})$
unary	PTM	$O(\log n)$	realtime
unary	$(k>2)$ PkCA	$O(\sqrt[k-1]{n})$	realtime
unary	P2CA	$O(n)$	realtime
binary	PTM	$O(\log\log n)$	realtime
binary	(for $k\geq 1$ exists m) PmCA	$O(\sqrt[k]{\log n})$	realtime
binary	P2CA	$O(\sqrt[k]{n})$	realtime
binary	PCA[2]	$O(n)$	realtime

[1] This result is proven for any error bound $\epsilon > \frac{2}{5}$.
[2] This result is proven for any error bound $\epsilon > \frac{4}{9}$.
All other results are proven for any error bound $\epsilon > 0$.

Table 6.3: Recognition of uncountably many languages.

After that, we consider different probabilistic models that can verify uncountably many languages with bounded error. We consider one-way private-coin IPSs with one prover and summarize our results in Table 6.4. Sweeping PFA verifier mentioned in the table is guaranteed to stop the computation in linear time, and the other results have expected time complexity. In our proofs, we use the languages `USQAURE`, `UPOWER64`, `DIMA2`, and $\texttt{DIMA2}_1$.

alphabet	verifier	space	time
unary	2PFA	$O(1)$	$O(n^2)$, expected
binary	sweeping PFA	$O(1)$	$O(n)$
unary	restarting realtime PFA[1]	$O(1)$	$O(2^n)$, expected
binary	restarting realtime PFA	$O(1)$	$O(n)$, expected

[1] This result is proven for error bound $\frac{2}{5}$, all other results are proven for any error bound $\epsilon > 0$.

Table 6.4: Verification of uncountably many languages.

We also present related results on quantum and ultrametric automata. We show that exponential expected time restarting realtime QCFAs can recognize uncountably many languages with any error bound. For the same task, we also present unary middle logarithmic–space 2QCCAs. Then, we switch our focus to the recognition with cutpoint. We show that unary 2-state realtime QFAs can recognize uncountably many languages with cutpoint $\frac{1}{2}$. Similarly, we also show that, for any prime number p, a realtime p-ultrametric automaton with two states can recognize uncountably many languages.

We specify certain open problems. Very restricted bounded-error non-constant-space models can recognize uncountably many languages. Even though constant-space PTMs can verify uncountably many languages, it is open whether they can recognize uncountably many languages with bounded error.

The next interesting open question is about the verification of all languages in constant space. In our results in IPS with one prover non-members may not be rejected with high probability, while in IPS with two provers non-members are always rejected with high probability. Therefore, the question is, whether constant-space verifier can reject non-members with high probability in IPS with one prover.

There are open problems regarding unary languages. We still do not know, whether unary two-way $o(\log \log n)$-space PTMs or one-way $o(\log n)$-space PTMs can recognize nonregular languages. If one of the mentioned models turns out to recognize nonregular languages, then most probably it can recognize uncountably many languages.

Bibliography

[1] S. Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A*, 461(2063):3473–3482, 2005.

[2] L. M. Adleman, J. DeMarrais, and M.-D. A. Huang. Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540, 1997.

[3] H. Alt and K. Mehlhorn. A language over a one symbol alphabet requiring only $O(\log \log n)$ space. *ACM SIGACT*, 7:31–33, 1975.

[4] A. Ambainis and J. Watrous. Two-way finite automata with quantum and classical states. *Theoretical Computer Science*, 287(1):299–311, 2002.

[5] A. Ambainis and A. Yakaryılmaz. Automata and quantum computing. Technical Report 1507.01988, arXiv, 2015.

[6] L. Babai. Trading group theory for randomness. In *STOC'85: Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[7] Z. Bednárová, V. Geffert, K. Reinhardt, and A. Yakaryılmaz. New results on the minimum amount of useful space. *International Journal of Foundations of Computer Science*, 27(2):259–282, 2016.

[8] M. Ben-or, S. Goldwasser, J. Killian, and A. Wigderson. Multi prover interactive proofs: How to remove intractability. In *Proc. of 20th STOC*, pages 113–131, 1988.

[9] K. Chandrasekharan. *Chebyshev's theorem on the distribution of prime numbers*, pages 63–83. Springer, 1968.

[10] A. Condon. *Complexity Theory: Current Research*, chapter The complexity of space bounded interactive proof systems, pages 147–190. Cambridge University Press, 1993.

[11] A. Condon and R. J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *FOCS'89: Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 462–467, 1989.

[12] H. G. Demirci, M. Hirvensalo, K. Reinhardt, A. C. C. Say, and A. Yakaryılmaz. Classical and quantum realtime alternating automata. In *NCMA*, volume 304, pages 101–114. Österreichische Computer Gesellschaft, 2014. (arXiv:1407.0334).

[13] H. G. Demirci, A. C. C. Say, and A. Yakaryılmaz. The complexity of debate checking. *Theory of Computing Systems*, 57(1):36–80, 2015.

[14] C. Dwork and L. Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal on Computing*, 19(6):1011–1123, 1990.

[15] C. Dwork and L. Stockmeyer. Finite state verifiers I: The power of interaction. *Journal of the ACM*, 39(4):800–828, 1992.

[16] U. Feige and A. Shamir. Multi-oracle interactive protocols with space bounded verifiers. In *Structure in Complexity Theory Conference*, pages 158–164, 1989.

[17] U. Feige, A. Shamir, and M. Tennenholz. The noisy oracle problem. In *Proceedings of CRYPTO 88*.

[18] R. Freivalds. Probabilistic two-way machines. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 33–45, 1981.

[19] R. Freivalds. Space and reversal complexity of probabilistic one-way Turing machines. In *Conference on Fundamentals of Computation Theory*, pages 159–170, 1983.

[20] R. Freivalds. Ultrametric finite automata and Turing machines. In *Developments in Language Theory*, volume 7907 of *LNCS*, pages 1–11. Springer, 2013.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[22] J. Kaņeps. Regularity of one-letter languages acceptable by 2-way finite probabilistic automata. In *FCT'91*, pages 287–296, 1991.

[23] J. Kaņeps and R. Freivalds. Minimal nontrivial space complexity of probabilistic one-way Turing machines. In *MFCS'90*, volume 452 of *LNCS*, pages 355–361, 1990.

[24] J. Kaņeps, D. Geidmanis, and R. Freivalds. Tally languages accepted by Monte Carlo pushdown automata. In *RANDOM*, volume 1269 of *LNCS*, pages 187–195. Springer, 1997.

[25] A. Y. Kitaev, A. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

[26] S. V. Kozyrev. Ultrametric analysis and interbasin kinetics. In *p-adic mathematical physics: 2nd International Conference*, volume 826, pages 121–128. AIP Publishing, 2006.

[27] D. A. Madore. A first introduction to p-adic numbers. Online, 2000.

[28] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[29] A. Naumovs, M. Dimitrijevs, and A. Yakaryılmaz. The minimal probabilistic and quantum finite automata recognizing uncountably many languages with fixed cutpoints. Technical Report 1904.01381, arXiv, 2019.

[30] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[31] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, New York, 1971.

[32] M. O. Rabin. Probabilistic automata. *Information and Control*, 6:230–243, 1963.

[33] A. C. C. Say and A. Yakaryılmaz. Computation with narrow CTCs. In *UC*, volume 6714 of *LNCS*, pages 201–211, 2011.

[34] A. C. C. Say and A. Yakaryılmaz. Computation with multiple CTCs of fixed length and width. *Natural Computing*, 11(4):579–594, 2012.

[35] A. C. C. Say and A. Yakaryılmaz. Quantum finite automata: A modern introduction. In *Computing with New Resources*, volume 8808 of *LNCS*, pages 208–222. Springer, 2014.

[36] A. C. C. Say and A. Yakaryılmaz. Magic coins are useful for small-space quantum machines. *Quantum Information & Computation*, 17(11&12):1027–1043, 2017.

[37] O. Scegulnaja-Dubrovska and R. Freivalds. A context-free language not recognizable by postselection finite quantum automata. In R. Freivalds, editor, *Randomized and quantum computation*, pages 35–48, 2010. Satellite workshop of MFCS and CSL 2010.

[38] O. Scegulnaja-Dubrovska, L. Lāce, and R. Freivalds. Postselection finite quantum automata. In *Unconventional Computation*, volume 6079 of *Lecture Notes in Computer Science*, pages 115–126, 2010.

[39] A. M. Shur and A. Yakaryılmaz. Quantum, stochastic, and pseudo stochastic languages with few states. In *UCNC 2014*, volume 8553 of *LNCS*, pages 327–339. Springer, 2014.

[40] A. M. Shur and A. Yakaryılmaz. More on quantum, stochastic, and pseudo stochastic languages with few states. *Natural Computing*, 15(1):129–141, 2016.

[41] A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Springer-Verlag, 1994.

[42] P. Turakainen. Generalized automata and stochastic languages. *Proceedings of the American Mathematical Society*, 21:303–309, 1969.

[43] V. S. Vladimirov, I. V. Volovich, and E. I. Zelenov. p-adic analysis and mathematical physics. *World Scientific*, 1995.

[44] A. Yakaryılmaz. Public qubits versus private coins. In *The Proceedings of Workshop on Quantum and Classical Complexity*, pages 45–60. Univeristy of Latvia Press, 2013. ECCC:TR12-130.

[45] A. Yakaryılmaz and A. C. C. Say. Languages recognized by nondeterministic quantum finite automata. *Quantum Information and Computation*, 10(9&10):747–770, 2010.

[46] A. Yakaryılmaz and A. C. C. Say. Succinctness of two-way probabilistic and quantum finite automata. *Discrete Mathematics and Theoretical Computer Science*, 12(2):19–40, 2010.

[47] A. Yakaryılmaz and A. C. C. Say. Probabilistic and quantum finite automata with postselection. Technical Report arXiv:1102.0666, 2011. (A

preliminary version of this paper appeared in the *Proceedings of Randomized and Quantum Computation (satellite workshop of MFCS and CSL 2010)*, pages 14–24, 2010).

[48] A. Yakaryılmaz and A. C. C. Say. Unbounded-error quantum computation with small space bounds. *Information and Computation*, 279(6):873–892, 2011.

[49] A. Yakaryılmaz and A. C. C. Say. Proving the power of postselection. *Fundamenta Informaticae*, 123(1):107–134, 2013.

[50] A. Yakaryılmaz and A. C. C. Say. 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.