University of Latvia
Faculty of Computing

Kaspars Balodis

Unconventional Finite Automata and Algorithms

Doctoral Thesis

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

Scientific Advisor:
Dr. habil. math., Prof. Rūsiņš Mārtiņš Freivalds

Riga, 2016

This work has been supported by the European Social Fund within the project "Support for Doctoral Studies at University of Latvia".

Abstract

In this thesis we investigate several unconventional models of finite automata and algorithms.

We start with more conventional types of automata and prove differentiation results for the descriptional complexity classes of two-way probabilistic and alternating finite automata.

Then we introduce ultrametric finite automata which use p-adic numbers as amplitudes describing the branching process of the computation. We show that they can be more succinct than deterministic automata and show that adding an extra head to an ultrametric automaton increases its computing power. We also examine the size complexity of all the above-mentioned automata for the counting problem.

Finally, we examine two-way frequency finite automata and show their relationship with automata with linearly-bounded counters.

In the second part we study the properties of the computations when the requirements for the algorithms are changed in two different ways.

First, we investigate the query complexity of functions if the algorithms are allowed to make branchings into multiple computational paths with p-adic amplitudes, therefore obtaining so-called ultrametric query algorithms.

Secondly, we generalize the notion of frequency computation by requiring some structure for the correct outputs.

Contents

List of Figures

List of Tables

Introduction

Relevance of the Thesis

Typically computer is viewed as a deterministic machine. However scientists have also extensively studied other models of computation. Some of them are based on nondeterminism and alternation, some of them are intended to be physically realizable or are inspired by natural processes. Such examples include randomized and quantum computing, cellular automata, DNA computing and neural networks. However others are just a mathematically elegant formal systems that describe a process which can be viewed as a computation (lambda calculus, Markov algorithms, Wang tiles, etc.).

There is no globally accepted definition of what type of computation should be considered unconventional. For example the notion of nondeterminism is so widely used in computer science that practically nobody would consider it unconventional. However, as noted by Arora and Barak in their textbook "Computational Complexity: A Modern Approach": "One should note that, unlike standard Turing machines, nondeterministic Turing machines are not intended to model any physically realizable computation device." [2, p. 42] Nevertheless, despite its physical nonrealizability the nondeterministic Turing machine is undoubtedly a very useful concept and in fact one of central in computational complexity theory. Therefore, we can see that the physical realizability of a model of computation is not a determining indicator of conventionality.

Also it is not uncommon in mathematics that a new concept or result is not readily applicable at the time of its discovery, but may find its use several centuries later. As a classical example, it is highly unlikely that the ancient Greek mathematicians when thinking about prime numbers could have imagined their use in cryptography more than 2000 years later. Sometimes results in one field have unexpected applications in other fields. For example, the methods of quantum computing have in many occasions turned out useful to prove classical results in fields that have nothing to do with quantum computing (see [15] for a survey).

Therefore it is not impossible that unconventional models of computation, however unimaginable as physical devices, can later turn out useful in unexpected ways.

Subject and Goals of the Research

Different degrees of unconventionality are possible. Unconventional computation can be mildly unconventional by simply considering a computation in a

setting that is not usually (conventionally) considered, or it might be highly unconventional by introducing some novel, almost unimaginable type of computation.

The goal of the thesis was to examine different unconventional computational models and prove new, previously unknown properties about them. A large part of the thesis is devoted to unconventional finite automata. The thesis focuses on three unconventional types of computation – probabilistic, ultrametric, and frequency. We consider several models of computation with varying degrees of unconventionality, namely:

- probabilistic and alternating two-way finite automata,
- ultrametric finite automata and query algorithms,
- frequency finite automata and structured frequency algorithms.

Arguably the most conventional computation type considered in this thesis is the two-way finite automata. Traditionally finite automata have been considered in one-way setting and with focus on the languages that can be recognized. Indeed, if one cares only about the class of languages recognizable by finite automata then there is no need to consider nondeterministic or alternating automata or two-way versions of them because all of those models can recognize exactly the same class of languages – the class of regular languages. However the situation changes remarkably if we are interested in the state complexity of the automata.

As noted by Kapoutsis, a rich and meaningful complexity theory arises when one considers the size complexity of two-way finite automata. It is quite similar to the Turing machine time and space complexity theory and contains all the standard features of a complexity theory such as complexity classes, reductions, complete problems and even alternating hierarchy. Moreover, the connection to Turing machine complexity theory is not just conceptual – some results about polynomial-size deterministic and nondeterministic automata would directly translate to space-bounded Turing machine complexity theory, specifically the $L \stackrel{?}{=} NL$ question. For an overview of this two-way finite automata size complexity theory, see, for example, [31, 32, 40]. In this thesis we prove a relationship between the two of the least conventional types of automata in this theory, namely the alternating and probabilistic automata.

For every prime number p the p-adic numbers are a completion of the field of rational numbers with respect to the p-adic metric, in the same way as real numbers are a completion of rational numbers with respect to the usual metric. A theorem by Ostrowski shows that every non-trivial absolute value on the rational numbers is equivalent to either the usual real absolute value or a p-adic absolute value. Therefore p-adic numbers can be considered as one of two possible natural extensions of the field of rational numbers (the other one being usual real numbers). p-adic numbers have been used in various ways by theoretical physicists in attempts to understand the nature of fundamental physics. However, the use of p-adic numbers in computational models is a relatively new concept recently introduced by Freivalds. In ultrametric computation p-adic numbers are used to describe the branching of a computation. In this way the p-adic numbers have been used to define ultrametric automata, ultrametric Turing machines, ultrametric query algorithms, and ultrametric learning

algorithms. The ultrametric computations, unlike deterministic, randomized or even quantum computations, are not meant to be realizable as physical devices, at least we are not aware of a practical way to implement them. Nevertheless, they are equally interesting as an abstract computational model. In this thesis we define ultrametric finite automata and consider various properties that arise out of their definition. Additionally we introduce the model of ultrametric query algorithms which is similar to probabilistic and quantum query algorithms. We prove results about ultrametric query complexity of Boolean functions.

Frequency computation is a notion that was introduced in 1960 by G. Rose. A frequency algorithm (m, n)-computes a function f if given any n distinct inputs $x_1, \ldots, x_n$ it produces n outputs $y_1, \ldots, y_n$ for which at least m of the equations $y_i = f(x_i)$ hold, i.e., the algorithm is required to output the correct answer on at least m of n different inputs. While for some this notion might already seem rather unconventional by itself, we modify it in two different ways therefore imparting an additional degree of unconventionality. The first way involves finite automata. While frequency finite automata have already been considered earlier, they have been considered only in the setting of one-way automaton which simultaneously reads a symbol from each of the input words. We introduce two-way frequency finite automata – a model which has not been considered earlier in the literature. We show their relationship with finite automata with linearly bounded counters and the complexity class *LOGSPACE*.

The other modification is changing the definition of frequency computation by requiring some structure for the correct answers therefore obtaining the so-called structured frequency computation. It is also a new model which has not been considered previously. In this setting we examine which structures allow the computing of recursive sets and which – nonrecursive. We also investigate graph frequency computation where the size of the structures is limited to 2 so they can be represented as the edges of a graph.

The Structure of the Thesis and Research Questions

The research questions, proposed hypotheses and proven theorems differ from chapter to chapter, however most of them try to compare unconventional computation models with other better known models.

The thesis contains six chapters divided into two parts. Each chapter is devoted to one unconventional computation model. Part I considers finite automata.

In the first chapter we consider two-way probabilistic and alternating automata which are the most conventional computation type considered in this thesis. In this chapter we do not define new computation models, but instead work in a theoretical framework introduced by Sakoda and Sipser and further developed by Kapoutsis, Královič and others. We shortly describe the complexity theory of two-way automata and give definitions for two-way probabilistic and alternating finite automata and the corresponding complexity classes. Then we prove the main result of the chapter – show a language that can be recognized with a linear-size one-way alternating automaton, but cannot be recognized with any polynomial-size probabilistic two-way automaton. This is a previously un-

known, novel result and it shows that the corresponding complexity classes $2P_2$ and $2A$ are not equal.

In the second chapter we consider ultrametric finite automata. We describe the p-adic numbers and give definitions for ultrametric finite automata and investigate their properties. We show that with a regulated ultrametric automaton only regular languages can be recognized and we give a bound on the number of states for an equivalent deterministic automaton. We show that for some languages ultrametric automata can be smaller than deterministic automata. We also compare different definitions for ultrametric automata. In the second part of the chapter we consider multi-head ultrametric automata. We show that ultrametric one-way automata can recognize languages not recognizable by any k-head nondeterministic automata. For two-way automata we prove that adding an extra head to the automaton increases the class of recognizable languages.

In the third chapter we focus on counting with finite automata by examining how many states are needed for different models of automata to recognize the language $C_n = \{1^n\}$. At first we list known results about deterministic, nondeterministic and alternating finite automata, and then proceed to prove new results about probabilistic and ultrametric finite automata. We show an optimal 3-state one-way probabilistic automaton. We show how to construct a constant-sized two-way probabilistic automaton with the number of states not depending on the required probability for the correct answer. We also show an optimal 2-state ultrametric automaton.

In the fourth chapter we consider two-way frequency automata. We introduce their definition and show that with frequency (m, n) it is possible to recognize any language recognizable with a two-way automaton with $n - m$ linearly bounded counters. We also show that any language from the class $LOGSPACE$ can be recognized by some frequency automaton.

Part II is devoted to unconventional algorithms. In the fifth chapter we introduce a new type of algorithms – ultrametric query algorithms which use p-adic numbers to describe the branching of an algorithm. We show results about the ultrametric query complexity. We show that the exact ultrametric query complexity is at most the polynomial degree of a Boolean function. For several functions we show ultrametric query algorithms with complexity 1.

In the sixth chapter we introduce structured frequency computation. We prove that with overlapping structures only recursive sets can be recognized and show that the size of overlapping structures is at least $\sqrt{n}$. We also show how to construct structures which nearly achieves this bound by using projective planes. In the second part of the chapter we consider the special case of graph structures and categorize which graphs allow only computation of recursive sets.

In conclusion we list some open problems and further research directions in each of the considered topics.

Description of the Methodology

As usual in mathematical and theoretical computer science papers, the used methods are mainly proofs of mathematical statements (theorems). Different proof methods are used, ranging from simple to relatively sophisticated. Worth mentioning is Theorem 6.20 whose proof is computer-assisted. An exhaustive search was used to examine all 7-vertex rooted trees and all possible ways to

distribute their non-root vertices into pairs.

We also introduce new computation models with precise mathematical definitions. In this way we define ultrametric finite automata, frequency finite automata, ultrametric query algorithms, and structured frequency computation.

Approbation of the Results

The results of this thesis are published in 7 publications of which 4 are indexed in Elsevier Scopus and 3 in Thomson Reuters Web of Science.

1. Kaspars Balodis, Jānis Iraids, Rūsiņš Freivalds.
   Structured Frequency Algorithms.
   Proceedings of TAMC 2015, Lecture Notes in Computer Science, vol. 9076, pp. 1–12, 2015. *(indexed in Scopus and Web of Science)*
   Approximate contribution of the author: 60-80%

2. Rihards Krišlauks, Kaspars Balodis.
   On the Hierarchy Classes of Finite Ultrametric Automata.
   Proceedings of SOFSEM 2015: Theory and Practice of Computer Science, Lecture Notes in Computer Science, vol. 8939, pp. 314–326, 2015. *(indexed in Scopus and Web of Science)*
   Approximate contribution of the author: 25-50%

3. Kaspars Balodis.
   Counting with Probabilistic and Ultrametric Finite Automata.
   Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday, Lecture Notes in Computer Science, vol. 8808, pp. 1–14, 2014. *(indexed in Scopus and Web of Science)*
   Approximate contribution of the author: 100%

4. Kārlis Jēriņš, Kaspars Balodis, Rihards Krišlauks, Kristīne Cīpola, Rūsiņš Freivalds.
   Ultrametric query algorithms.
   Proceedings of SOFSEM 2014: Theory and Practice of Computer Science, Volume II: Student Research Forum, pp. 21–29, ISBN 978-80-8152-085-3, 2014.
   Approximate contribution of the author: 15-25%

5. Kaspars Balodis.
   One Alternation Can Be More Powerful Than Randomization in Small and Fast Two-Way Finite Automata.
   Proceedings of FCT 2013, Lecture Notes in Computer Science, vol. 8070, pp. 40–47, 2013. *(indexed in Scopus)*
   Approximate contribution of the author: 100%

6. Kaspars Balodis, Anda Beriņa, Kristīne Cīpola, Maksims Dimitrijevs, Jānis Iraids, Kārlis Jēriņš, Vladimirs Kacs, Jānis Kalējs, Rihards Krišlauks, Kārlis Lukstiņš, Reinholds Raumanis, Natālija Somova, Irina Ščeguļnaja, Anna Vanaga, Rūsiņš Freivalds.
   On the State Complexity of Ultrametric Finite Automata.

Proceedings of SOFSEM 2013: Theory and Practice of Computer Science, Volume II: Student Research Forum, pp. 1–9, ISBN 978-80-87136-15-7, 2013.
Approximate contribution of the author: 40-50%

7. Kaspars Balodis, Anda Beriņa, Gleb Borovitsky, Rūsiņš Freivalds, Ginta Garkāje, Vladimirs Kacs, Jānis Kalējs, Ilja Kucevalovs, Jānis Ročāns, Madars Virza.
Probabilistic and Frequency Finite-State Transducers.
Proceedings of SOFSEM 2012: Theory and Practice of Computer Science, Volume II: Student Research Forum, pp. 1–12, ISBN 978-80-87136-13-3, 2012.
Approximate contribution of the author: 40-50%

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

1. SOFSEM 2015 (41st International Conference on Current Trends in Theory and Practice of Computer Science), Pec pod Sněžkou, Czech Republic, 2015.
Presentation: *On the Hierarchy Classes of Finite Ultrametric Automata.*

2. Joint Estonian-Latvian Theory Days, Ratnieki, Latvia, 2014.
Presentation: *Structured Frequency Algorithms.*

3. Latvijas Universitātes 72. konference, Rīga, Latvia, 2014.
Presentation: *Par divvirzienu galīgiem automātiem.*

4. SOFSEM 2014 (40th International Conference on Current Trends in Theory and Practice of Computer Science), Nový Smokovec, High Tatras, Slovakia, 2014.
Poster presentation: *Ultrametric query algorithms.*

5. FCT 2013 (19th International Symposium on Fundamentals of Computation Theory), Liverpool, United Kingdom, 2013.
Presentation: *One Alternation Can Be More Powerful Than Randomization in Small and Fast Two-Way Finite Automata.*

6. SOFSEM 2013 (39th International Conference on Current Trends in Theory and Practice of Computer Science), Špindlerův Mlýn, Czech Republic, 2013.
Poster presentations: *On the State Complexity of Ultrametric Finite Automata* and *Ultrametric Turing Machines with Limited Reversal Complexity.*

7. Latvijas Universitātes 70. konference, Rīga, Latvia, 2012.
Presentation: *Par galīgiem automātiem uz bezgalīgas lentas.*

8. SOFSEM 2012 (38th International Conference on Current Trends in Theory and Practice of Computer Science), Špindlerův Mlýn, Czech Republic, 2012.
Poster presentation: *Probabilistic and Frequency Finite-State Transducers.*

The author during his doctoral studies has also participated in the following publications not directly related to the thesis (in these the contribution of the author typically fluctuates around $\frac{1}{n}$ where n is the number of authors of the paper). 4 of these publications are indexed in Elsevier Scopus and 3 in Thomson Reuters Web of Science.

1. Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, Juris Smotrovs.
   Separations in Query Complexity Based on Pointer Functions.
   arXiv preprint, arXiv:1506.04719, 24 p., 2015.

2. Kaspars Balodis, Jānis Iraids.
   Quantum Lower Bound for Graph Collision Implies Lower Bound for Triangle Detection.
   arXiv preprint, arXiv:1507.03885, 4 p., 2015.

3. Scott Aaronson, Andris Ambainis, Kaspars Balodis, Mohammad Bavarian.
   Weak Parity.
   Proceedings of ICALP 2014, Lecture Notes in Computer Science, vol. 8572, pp. 26–38, 2014. *(indexed in Scopus and Web of Science)*

4. Ilja Kucevalovs, Kaspars Balodis, Rūsiņš Freivalds.
   Co-learning of Functions by Probabilistic Algorithms.
   Proceedings of 2nd International Symposium on Computer, Communication, Control and Automation, pp. 71–73, Atlantis Press, 2013. *(indexed in Web of Science)*

5. Andris Ambainis, Kaspars Balodis, Jānis Iraids, Raitis Ozols, Juris Smotrovs.
   Parameterized Quantum Query Complexity of Graph Collision.
   Proceedings of Workshop on Quantum and Classical Complexity, pp. 5–16, ISBN 978-9984-45-743-7, 2013.

6. Andris Ambainis, Artūrs Bačkurs, Kaspars Balodis, Juris Smotrovs, Agnis Škuškovniks, Madars Virza.
   Worst case analysis of non-local games.
   Proceedings of SOFSEM 2013: Theory and Practice of Computer Science, Lecture Notes in Computer Science, vol. 7741, pp. 121–132, 2013. *(indexed in Scopus)*

7. Jānis Iraids, Kaspars Balodis, Juris Čerņenoks, Mārtiņš Opmanis, Rihards Opmanis, Kārlis Podnieks.
   Integer Complexity: Experimental and Analytical Results.
   Scientific Papers University of Latvia, Computer Science and Information Technologies, vol. 787, pp. 153–179, 2012.

8. Andris Ambainis, Artūrs Bačkurs, Kaspars Balodis, Dmitrijs Kravčenko, Raitis Ozols, Juris Smotrovs, Madars Virza.
   Quantum Strategies Are Better Than Classical in Almost Any XOR Game.
   Proceedings of ICALP 2012, Lecture Notes in Computer Science, vol. 7391, pp. 25–37, 2012. *(indexed in Scopus and Web of Science)*

9. Kaspars Balodis, Ilja Kucevalovs, Rūsiņš Freivalds.
Frequency prediction of functions.
Proceedings of MEMICS 2011, Lecture Notes in Computer Science, vol. 7119, pp. 76–83, 2012. *(indexed in Scopus)*

Part I

Finite Automata

Overview of Part I

This part of the thesis is devoted to finite automata. Starting with more conventional types of automata, in Chapter 1 we work in a theoretical framework of two-way finite automata complexity theory introduced by Sakoda and Sipser and developed by Kapoutsis and others. We differentiate the complexity classes of families of languages recognized by polynomial-size alternating and probabilistic families of automata.

In Chapter 2 we introduce ultrametric finite automata and show that they can be more concise than their classical counterparts. We also show a hierarchy of languages recognized by multi-head ultrametric finite automata. In Chapter 3 the state complexity is compared for different types of automata for the counting problem, i.e., the problem of recognizing the one-word unary language $L_n = \{1^n\}$. In Chapter 4 we introduce two-way frequency finite automata and prove their relationship with automata with linearly-bounded counters.

Chapter 1

Classical Automata

In this chapter we work in a theoretical framework of state complexity of two-way finite automata introduced by Sakoda and Sipser and developed by Kapoutsis. Two-way finite automata are considered with the focus on the increase of the number of states for recognizing a language family $\mathcal{L} = (L_1, L_2, L_3, \dots)$. Special interest is given to whether this increase can be bounded by a polynomial.

As the main result we show a family of languages which can be recognized by a family of linear-size one-way alternating automata (making only 1 alternation), but for every family of two-way bounded-error probabilistic automata which work in a polynomial time the increase of the number of states is super-polynomial.

The chapter is organized as follows. In Section 1.1 we give an introduction and definitions. In Section 1.2 we introduce the aforementioned two-way finite automata complexity theory. In Section 1.3 we present the main result. In Section 1.4 we show some results in the opposite direction, and we end the chapter by giving some short conclusions in Section 1.5.

1.1 Introduction and Definitions

Finite automata is a well-known model of computation and has many practical applications in computer science, including string processing algorithms, programming languages, network protocols, and others.

On a theoretical level, finite automata represent a basic model of computer with a finite memory.

We use more or less standard definitions for the finite automata [56]. We give the definition of a more general two-way alternating finite automaton and the nondeterministic and deterministic automata (and the one-way versions) can be derived by adding restrictions to the transition function. Following [24] we define it as follows:

Definition 1.1. *A two-way alternating finite automaton (2AFA) is a tuple* $\mathcal{M} = (Q_\exists, Q_\forall, \Sigma, \delta, q_0, F)$*, where*

> $Q_\exists$ *and* $Q_\forall$ *are finite sets of existential and universal states, respectively, with* $Q_\exists \cap Q_\forall = \emptyset$ *and* $Q = Q_\exists \cup Q_\forall$*,*
>
> Σ *is the input alphabet,*

$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{L,N,R\}}$ *is the transition function, where* $\vdash, \dashv \notin \Sigma$ *are the left and right endmarkers, and L, N, R denote the head movement,*

$q_0 \in Q$ *is the starting state, and*

$F \subseteq Q$ *is the set of accepting states (also called final states).*

The input word w is presented to the automaton enclosed by the endmarkers as $\vdash w \dashv$. The machine starts in the state q_0 with the input head positioned on the first symbol of w. In each step $\mathcal{M}$ reads an input symbol, changes its state, and moves the input head one position to the left, right, or keeps it stationary, depending on whether δ returns L, R, or N, respectively. The automaton is neither allowed to move the head to the left while reading $\vdash$, nor allowed to move the head to the right while reading $\dashv$. The only exception is a transition from $\dashv$ to an accepting state and moving the head to the right. An individual computation path is considered as accepting if it falls off to the right of the endmarker $\dashv$ in an accepting state. If a computation path reaches a configuration from which δ admits no valid transition then it is considered rejecting. Similarly computation paths that lead to infinite loop are considered rejecting as well.

The global rules for accepting are defined as is usual for alternating devices: if, at the given moment, the function δ admits to execute several transitions, the machine (i) nondeterministically chooses one of them, if it is in an existential state, but (ii) follows, in parallel, all possible computation paths, if the current state is universal. By nondeterminism of (i), there may exist several different computations for the same input. By (ii), the computation forks into parallel processes. The input is accepted, if the nondeterministically chosen computation, starting in q_0 at the beginning of the input word, forms an accepting computation subtree of parallel branches, embedded in the full tree of all possible computation paths, such that all branches in the subtree halt in accepting states.

The automaton $\mathcal{M}$ is said to be one-way (1AFA) if its input head motions are restricted to R and N. For one-way machines, we usually do not embed the input between endmarkers, but we require the machine to halt in accepting states for all accepting computations after reading the entire input.

An alternation is a computation step in which the automaton switches from a state $q \in Q_\exists$ to a state $q' \in Q_\forall$ or from a state $q \in Q_\forall$ to a state $q' \in Q_\exists$.

A nondeterministic automaton (2NFA) is a special case of 2AFA $\mathcal{M} = (Q_\exists, Q_\forall, \Sigma, \delta, q_0, F)$ for which $Q_\forall = \emptyset$. A deterministic automaton (2DFA) is a special case of 2NFA for which the transition function δ assigns at most one successor state and movement direction for each state and input symbol. Similarly the automata are one-way (1NFA and 1DFA, respectively) if their input head motions are restricted to R and N.

Probabilistic (one-way) finite automata (1PFAs) were introduced by Rabin in [53]. Again, we start with a more general two-way definition:

Definition 1.2. *A two-way probabilistic finite automaton (2PFA) is a tuple* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *where*

Q *is the finite set of states,*

Σ *is the input alphabet,*

$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \times Q \times \{L, N, R\} \to P$ is the probabilistic transition function, $\vdash, \dashv \notin \Sigma$ are the left and right endmarkers, respectively, and L, N, R denote the head movement, and $P \subseteq [0, 1]$ is the set of allowed probabilities,

$q_0 : Q \to P$ is the starting distribution, and

$F \subseteq Q$ is the set of accepting states.

For 2PFAs the transition function $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \times Q \times \{L, N, R\} \to P$ assigns for each possible transition the probability of making this transition and $q_0 : Q \to P$ shows the starting probability distribution.

If $P = \mathbb{Q} \cap [0, 1]$, where $\mathbb{Q}$ is the set of rational numbers, then we call the automaton *rational*. If $P = \{0, \frac{1}{2}, 1\}$, then we call the automaton *coin-flipping*.

The 2PFA $\mathcal{M}$ is said to be one-way (1PFA) if its input head motions are restricted to R and N.

To have a meaningful acceptance condition, we allow the automaton to "fall off" the right endmarker $\dashv$ into any state q and say that the current run accepts the word if $q \in F$ and rejects otherwise. Infinite loops are also considered rejecting. Let $\mathcal{A}(x)$ denote the probability that a PFA $\mathcal{A}$ accepts the word x. We say that a PFA $\mathcal{A}$ recognizes language L with cutpoint $\lambda \in [0, 1]$ if $\forall x \in L\ \mathcal{A}(x) > \lambda$ and $\forall x \notin L\ \mathcal{A}(x) < \lambda$. We say that a PFA $\mathcal{A}$ recognizes language L with an isolated cutpoint $\lambda \in [0, 1]$ if there exists $\delta > 0$ (called isolation radius) such that $\forall x \in L\ \mathcal{A}(x) > \lambda + \delta$ and $\forall x \notin L\ \mathcal{A}(x) < \lambda - \delta$. The PFAs with isolated cutpoint are called $\mathrm{P_2FAs}$.

1.2 Complexity Theory

The increase in complexity when going from a deterministic to a nondeterministic model has been an important topic in computer science. Most notorious of the kind is the question about the time complexity of Turing machines, i.e., $P \stackrel{?}{=} NP$ question.

Recently there has been some increased interest in descriptional complexity, particularly in the field of size complexity of two-way finite automata (2FAs). It originated in a question raised by Sakoda and Sipser in 1978 about the increase in the number of states for a two-way deterministic finite automaton (2DFA) to simulate a one-way or two-way nondeterministic automaton (1NFA or 2NFA) [55]. It was conjectured that the increase is not polynomial. Moreover, it is believed that 2^n states are necessary in the worst case for a 2DFA to simulate an n-state 1NFA. However, the best known separation is only $\Omega(n^2)$ (and it is achieved by a unary language [12]).

Sakoda and Sipser introduced an elegant theoretical framework for describing the complexity classes of families of languages recognized by families of small (polynomial-size) two-way deterministic and nondeterministic finite automata. In recent papers [31, 32] Kapoutsis has extended the framework to include alternating, probabilistic and other classes, as well as systematized the research in the size complexity theory of two-way finite automata (or Minicomplexity – a term coined in [32]). He showed that a rich complexity theory emerges with all the standard features of a complexity theory including complexity classes, reductions, completeness, etc., which mimics the complexity theory of time- or

space-bounded Turing machines (see Fig. 1.1). The central question of this theory is the Sakoda-Sipser conjecture, which corresponds to the $P \stackrel{?}{=} NP$ question for Turing machines and time, or to the $L \stackrel{?}{=} NL$ question for Turing machines and space.

It is also been known for a long time that answering the Sakoda-Sipser conjecture in the negative using only polynomially long instances would imply that $L \neq NL$ [8].

Following this framework, instead of individual languages and finite automata we consider families of languages and families of finite automata. Analogous to the time bound of Turing machines depending on the input size is the state-size bound for 2FAs depending on the index of the language.

A simplification is to consider promise problems instead of languages. It allows to give more emphasis to the core of the problem instead of technicalities, for example because it is not necessary for the automata to check if the input is in the correct form and reject ill-formated words, thereby allowing the automata to be simpler and more elegant. However, this is just for convenience – all of our results stay the same also if we consider languages instead of promise problems.

Definition 1.3. *A (promise) problem over Σ is a pair $\mathcal{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. Every $w \in L \cup \tilde{L}$ is an instance of $\mathcal{L}$: positive, if $w \in L$; or negative, if $w \in \tilde{L}$. To solve $\mathcal{L}$ is to accept all $w \in L$ but no $w \in \tilde{L}$.*

We call $\mathcal{L}' = (L', \tilde{L}')$ a subproblem of $\mathcal{L} = (L, \tilde{L})$ if $L' \subseteq L$ and $\tilde{L}' \subseteq \tilde{L}$.

The complement of a problem $\mathcal{L} = (L, \tilde{L})$ is the problem $\neg\mathcal{L} = (\tilde{L}, L)$. The complement of a family of problems $(\mathcal{L}_h)_{h\geq 1}$ is the family $(\neg\mathcal{L}_h)_{h\geq 1}$.

A problem $\mathcal{L}$ over Σ is regular if there exists a regular language $R \subseteq \Sigma^*$ such that $\mathcal{L}$ is a subproblem of $(R, \Sigma^* \setminus R)$.

A family of automata $(\mathcal{M}_h)_{h\geq 1}$ solves a family of problems $(\mathcal{L}_h)_{h\geq 1}$ if for each $h \geq 1$ the automaton $\mathcal{M}_h$ solves $\mathcal{L}_h$.

The automata are called small if there is a polynomial p such that for each h the automaton $\mathcal{M}_h$ has at most $p(h)$ states.

2D is the class of families of problems solvable by a family of small 2DFAs:

$$2D = \left\{ (\mathcal{L}_h)_{h\geq 1} \,\middle|\, \begin{array}{c}\text{there exist 2DFAs } (\mathcal{M}_h)_{h\geq 1} \text{ and polynomial } p \text{ such that} \\ \mathcal{M}_h \text{ solves } \mathcal{L}_h \text{ with at most } p(h) \text{ states, for all } h\end{array} \right\}$$

2N is the class of families of problems solvable by a family of small 2NFAs and co-2N is the class of families of problems whose complements are solvable by a family of small 2NFAs. Analogous classes for one-way automata are 1D, 1N and co-1N.

Therefore the above mentioned problem about 2DFAs and 2NFAs can be reformulated as $2D \stackrel{?}{=} 2N$.

Definition 1.4.

$$2A = \left\{ (\mathcal{L}_h)_{h\geq 1} \,\middle|\, \begin{array}{c}\textit{there exist 2AFAs } (M_h)_{h\geq 1} \textit{ and polynomial } p \textit{ such that} \\ M_h \textit{ solves } \mathcal{L}_h \textit{ using at most } p(h) \textit{ states for all } h\end{array} \right\}$$

For all $k \geq 1$:

$$2\Sigma_k = \left\{ (\mathcal{L}_h)_{h\geq 1} \,\middle|\, \begin{array}{c}\textit{there exist 2AFAs } (M_h)_{h\geq 1} \textit{ and polynomial } p \textit{ such that} \\ M_h \textit{ solves } \mathcal{L}_h \textit{ starting in an existential state and using} \\ \textit{at most } k-1 \textit{ alternations and } p(h) \textit{ states for all } h\end{array} \right\}$$

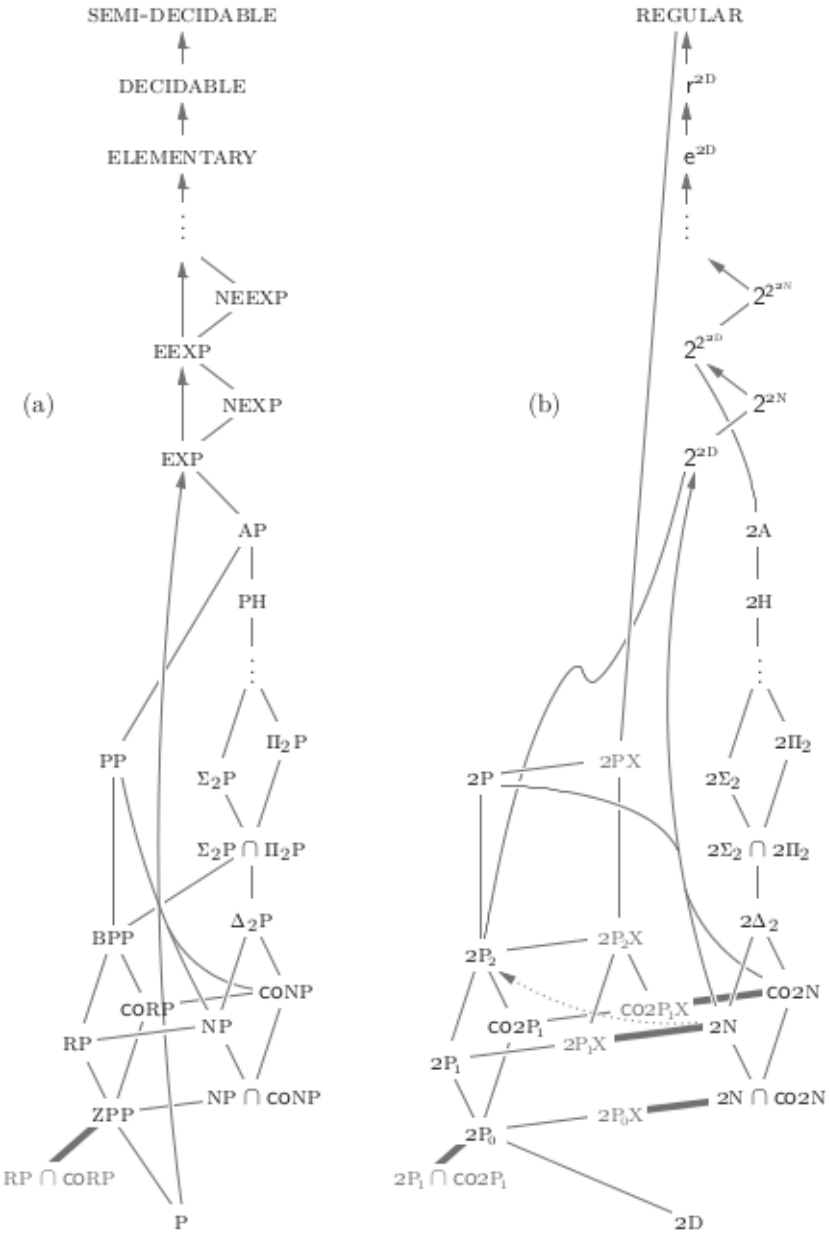

Figure 1.1: 2FA complexity theory (b) and its similarity to time-bounded Turing machine complexity theory (a). A bold line $C - C'$ means $C = C'$; a simple line from C upwards to C' means $C \subseteq C'$; an arrow $C \to C'$ means $C \subsetneq C'$; a dotted arrow $C \dashrightarrow C'$ means $C \not\supseteq C'$. Image by C. A. Kapoutsis [31].

$$2\Pi_k = \left\{ (\mathcal{L}_h)_{h\geq 1} \middle| \begin{array}{c} \textit{there exist 2AFAs } (M_h)_{h\geq 1} \textit{ and polynomial } p \textit{ such that} \\ M_h \textit{ solves } \mathcal{L}_h \textit{ starting in an universal state and using} \\ \textit{at most } k-1 \textit{ alternations and } p(h) \textit{ states for all } h \end{array} \right\}$$

$1\Sigma_k$ and $1\Pi_k$ denote the corresponding classes for one-way automata.

Recently it was proved in [24] that the alternating hierarchy of 2AFAs does not collapse – for each $k \geq 2$ both $2\Sigma_{k-1}$ and $2\Pi_{k-1}$ are proper subsets of $1\Sigma_k$ and $1\Pi_k$ (and thus also of $2\Sigma_k$ and $2\Pi_k$) (as it was proved after the overview in [31], it is not depicted in Fig. 1.1).

We say that a 2PFA $\mathcal{M}$ solves a problem $\mathcal{L} = (L, \tilde{L})$ if for each $w \in L$ the probability of $\mathcal{M}$ accepting w is $> \frac{1}{2}$ and for each $w \in \tilde{L}$ the probability of $\mathcal{M}$ accepting w is $< \frac{1}{2}$.

If the difference between the probabilities of $\mathcal{M}$ accepting positive and negative instances of $\mathcal{L}$ are significant in the length of the input word, i.e., there exists a polynomial r such that for each $w \in L$ the probability of $\mathcal{M}$ accepting w is $\geq \frac{1}{2} + \frac{1}{r(|w|)}$ and for each $w \in \tilde{L}$ the probability of $\mathcal{M}$ accepting w is $\leq \frac{1}{2} - \frac{1}{r(|w|)}$, then we say that $\mathcal{M}$ accepts $\mathcal{L}$ with bounded error and isolated cutpoint with isolation radius $\frac{1}{r(|w|)}$. We call such automaton $2P_2FA$. For a family of problems $(\mathcal{L}_h)_{h\geq 1}$ to be solved with bounded error we require the isolation radius to be significant in h as well.

We say that a 2PFA $\mathcal{M}$ is fast if there exists a polynomial p such that for each word $w \in \Sigma^*$ the expected running time of $\mathcal{M}$ is upper-bounded by $p(|w|)$.

The corresponding complexity class of problems solvable by a family of small and fast rational $2P_2FA$s is $2P_2$.

Definition 1.5.

$$2P_2 = \left\{ (\mathcal{L}_h)_{h\geq 1} \middle| \begin{array}{c} \textit{there exist rational } 2P_2FAs \ (M_h)_{h\geq 1} \textit{ and polynomials } p, q, r \textit{ such that} \\ M_h \textit{ solves } \mathcal{L}_h \textit{ with isolation radius } \frac{1}{r(h,n)} \textit{ using at most } p(h) \textit{ states and} \\ q(h,n) \textit{ steps on average, for all } h \textit{ and all } n \textit{ and all } n\textit{-long instances} \end{array} \right\}$$

The corresponding class for one-way automata is $1P_2$.

We also define two classes for less restricted types of 2PFAs. The complexity class for unrestricted runtime is $2P_2X$. If we drop the requirement for the cutpoint to be isolated then the corresponding complexity class is 2P. However now we explicitly have to ask for all the families to contain only regular problems.

Definition 1.6.

$$2P_2X = \left\{ (\mathcal{L}_h)_{h\geq 1} \middle| \begin{array}{c} \textit{there exist rational } 2P_2FAs \ (M_h)_{h\geq 1} \textit{ and polynomials } p, r \textit{ such that} \\ M_h \textit{ solves } \mathcal{L}_h \textit{ with isolation radius } \frac{1}{r(h,n)} \textit{ using at most } p(h) \textit{ states,} \\ \textit{for all } h \textit{ and all } n \textit{ and all } n\textit{-long instances, and each } \mathcal{L}_h \textit{ is regular} \end{array} \right\}$$

$$2P = \left\{ (\mathcal{L}_h)_{h\geq 1} \middle| \begin{array}{c} \textit{there exist rational } 2PFAs \ (M_h)_{h\geq 1} \textit{ and polynomials } p, q \textit{ such that} \\ M_h \textit{ solves } \mathcal{L}_h \textit{ using at most } p(h) \textit{ states and } q(h,n) \textit{ steps on average,} \\ \textit{for all } h \textit{ and all } n \textit{ and all } n\textit{-long instances, and each } \mathcal{L}_h \textit{ is regular} \end{array} \right\}$$

Again, the corresponding class for one-way automata is 1P.

1.3 Alternation over Randomization

Randomness has always been an important concept in computer science. It is a long standing question in which settings randomization adds power to computation. The most fundamental open question of this kind is whether the class BPP of languages recognized in polynomial time by bounded-error probabilistic

Turing machines is equal to the class P of languages recognized in polynomial time by deterministic Turing machines.

In the finite automata setting, some results are known that show the advantage of probabilistic automata over some classes of non-randomized automata. It started with Freivalds' surprising result that a $2P_2FA$ can recognize a non-regular language $L = \{0^n1^n \mid n \geq 1\}$ [18]. Later Greenberg and Weiss showed [25] that an expected runtime of $2^{\Omega(n)}$ is needed for any $2P_2FA$ recognizing this language, and Dwork and Stockmeyer proved [16] that a superpolynomial expected runtime is needed for a $2P_2FA$ to recognize any non-regular language, and when restricted to a polynomial runtime the class of languages recognized by $2P_2FAs$ is the class of regular languages. However, they also showed that polynomial-time $2P_2FA$ can be more succinct than any 2NFA. $O\left(\frac{log^2 n}{loglogn}\right)$ states are sufficient for a polynomial-time $2P_2FA$ to recognize language $L_n = \{1^n\}$, but any 2NFA requires at least n states. Kapoutsis et al. showed [33] that for any $n \geq 2$ there exists a language that is recognized by an $O(n^2)$-state sweeping Las Vegas automata but needs $2^{\Omega(n)}$ states for any sweeping deterministic automaton. In [61] a family of languages is shown that can be recognized by a $2P_2FA$ with a constant number of states (the automata differ only in the transition probabilities), but for any equivalent 2NFA the necessary number of states grows without a bound.

It seems that the results of this section of the thesis are ones of the few showing advantage in the opposite direction – that a class of non-randomized finite automata can be more powerful than a class of probabilistic finite automata. Namely, we show two families of problems, one which is solvable by a family of 1AFAs $(\mathcal{A}_h)_{h\geq 1}$ of size $2h+3$ starting in an existential state and using 1 alternation and another one solvable by a family of 1AFAs $(\mathcal{A}'_h)_{h\geq 1}$ of size $2h+3$ starting in a universal state and using 1 alternation. We show that that for neither of the problems does there exist a family of small and fast $2P_2FAs$ or small 2NFAs solving them or their complements. Therefore, in the terms of two-way finite automata complexity theory, neither $1\Sigma_2$ nor $1\Pi_2$ is contained in $2P_2$, $2N$ or co-$2N$. We should note that the results about $2N$ and co-$2N$ also follow from the more general result of [24] showing that the entire alternating hierarchy does not collapse.

Definition 1.7. *Let $f(x_1, \cdots, x_n)$ be a Boolean function and $(y_{11} \wedge y_{12} \wedge \cdots \wedge y_{1l_1}) \vee (y_{21} \wedge y_{22} \wedge \cdots \wedge y_{2l_2}) \vee \cdots \vee (y_{m1} \wedge y_{m2} \wedge \cdots \wedge y_{ml_m})$ its disjunctive normal form (DNF) where m is the number of clauses, l_i $(1 \leq i \leq m)$ is the number of literals in the i-th clause, and each y_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq l_i)$ is either x_k or $\overline{x_k}$ for some $1 \leq k \leq n$. Assume that each variable appears in a clause no more than once and there are no more than 2^n clauses.*

Then we call `[`$Z_{11}Z_{12}\ \cdots\ Z_{1n}$`]``[`$Z_{21}Z_{22}\ \cdots\ Z_{2n}$`]`$\cdots$`[`$Z_{m1}Z_{m2}\ \cdots\ Z_{mn}$`]` *a DNF-encoding of f where*

$$Z_{ik} = \begin{cases} \texttt{x} & \text{if } x_k \text{ appears in the } i\text{-th clause as } x_k \\ \bar{\texttt{x}} & \text{if } x_k \text{ appears in the } i\text{-th clause as } \overline{x_k} \\ \texttt{-} & \text{if } x_k \text{ does not appear in the } i\text{-th clause.} \end{cases}$$

Denote by DEC_{DNF} the function that maps a DNF-encoding to its function.

For example, a DNF-encoding of $f(x_1, x_2, x_3, x_4, x_5) = (x_1 \wedge \overline{x_2} \wedge x_4 \wedge x_5) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_4 \wedge \overline{x_5})$ is `[`$\texttt{x}\bar{\texttt{x}}\texttt{-xx}$`]``[`$\texttt{x-x--}$`]``[`$\texttt{-}\bar{\texttt{x}}\texttt{-x}\bar{\texttt{x}}$`]`. Note that there might be

many DNF-encodings of the same function, but a DNF-encoding unambiguously defines a function.

Next we define a promise problem over $\Sigma = \{\texttt{[}, \texttt{]}, \texttt{x}, \bar{\texttt{x}}, \texttt{-}, \texttt{.}, \texttt{0}, \texttt{1}\}$ with the following intuitive meaning: given a DNF and an assignment for its variables, check that the value of the DNF under this assignment is 1.

Definition 1.8. EVALUATE-DNF-FUNCTION$_h$ =
$(\{F.x_1 \dots x_h \mid \exists f : DEC_{DNF}(F) = f \wedge \forall i\, x_i \in \{0, 1\} \wedge f(x_1, \dots, x_h) = 1\},$
$\{F.x_1 \dots x_h \mid \exists f : DEC_{DNF}(F) = f \wedge \forall i\, x_i \in \{0, 1\} \wedge f(x_1, \dots, x_h) = 0\})$
EVALUATE-DNF-FUNCTION = $(\text{EVALUATE-DNF-FUNCTION}_h)_{h \geq 1}$

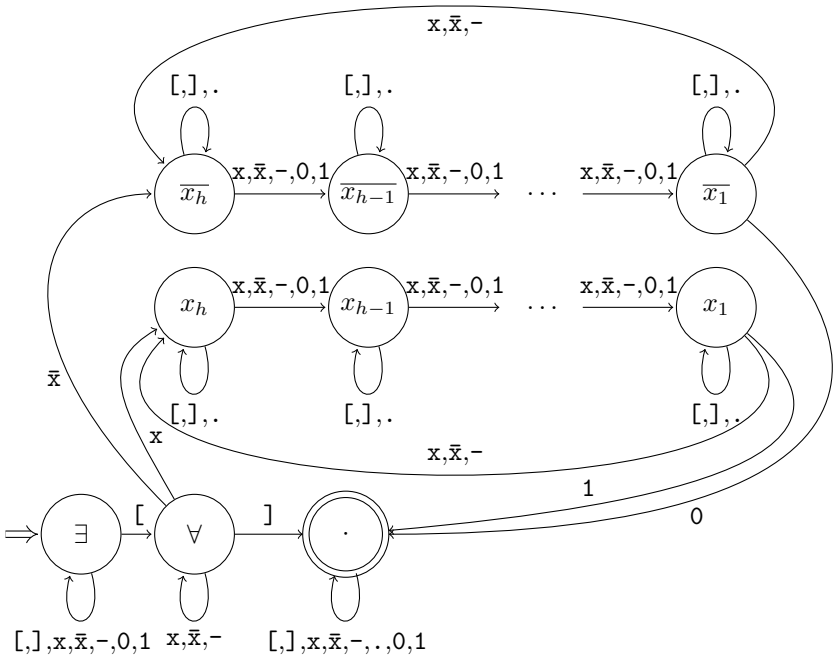

Figure 1.2: One-way alternating automaton $\mathcal{A}_h$ with $2h + 3$ states solving EVALUATE-DNF-FUNCTION$_h$. Double arrow shows the starting state. State $\exists$ is existential and all other states are universal. Double circled state is accepting.

Theorem 1.9. *There exists a family of 1AFAs $(\mathcal{A}_h)_{h \geq 1}$ with $2h+3$ states that solves the problem* EVALUATE-DNF-FUNCTION *starting in an existential state and using 1 alternation.*

Proof. See the automaton $\mathcal{A}_h$ in Fig. 1.2. After a careful examination it should be evident that $\mathcal{A}_h$ solves EVALUATE-DNF-FUNCTION$_h$. It starts by reading in each clause of the DNF and "saving" it in a configuration in an existential branch of computation. If the clause contains x_i (or $\overline{x_i}$), it is stored in the state x_i (or $\overline{x_i}$ respectively). Reading subsequent clauses leaves the previously saved clauses intact as the configuration for each subsequent clause makes exactly h shifts and therefore one full rotation through states $x_1, \dots, x_h$ and $\overline{x_1}, \dots, \overline{x_h}$ returning to the original configuration.

A high-level description of how the automaton works would be as follows: in the formula part of the word it nondeterministically guesses which clause in the

DNF is true and universally all the variables appearing in this clause must have agreeing values (i.e., 1 for x_k and 0 for $\overline{x_k}$), which are checked in the assignment part.

By the construction of the automaton, it starts in an existential state and on no input can it make more than 1 alternation. □

Now we consider a similar problem based on the conjunctive normal form (CNF).

Definition 1.10. *Let $f(x_1, \cdots, x_n)$ be a Boolean function and $(y_{11} \vee y_{12} \vee \cdots \vee y_{1l_1}) \wedge (y_{21} \vee y_{22} \vee \cdots \vee y_{2l_2}) \wedge \cdots \wedge (y_{m1} \vee y_{m2} \vee \cdots \vee y_{ml_m})$ its conjunctive normal form where m is the number of clauses, l_i $(1 \leq i \leq m)$ is the number of literals in the i-th clause, and each y_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq l_i)$ is either x_k or $\overline{x_k}$ for some $1 \leq k \leq n$. Assume that each variable appears in a clause no more than once and there are no more than 2^n clauses.*

Then we call $[z_{11}z_{12} \cdots z_{1n}][z_{21}z_{22} \cdots z_{2n}] \cdots [z_{m1}z_{m2} \cdots z_{mn}]$ a CNF-encoding of f where

$$z_{ik} = \begin{cases} x & \text{if } x_k \text{ appears in the } i\text{-th clause as } x_k \\ \bar{x} & \text{if } x_k \text{ appears in the } i\text{-th clause as } \overline{x_k} \\ - & \text{if } x_k \text{ does not appear in the } i\text{-th clause.} \end{cases}$$

Denote by DEC_{CNF} the function that maps a CNF-encoding to its function.

Definition 1.11. EVALUATE-CNF-FUNCTION$_h$ $=$
$(\{F.x_1 \ldots x_h \mid \exists f : DEC_{CNF}(F) = f \wedge \forall i\, x_i \in \{0, 1\} \wedge f(x_1, \ldots, x_h) = 1\},$
$\{F.x_1 \ldots x_h \mid \exists f : DEC_{CNF}(F) = f \wedge \forall i\, x_i \in \{0, 1\} \wedge f(x_1, \ldots, x_h) = 0\})$
EVALUATE-CNF-FUNCTION $=$ (EVALUATE-CNF-FUNCTION$_h$)$_{h \geq 1}$

Theorem 1.12. *There exists a family of 1AFAs $(\mathcal{A}'_h)_{h \geq 1}$ with $2h + 3$ states that solves the problem* EVALUATE-CNF-FUNCTION *starting in a universal state and using 1 alternation.*

Proof. See the automaton $\mathcal{A}'_h$ in Fig. 1.3. The automaton is very similar to the one for EVALUATE-DNF-FUNCTION$_h$ with a different behavior in the states $\forall$ and $\exists$. Therefore each clause of the CNF is now "saved" in a configuration in a universal branch of computation.

Again, a high-level description of the automaton is as follows: in the formula part of the word it universally for every clause in the CNF nondeterministically guesses which literal is true in this clause, and then verifies these guesses in the assignment part.

The automaton $\mathcal{A}'_h$ starts in a universal state and on no input can it make more than 1 alternation. □

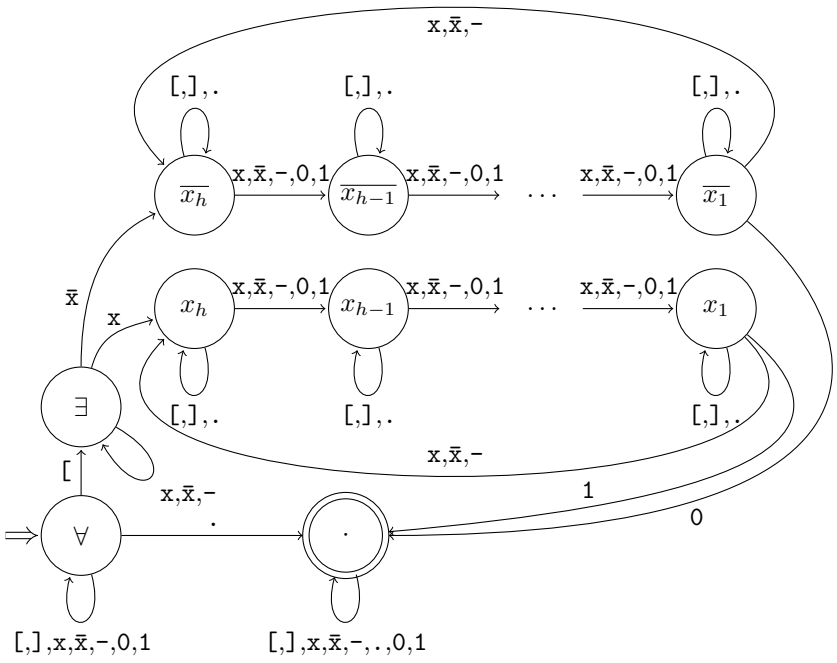

Figure 1.3: One-way alternating automaton $\mathcal{A}'_h$ with $2h + 3$ states solving EVALUATE-CNF-FUNCTION$_h$. Double arrow shows the starting state. State $\forall$ is universal and all other states are existential. Double circled state is accepting.

Theorem 1.13. *There exists no family of 2NFAs of fast 2P$_2$FAs that solves* EVALUATE-DNF-FUNCTION *or* EVALUATE-CNF-FUNCTION *with a polynomial number of states.*

To prove this theorem we will need two additional lemmas. The first one is about restricting the probability set of a 2PFA without significantly changing the isolation radius.

Lemma 1.14. *If there exists a k-state 2PFA $\mathcal{M}$ that solves $\mathcal{L} = (L, \tilde{L})$ with isolation radius $\delta(n)$ and with the expected runtime bounded by t for any $x \in L \cup \tilde{L}$, then for every $\varepsilon > 0$ there exists a k-state 2PFA $\mathcal{M}'$ that solves $\mathcal{L}$ with isolation radius $\delta(n) - \varepsilon$ such that $\mathcal{M}'$ uses transition probabilities from a set P with $|P| \leq \frac{3kt}{\varepsilon} + 1$.*

Proof. The automaton $\mathcal{M}'$ is constructed by substituting all transition probabilities in $\mathcal{M}$ by rounding down to the closest number in $P = \left\{0, \frac{1}{b}, \frac{2}{b}, \ldots, \frac{b-1}{b}, 1\right\}$ for some b (and distributing the remaining probability to an arbitrary state so that the sum of all probabilities for every transition is 1). In each step the probability of the automaton taking a different transition because of this rounding is $\leq \frac{3k}{b}$ (at most $\frac{1}{b}$ for each state and head movement direction). Therefore the total probability to make a different transition on any word x is $\leq \frac{3kt}{b}$. By choosing $b = \frac{3kt}{\varepsilon}$ the probability of the automaton $\mathcal{M}'$ making such error is $\leq \varepsilon$ therefore the isolation radius for $\mathcal{M}'$ is at least $\delta(n) - \varepsilon$. □

The second lemma essentially shows that adding some fixed prefix and suffix to every word does not help 2P$_2$FAs and 2NFAs to recognize the language.

Lemma 1.15. *Assume problems $\mathcal{L} = (L, \tilde{L})$ and $\mathcal{L}' = (L', \tilde{L}')$ over Σ are such that there exist $u, v \in \Sigma^*$ such that for all $x \in \Sigma^*$:*

$$x \in L' \Rightarrow uxv \in L,$$
$$x \in \tilde{L}' \Rightarrow uxv \in \tilde{L}.$$

If there exists a k-state 2NFA or 2P$_2$FA $\mathcal{M}$ solving $\mathcal{L}$, then there also exists a $(k+1)$-state NFA or 2P$_2$FA $\mathcal{M}'$, respectively, solving $\mathcal{L}'$. Moreover, for any x the acceptance probability and expected running time of $\mathcal{M}'$ on x is the acceptance probability and at most the expected running time of $\mathcal{M}$ on uxv, respectively.

Proof. We can transform a 2NFA $\mathcal{M}$ solving $\mathcal{L}$ into a 2NFA $\mathcal{M}'$ solving $\mathcal{L}'$ by altering its transition function δ to δ' so that the automaton treats the string $\vdash u$ as one symbol $\vdash$ and similarly the string $v\dashv$ as $\dashv$. That is, for every state q calculate the set of states $Q_q^{\vdash}$ in which the automaton can leave $\vdash u$, given that it entered $\vdash u$ from the right side in state q, and set $\delta'(q, \vdash) = Q_q^{\vdash} \times \{R\}$. Also for every state q calculate the set of states $Q_q^{\dashv}$ in which the automaton can leave $v\dashv$ to the left side and the set of accepting states $F_q^{\dashv}$ in which the automaton falls off to the right side of $\dashv$, given that it entered $v\dashv$ from the left side in state q, and set $\delta'(q, \dashv) = \left(Q_q^{\dashv} \times \{L\}\right) \cup \left(F_q^{\dashv} \times \{R\}\right)$. A new starting state is added to $\mathcal{M}'$ with transitions to all the states in which $\mathcal{M}$ can leave $\vdash u$ when started in the starting state on the first symbol of u. Therefore $\mathcal{M}'$ on $\vdash x \dashv$ will work exactly as $\mathcal{M}$ on $\vdash uxv \dashv$ (considering strings $\vdash u$ and $v\dashv$ as symbols $\vdash$ and $\dashv$).

The same idea is applicable to make from the 2P$_2$FA $\mathcal{M}$ a new 2P$_2$FA $\mathcal{M}'$: now for every two states q, q' we set $\delta'(q, \vdash, q', R)$ to the probability of $\mathcal{M}$ leaving $\vdash u$ in the state q', given that $\mathcal{M}$ entered $\vdash u$ from the right side in the state q. If there is a positive probability p to enter an infinite loop when entering $\vdash u$ from the right side in the state q, then $\delta'(q, \vdash, q_{stop}, N)$ is set to p where q_{stop} is a new special state for which $\delta'(q_{stop}, \vdash, q_{stop}, N) = 1$.

The new starting distribution of $\mathcal{M}'$ is set to the probability distribution of possible leaving states, given that the automaton $\mathcal{M}$ was started in its starting distribution on the first symbol of u.

For every two states q, q' $\delta'(q, \dashv, q', L)$ is set to the probability of $\mathcal{M}$ leaving $v\dashv$ to the left side in the state q', given that $\mathcal{M}$ entered $v\dashv$ from the left side in the state q. $\delta'(q, \dashv, q_{stop}, N)$ is set to the probability of $\mathcal{M}$ entering an infinite loop after entering $v\dashv$ from the left side in the state q. $\delta'(q, \dashv, q', R)$ is set to the probability of $\mathcal{M}$ falling off the right endmarker in q' after entering $v\dashv$ from the left side in the state q.

The probability for the automaton $\mathcal{M}'$ to accept $\vdash x \dashv$ is exactly the same as for the automaton $\mathcal{M}$ to accept $\vdash uxv \dashv$ and the expected runtime has not increased. □

Proof of Theorem 1.13. We will prove that, if the general evaluation problem could be solved by a small 2NFA (resp., small and fast 2P$_2$FA), then every explicit evaluation problem would be solved by a small 2NFA (resp., a small 2P$_2$FA with only exponentially fine distributions), contradicting the fact that the number of such problems is exponentially larger than the number of such automata.

For convenience we define some additional families of problems.

Definition 1.16. *For any h-variable Boolean function f and for any DNF-encoding F of f, we define the problem* EVALUATE-EXPLICIT-DNF-FUNCTION$_F$ $=$
$(\{F.x_1 \dots x_h \mid \forall i\, x_i \in \{0,1\} \wedge f(x_1,\dots,x_h) = 1\},$
$\{F.x_1 \dots x_h \mid \forall i\, x_i \in \{0,1\} \wedge f(x_1,\dots,x_h) = 0\})$

We define the problem EVALUATE-EXPLICIT-CNF-FUNCTION$_F$ *analogously for any h-variable Boolean function f and for any CNF-encoding F of f.*

It is easy to see that EVALUATE-EXPLICIT-DNF-FUNCTION$_F$ and EVALUATE-EXPLICIT-CNF-FUNCTION$_F$ are subproblems of EVALUATE-DNF-FUNCTION$_h$ and EVALUATE-CNF-FUNCTION$_h$, respectively, where h is the number of arguments of the function encoded by F and therefore any automaton solving EVALUATE-DNF-FUNCTION$_h$ or EVALUATE-CNF-FUNCTION$_h$ also solves EVALUATE-EXPLICIT-DNF-FUNCTION$_F$ or EVALUATE-EXPLICIT-CNF-FUNCTION$_F$, respectively.

Next, for every Boolean function f we define a similar problem, with the difference that the input word does not contain the encoding of the Boolean function.

Definition 1.17. EVALUATE-FUNCTION$_f$ $=$
$(\{x_1 \dots x_h \mid \forall i\, x_i \in \{0,1\} \wedge f(x_1,\dots,x_h) = 1\},$
$\{x_1 \dots x_h \mid \forall i\, x_i \in \{0,1\} \wedge f(x_1,\dots,x_h) = 0\})$

The difference between EVALUATE-EXPLICIT-DNF-FUNCTION$_F$, EVALUATE-EXPLICIT-CNF-FUNCTION$_F$ and EVALUATE-FUNCTION$_f$ is that in the former two every word is in the form $\vdash \mathtt{F}.x \dashv$ with some fixed F, but in the latter it is simply $\vdash x \dashv$. In both cases the automaton must accept the words for which $f(x) = 1$ and not accept the words for which $f(x) = 0$ where f is the function encoded by F.

From Lemma 1.15 it follows that if there exists a k-state 2NFA or 2P$_2$FA $\mathcal{M}$ that solves EVALUATE-EXPLICIT-DNF-FUNCTION$_F$ (resp., EVALUATE-EXPLICIT-CNF-FUNCTION$_F$), then there exists a $(k+1)$-state 2NFA or 2P$_2$FA $\mathcal{M}'$, respectively, that solves EVALUATE-FUNCTION$_f$ where $f = DEC_{DNF}(F)$ (resp., $f = DEC_{CNF}(F)$).

However, a remark about the running time of the 2P$_2$FA is necessary. Although the running time of the 2P$_2$FA has not increased, the input words in EVALUATE-FUNCTION$_f$ have become much shorter than in EVALUATE-EXPLICIT-DNF-FUNCTION$_F$ and EVALUATE-EXPLICIT-CNF-FUNCTION$_F$.

The length of an encoding and the length of the words in EVALUATE-EXPLICIT-DNF-FUNCTION$_F$ or EVALUATE-EXPLICIT-CNF-FUNCTION$_F$ are bounded by $O(h2^h)$, therefore if the expected running time of $\mathcal{M}$ is polynomial in the length of the input word, then the expected running time of $\mathcal{M}'$ is bounded by $2^{O(h)}$.

A similar remark is needed about the isolation radius of the 2P$_2$FA. If the isolation radius of $\mathcal{M}$ was bounded by $\frac{1}{r(h,n)}$ for some polynomial r then for $\mathcal{M}'$ it is bounded by $\frac{1}{r(h,O(h2^h))}$ which is $\frac{1}{2^{O(h)}}$.

So far we have proved that, if there exists a k-state 2NFA or fast 2P$_2$FA solving EVALUATE-DNF-FUNCTION$_h$ or EVALUATE-CNF-FUNCTION$_h$, then for each Boolean function f on h variables there exists a k-state 2NFA or $(k+1)$-state 2P$_2$FA with the expected running time bounded by $2^{O(h)}$ and isolation radius bounded by $\frac{1}{2^{O(h)}}$, respectively, solving EVALUATE-FUNCTION$_f$.

Using Lemma 1.14 we can transform the resulting 2P$_2$FA so that its transition function uses only probabilities from an appropriately sized set P with $|P| = 2^{poly(h)}$ where by $poly(h)$ we denote $h^{O(1)}$.

Note that there are 2^{2^h} different Boolean functions on h variables and each automaton can solve EVALUATE-FUNCTION$_f$ for at most one Boolean function f. Next, we proceed to count the number of automata with no more than $poly(h)$ states.

There are at most $2^{3\cdot(|\Sigma|+2)\cdot k^2} \cdot k \cdot 2^k$ different 2NFAs with no more than k states: for every two states, input alphabet symbol (or endmarker) and head movement direction there is a possibility to either have or not have such transition and k possibilities for the starting state and 2^k possibilities for the different sets of accepting states. For any k which is polynomial in h this is $2^{poly(h)}$ which for large enough h is less than the number of Boolean functions on h variables, therefore there exists a Boolean function which cannot be evaluated by a polynomial-size 2NFA.

If a 2PFA has all transition and starting distribution probabilities from a set P, then there are at most $|P|^{3\cdot(|\Sigma|+2)\cdot k^2} \cdot |P|^k \cdot 2^k$ such automata with no more than k states: for every two states, input alphabet symbol (or endmarker) and head movement direction the transition function assigns a probability from P of making this transition and $|P|^k$ possibilities for the starting distribution and 2^k possibilities for the set of accepting states. If $|P| = 2^{poly(h)}$, then for any $k = poly(h)$ the number of automata is also $2^{poly(h)}$ which for large enough h is less than the number of Boolean functions on h variables. Therefore there exists a Boolean function which cannot be evaluated by a polynomial-size 2P$_2$FA using probabilities from some P with $|P| = 2^{poly(h)}$.

So there exists no 2NFA or fast 2P$_2$FA with a polynomial number of states solving EVALUATE-DNF-FUNCTION or EVALUATE-CNF-FUNCTION. □

Therefore we have shown that the problems EVALUATE-DNF-FUNCTION and EVALUATE-CNF-FUNCTION are in $1\Sigma_2$ and $1\Pi_2$, respectively, but not in $2P_2$ or $2N$. Also it is easy to see that the complement of EVALUATE-FUNCTION$_f$ is EVALUATE-FUNCTION$_g$ for $g(x_1, \cdots, x_h) = \neg f(x_1, \cdots, x_h)$ therefore it follows that neither of these problems are in co-$2N$.

Corollary 1.18. *Neither $1\Sigma_2$ nor $1\Pi_2$ is contained in $2N \cup$ co-$2N \cup 2P_2$.*

1.4 Randomization over Alternation

In this section we show that if we relax the requirements for the 2PFAs, namely either the requirement for the 2PFA to be fast or to have at least polynomially decreasing isolation radius, then for any size bound there exists a family of problems not solvable by a family of 2AFAs of this size bound but solvable by a family of constant-sized 2PFAs.

Definition 1.19. *For all $h \geq 0$ we define* COUNT$_h$ $= (\{1^h\}, \{1^n \mid n \neq h\})$

Theorem 1.20. *For any $h \geq 0$ there exists a rational 4-state 1P$_2$FA which solves* COUNT$_h$.

Proof. Consider the following automaton $\mathcal{A}_h = (\{a,b,c,d\},\{1\},M_h,(1-\gamma,0,0,\gamma),\{b,d\})$ with the transition matrix

$$M_n = \begin{pmatrix} 1-\varepsilon_1 & \varepsilon_1 & 0 & 0 \\ 0 & 1-\varepsilon_2 & \varepsilon_2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where ε_1, ε_2 and γ depend on h (see Fig. 1.4).

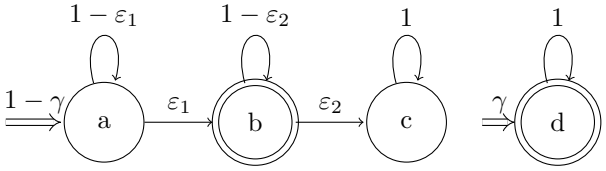

Figure 1.4: 4-state $1P_2FA$ solving COUNT_h. Double arrows show the starting probability distribution. Double circled states are accepting.

We will show that it is possible to choose $\varepsilon_1, \varepsilon_2$ so that the acceptance probability is the highest on the word 1^h. Let $\varepsilon_1 = \varepsilon_2$. Assuming that $\gamma = 0$, the probability of accepting 1^m is given by

$$(1-\varepsilon_1)^{m-1}\varepsilon_1 m$$

The derivative w.r.t. m of this quantity is

$$(1-\varepsilon_1)^{m-1}\varepsilon_1(1+m\ln(1-\varepsilon_1))$$

Solving it to be equal to 0 when $m = h$ gives $\varepsilon_1 = 1 - \mathrm{e}^{-1/h}$.

Therefore by setting $\varepsilon_1 = \varepsilon_2 = 1-\mathrm{e}^{-1/h}$ we get an automaton which has the maximal probability of acceptance on the word 1^h and the acceptance probability decreases as the length of the word increases or decreases. We can approximate this value by close enough rational number to obtain a rational automaton.

For $h \geq 2$ this maximal probability is less than $\frac{1}{2}$. We can shift the acceptance cutpoint to $\frac{1}{2}$ by making the automaton accept every word with some fixed probability, i.e., by setting an appropriate γ. For $h < 2$ it is straightforward to construct a deterministic automaton accepting COUNT_h.

Note that, although as h increases the difference between the probabilities of the automaton $\mathcal{A}_h$ to accept 1^h and 1^{h+1} (or 1^h and 1^{h-1}) decreases, for every fixed h there exist $\delta > 0$ such that the word 1^h is accepted with probability greater than $\frac{1}{2} + \delta$ and the probability to accept any other word is less than $\frac{1}{2} - \delta$, i.e. for every fixed automaton the cutpoint is isolated. □

Theorem 1.21. *For any constant $\delta < \frac{1}{2}$ there exists a constant k such that for any $n \geq 0$ there exists a coin-flipping k-state $2P_2FA$ which solves COUNT_h with isolation radius δ.*

Proof. Let $\text{EQUAL} = (\{0^n1^n\},\{0^n1^m \mid n \neq m\})$. In [18] Freivalds proved that for any $\delta < \frac{1}{2}$ there exists a coin-flipping $2P_2FA$ $\mathcal{P}$ which solves EQUAL with

an isolated cutpoint with isolation radius δ. For every $h > 0$ let EQUAL-TO-h $= (\{0^h1^h\}, \{0^h1^m \mid h \neq m\})$ (i.e., for every EQUAL-TO-h the h is fixed and the language contains exactly one word). It is easy to see that EQUAL-TO-h is a subproblem of EQUAL therefore $\mathcal{P}$ also solves EQUAL-TO-h.

Furthermore for each h the problems $\mathcal{L}$ = EQUAL-TO-h, $\mathcal{L}'$ = COUNT$_h$ and the word $u = 0^h$ satisfy the conditions of Lemma 1.15 and the automaton $\mathcal{P}$ solving EQUAL-TO-h can be transformed to an automaton $\mathcal{P}'_h$ solving COUNT$_h$ by adding one extra state. Therefore the number of states of $\mathcal{P}'_h$ depends only on the value of δ. □

Theorem 1.22. *For any k there exists m such that no 2AFA with k states can solve* COUNT$_m$.

Proof. Easily, as there is only a finite number of k-state 2AFAs and any 2AFA can solve COUNT$_m$ for at most one m. □

Given any size bound one can construct a family of problems $(\text{COUNT}_{f(h)})_{h\geq 1}$ which admits no family of 2AFAs of this size bound solving it by choosing a fast enough growing f. However by Theorems 1.20 and 1.21 for this family of problems there exists a family of 1P$_2$FAs solving it (with superpolynomially decreasing isolation radius) or a family of 2P$_2$FAs solving it (with superpolynomial expected runtime).

Corollary 1.23. *Neither 1P nor $2P_2X$ is contained in 2A or in any other class of families of problems solvable by a family of size-bounded 2AFAs.*

1.5 Conclusions

We have shown two examples of families of languages which are in $1\Sigma_2$ or $1\Pi_2$, respectively, but not in $2N$, co-$2N$ and $2P_2$.

In the proof of Theorem 1.13 we used the fact that there exist hard Boolean functions (such that require superpolynomial number of states to evaluate them) for 2PFAs that use only probabilities from a set P with $|P| = 2^{poly(h)}$. We conjecture that there exist hard Boolean functions for the class of all rational 2PFAs as well. This would imply that the examined problems EVALUATE-DNF-FUNCTION and EVALUATE-CNF-FUNCTION besides not being in the class $2P_2$, would also not be in $2P$ and $2P_2X$ leading to a result of incomparability between the classes $2A$ and $2P \cup 2P_2X$. However, this transition from a rather large set of probabilities P to all rational probabilities seems to make the task of proving the existence of a hard function for 2PFA unexpectedly hard as one cannot use the counting argument anymore (there are infinitely many rational 2PFAs with k states). Some new methods might be needed for proving this conjecture.

From our results we know that the class $2A$ is not contained in $2P_2$, but it is not known if $2P_2$ is contained in $2A$ or are they incomparable. For some larger classes, namely $2P$ and $2P_2X$, we showed that that they are not contained in $2A$, however we did not manage to do this for $2P_2$.

Chapter 2

Ultrametric Automata

In this chapter we define ultrametric finite automata which are based on p-adic numbers and explore various properties that arise from this definition. We also show some examples where ultrametric finite automata are more powerful or require a smaller number of states than equivalent deterministic finite automata.

The chapter is organized as follows. In Section 2.1 we give some general introduction and motivation for the computational model. In Section 2.2 we give some brief introduction to p-adic numbers. In Section 2.3 we define the ultrametric finite automata, prove some important properties about them, and justify the design choices in the definition which differ from the way how ultrametric automata were originally defined. In Section 2.4 we prove that one-way one-head ultrametric finite automaton is more powerful than k-head nondeterministic finite automaton for any k, and finally, in Section 2.5 we prove that for two-way multi-head ultrametric finite automata the power of the automaton grows as the number of heads increase. The last two sections represent results from a joint work with Rihards Krišlauks.

2.1 Introduction

Pascal and Fermat believed that every event of indeterminism can be described by a real number between 0 and 1 called *probability*. Quantum physics introduced a description in terms of complex numbers called *amplitude of probabilities* and later in terms of probabilistic combinations of amplitudes most conveniently described by *density matrices*.

String theory [59], chemistry [39] and molecular biology [14, 36] have introduced p-adic numbers to describe measures of indeterminism.

There were no difficulties to implement probabilistic automata and algorithms practically. Quantum computation has made a considerable theoretical progress but practical implementation has met considerable difficulties. However, prototypes of quantum computers exist, some quantum algorithms are implemented on these prototypes, quantum cryptography is already practically used. Some people are skeptical concerning practicality of the initial spectacular promises of quantum computation but nobody can deny the existence of quantum computation.

We consider a new type of indeterministic algorithms called *ultrametric* algo-

rithms. They are very similar to probabilistic algorithms but while probabilistic algorithms use real numbers r with $0 \leq r \leq 1$ as parameters, ultrametric algorithms use *p-adic* numbers. Slightly simplifying the description of the definitions one can say that ultrametric algorithms are the same probabilistic algorithms, only the interpretation of the probabilities is different.

Our choice of p-adic numbers instead of real numbers is not quite arbitrary. In 1916 Alexander Ostrowski proved that any non-trivial absolute value on the rational numbers Q is equivalent to either the usual real absolute value or a p-adic absolute value. This result shows that using p-adic numbers is not merely one of many possibilities to generalize the definition of deterministic algorithms but rather the only remaining possibility not yet explored.

Moreover, Helmut Hasse's local-global principle states that certain types of equations have a rational solution if and only if they have a solution in the real numbers and in the p-adic numbers for each prime p.

For every prime number p there exists a different notion of absolute value in the set of rational numbers. These absolute values are traditionally called *ultrametric*. Absolute values are needed to consider *distances* among objects. We are used to rational and irrational numbers as measures for distances, and there is a psychological difficulty to imagine that something else can be used instead of irrational numbers. However, there is an important feature that distinguishes p-adic numbers from real numbers. Real numbers (both rational and irrational) are linearly ordered. p-adic numbers cannot be linearly ordered. This is why *valuations* of p-adic numbers are considered.

The situation is similar in Quantum Computation. Quantum amplitudes are complex numbers which also cannot be linearly ordered. The counterpart of valuation for quantum algorithms is *measurement* translating a complex number $a + bi$ into a real number $a^2 + b^2$. Valuations of p-adic numbers are rational numbers.

Ultrametric finite automata and ultrametric Turing machines were first introduced by Freivalds [20]. This has been followed by several papers where various aspects of them are studied in depth. Balodis et al. [7] have studied the descriptional complexity of ultrametric automata. They showed that ultrametric automata can achieve an exponential advantage in terms of the number of states required when compared to equivalent deterministic automata. Krišlauks et al. [41] have studied the reversal complexity of ultrametric Turing machines.

Ultrametric machines are similar to probabilistic machines, the difference being that it is not necessary for amplitudes (which are the equivalent of probabilities in probabilistic automata) to be in the range between 0 and 1. Instead p-adic numbers are used. We should note that in [58] a similar generalization of probabilistic automata was introduced, where "probabilities" can be arbitrary real numbers, and the acceptance condition is whether the probability to be in an accepting state is greater than a given threshold, furthermore it was shown that this generalization is in fact equivalent to probabilistic automata. However, unlike in these generalized probabilistic machines, the definition of ultrametric machines uses the concept of a p-adic norm.

2.2 p-adic Numbers

In describing the notion of *p-adic numbers* we follow the introductory text by David A. Madore [46].

Let p be an arbitrary prime number. A p-adic digit is a natural number between 0 and $p-1$ (inclusive). A p-adic integer is by definition a sequence $(a_i)_{i\in\mathbb{N}}$ of p-adic digits. We write this conventionally as

$$\cdots a_i \cdots a_2 a_1 a_0$$

(that is, the a_i are written from right to left).

If n is a natural number, and

$$n = \overline{a_{k-1}a_{k-2}\cdots a_1 a_0}$$

is its p-adic representation (in other words $n = \sum_{i=0}^{k-1} a_i p^i$ with each a_i a p-adic digit) then we identify n with the p-adic integer (a_i) with $a_i = 0$ if $i \geq k$. This means that natural numbers coincide with p-adic integers with a finite number of nonzero digits. Also note that 0 is the p-adic integer all of whose digits are 0, and that 1 is the p-adic integer all of whose digits are 0 except the right-most one which is 1. If $\alpha = (a_i)$ and $\beta = (b_i)$ are two p-adic integers, we will now define their sum. To this purpose, we define by induction a sequence (c_i) of p-adic digits and a sequence (ϵ_i) of elements of $\{0, 1\}$ (the "carries") as follows:

- ϵ_0 is 0.
- c_i is $a_i + b_i + \epsilon_i$ or $a_i + b_i + \epsilon_i - p$ according as which of these two is a p-adic digit. In the former case, $\epsilon_{i+1} = 0$ and in the latter, $\epsilon_{i+1} = 1$.

Under those circumstances, we let $\alpha + \beta = (c_i)$ and we call $\alpha + \beta$ the sum of α and β. Note that the rules described above are exactly the rules used for adding natural numbers in base p. In particular, if α and β turn out to be natural numbers, then their sum as a p-adic integer is no different from their sum as a natural number. Addition is therefore associative and commutative. Similarly, subtraction and multiplication of p-adic integers is done exactly the same as with natural numbers in base p.

Division of p-adics, however, cannot always be performed. For example, $\frac{1}{p}$ has no meaning as a p-adic integer - that is, the equation $p\alpha = 1$ has no solution - since multiplying a p-adic integer by p always gives a p-adic integer ending in 0. There is nothing really surprising here: $\frac{1}{p}$ cannot be performed in the integers either. However, what is mildly surprising is that division by p is essentially the only division which cannot be performed in the p-adic integers. This is one of the reasons why the notion of p-adic integers is generalized and *p-adic numbers* are introduced. They are formal sequences of p-adic digits such that the sequence is infinite in the left-hand direction but finite in the right-hand direction. The notion of *p-adic dot* is introduced. The field of p-adic numbers is denoted by $\mathbb{Q}_p$.

For example, with $p = 7$ we show that the number $\alpha = \cdots 333334$ is the number corresponding to $\frac{1}{2}$ by adding it to itself:

$$\begin{array}{r} \cdots 333334 \\ \cdots 333334 \\ \hline\hline \cdots 000001 \\ \hline \end{array}$$

Thus, in the 7-adic numbers, $\frac{1}{2}$ is an integer. And so are $\frac{1}{3}$ $(\cdots 44445)$, $\frac{1}{4}$ $(\cdots 1515152)$, $\frac{1}{5}$ $(\cdots 541254125413)$, $\frac{1}{6}$ $(\cdots 55556)$, $\frac{1}{8}$ $(\cdots 0606061)$, and so on. But $\frac{1}{7}$, $\frac{1}{14}$ and so on, are not 7-adic integers. They are expressed as follows.

$$\cdots 0000.1$$
$$\cdots 0000.01$$

It is important to notice that p-adic numbers that are not p-adic integers and irrational real numbers are incompatible. It is known that no p-adic number corresponds to π or e and there is a continuum of p-adic numbers not corresponding to any real number. Also for some p there exists a number corresponding to $\sqrt{2}$ and for some p there does not exist such number. Moreover, if $p_1 \neq p_2$ then p_1-adic and p_2-adic numbers are incompatible.

However, p-adic numbers is not merely a generalization of rational numbers. They are very special because of the notion of *absolute value* of numbers.

A function $d : X \times X \to \mathbb{R}_{\geq 0}$ where X is a non-empty set is called a metric iff it satisfies the following conditions:

1. $d(x, y) = 0$ iff $x = y$,
2. $d(x, y) = d(y, x)$,
3. $d(x, y) \leq d(x, z) + d(z, y)$ for all $z \in X$.

If the third property can be replaced by its stronger variant – the strong triangle inequality $d(x, y) \leq max\{d(x, z), d(z, y)\}$ – the norm is called ultrametric. Otherwise, it is called Archimedean.

If the set X is equipped with the addition and multiplication operations (and forms a vector space), then a notion of norm can be introduced. The metric function is used to find the distances among the elements of a set. The distance of a given element to zero $d(x, 0)$ is called the norm or absolute value of the element and is denoted by $\|x\|$.

The norm of an element satisfies the following properties:

1. $\|x\| = 0$ if and only if $x = 0$,
2. $\|x * y\| = \|x\| * \|y\|$,
3. $\|x + y\| \leq \|x\| + \|y\|$ (the triangle inequality).

Definition 2.1. *Let p be any prime number. For any nonzero integer a, let the p-adic ordinal (or valuation) of a, denoted $ord_p a$, be the highest power of p which divides a, i.e., the greatest m such that $a \equiv 0 \pmod{p^m}$. For any rational number $x = a/b$, denote $ord_p x$ to be $ord_p a - ord_p b$. Additionally, $ord_p x = \infty$ if and only if $x = 0$.*

For example, the 7-adic valuation of 7 is 1. That of 14 is also 1, as are those of 21, 28, 35, 42 or 56. The 7-adic valuation of 49, on the other hand, is 2, as is that of 98. And the 7-adic valuation of 343 is 3. The 2-adic valuation of an integer is 0 iff it is odd, it is at least 1 iff it is even, at least 2 iff the integer is multiple by 4, and so on. The 7-adic valuation of $\frac{1}{7}$ is -1, and so are those of $\frac{3}{7}$, $\frac{1}{14}$, $\frac{5}{56}$. The 7-adic valuation of $\frac{1}{2}$ or $\frac{8}{3}$ is 0. The 7-adic valuation of $\frac{7}{3}$ or $\frac{14}{5}$ is 1. The 7-adic valuation of $\frac{48}{49}$ is -2.

Definition 2.2. *Let p be any prime number. For arbitrary rational number α, its* p-norm *is:*

$$\|\alpha\|_p = \begin{cases} \frac{1}{p^{ord_p \alpha}}, & \text{if } \alpha \neq 0; \\ 0, & \text{if } \alpha = 0. \end{cases}$$

Equivalently, one can define the p-norm of a rational number $\alpha = \pm 2^{\alpha_2} 3^{\alpha_3} 5^{\alpha_5} 7^{\alpha_7} \cdots$ where $\alpha_i \in \mathbb{Z}$ as $\|\alpha\|_p = \begin{cases} p^{-\alpha_p}, & \text{if } \alpha \neq 0 \\ 0, & \text{if } \alpha = 0. \end{cases}$

For example, $\|p\|_p = \frac{1}{p}, \|1\|_p = 1, \|2p\|_p = \frac{1}{p}$ (if p is odd), and $\|\frac{1}{p^2}\| = p^2$.

The valuation of a p-adic number (a_i) can equivalently be defined as the greatest i_0 such that $a_i = 0$ for all $i < i_0$. Sometimes it will be very convenient to use this definition. Hence a p-adic integer is exactly a p-adic number with non negative valuation. It is not hard to check that this definition coincides with the aforementioned one for integers, hence for rationals. Note that the p-adic absolute value of a p-adic number is a real number (and in fact rational).

p-adic numbers are discussed in more detail in [46]. The use of p-adics in other sciences can be seen in [39, 14].

2.3 p-adic Automata

The notion of p-adic numbers is widely used in mathematics but not so much in Computer Science. The aim of this section is to show that the notion of ultrametric automata is somehow natural.

In mathematics, a stochastic matrix is a matrix used to describe the transitions of a Markov chain. A *right stochastic matrix* is a square matrix each of whose rows consists of nonnegative real numbers, with each row summing to 1. A *stochastic vector* is a vector whose elements consist of nonnegative real numbers which sum to 1. The *finite probabilistic automaton* is defined as an extension of a non-deterministic finite automaton $(Q, \Sigma, \delta, q_0, F)$, with the initial state q_0 replaced by a stochastic vector giving the probability of the automaton being in a given initial state, and with stochastic matrices corresponding to each symbol in the input alphabet describing the state transition probabilities. It is important to note that if A is the stochastic matrix corresponding to the input symbol a and B is the stochastic matrix corresponding to the input symbol b, then the product AB describes the state transition probabilities when the automaton reads the input word ab. Additionally, the probabilistic automaton has a threshold λ being a real number between 0 and 1. If the probabilistic automaton has only one *final state* then the input word x is said to be accepted if after reading x the probability of the final state has a probability exceeding λ. If there are several final states, the word x is said to be accepted if the total of probabilities of the final states exceeds λ.

Ultrametric automata (introduced by Freivalds in [20]) are defined exactly in the same way as probabilistic automata, only the parameters called *probabilities of transition* from one state to another one are real numbers between 0 and 1 in probabilistic automata, and they are p-adic numbers called *amplitudes* in the ultrametric automata. At the beginning of the work, the states of the automaton get *initial amplitudes* being p-adic numbers. When reading the current symbol of the input word, the automaton changes the amplitudes of all the states according

to the transition matrix corresponding to this input symbol. After reading the input word, the *measurement* is performed, and the amplitudes of all the states are transformed into the p-valuations of these amplitudes. The valuations are real numbers and it is possible to compare whether or not the valuation exceed the threshold λ.

Paavo Turakainen considered various generalizations of finite probabilistic automata in 1969 and proved that there is no need to demand in cases of probabilistic branchings that total of probabilities for all possible continuations equal 1 [58]. He defined generalized probabilistic finite automata where the "probabilities" can be arbitrary real numbers, and that languages recognizable by these generalized probabilistic finite automata are the same as for ordinary probabilistic finite automata. Hence we also allow usage of all possible p-adic numbers in p-ultrametric machines.

Definition 2.3. *A one-way one-head p-ultrametric finite automaton ($1\,U_pFA(1)$ or simply $1\,U_pFA$ or U_pFA) is a sextuple $\langle Q, \Sigma, q_0, \delta, Q_A, Q_R\rangle$ where*

Q is a finite set – the set of states,

Σ is a finite set – input alphabet,

$q_0 : Q \to \mathbb{Q}_p$ is the initial amplitude distribution,

$\delta : \Sigma \times Q \times Q \to \mathbb{Q}_p$ is the transition function,

$Q_A, Q_R \subseteq Q$ are the sets of accepting and rejecting states, respectively.

The automaton works as follows. At every timestep each of its states has an associated p-adic number called its amplitude. The automaton starts with the initial amplitude distribution $s_\varepsilon = q_0$. Then it proceeds by processing input word's $w = w_1 \dots w_n$ symbols one by one. The amplitude distribution after processing the i-th symbol of the word w is denoted as $s_{w_1 \dots w_i} : Q \to \mathbb{Q}_p$, with the amplitude in the state q defined as

$$s_{w_1 \dots w_i}(q) = \sum_{q' \in Q} s_{w_1 \dots w_{i-1}}(q') \cdot \delta\left(w_i, q', q\right)$$

for every $q \in Q$. After the n-th symbol, the final amplitude distribution $s_{w_1 \dots w_n}$ is obtained. If the sum of the p-norms of final amplitudes over the accepting states is greater than the sum of final amplitudes over the rejecting states, i.e. if

$$\sum_{q \in Q_A} \|s_w(q)\|_p > \sum_{q \in Q_R} \|s_w(q)\|_p$$

then the word w is said to be accepted, otherwise – rejected.

It is sometimes useful to describe a state of the automaton as an $|Q|$-dimensional row vector s and the transition function as a set of $|\Sigma|$ matrices $\{A_\sigma\}$ ($\sigma \in \Sigma$) with $(A_\sigma)_{q',q} = \delta(\sigma, q', q)$. Then the state after reading the word $\sigma_1\sigma_2 \dots \sigma_n$ is simply $q_0 \cdot M_{\sigma_1} \cdot M_{\sigma_2} \cdot \ldots \cdot M_{\sigma_n}$.

Definition 2.4. *We say that a U_pFA* **recognizes** *language $L \subseteq \Sigma^*$ if any word $w \in \Sigma^*$ is accepted if and only if $w \in L$.*

It is easy to construct a U_pFA that recognizes a nonregular language.

Theorem 2.5. *For every prime p there exists a U_pFA recognizing $L = \{w \in \{0,1\}^* \mid w$ contains more 0s than 1s$\}$*

Proof. The automaton is shown in Fig. 2.1. For every $a \in \{0,1\}$ in the input word w the amplitude in state c_a is multiplied by p^{-1} therefore making the norm of this amplitude p times greater as $\|p^k\|_p = p^{-k}$. If after processing the whole word w the amplitude in state c_0 is greater than the amplitude in the state c_1 then the word is accepted, otherwise—rejected. □

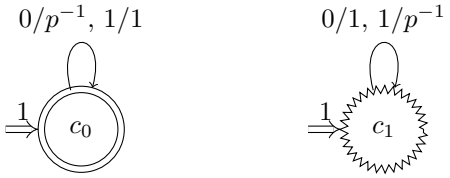

Figure 2.1: U_pFA recognizing $L = \{0^m 1^n \mid m < n\}$. The double circled state is accepting and the zigzagged state is rejecting. The double arrows show the starting amplitude distribution. Other arrows show the transitions. Label a/γ on arrow from q to r means that $\delta(a, q, r) = \gamma$.

To limit the focus on regular languages, we introduce the notion of a regulated U_pFA.

Definition 2.6. *If for U_pFA $M = \langle Q, \Sigma, s_0, \delta, Q_A, Q_R\rangle$ all transition amplitudes in δ are p-adic integers and there exist constants $d_1, d_2 \in \mathbb{Z}$ such that on any word $w \in \Sigma^*$ in any state $q \in Q$ either the amplitude $s_w(q)$ in state q after reading word w is equal to 0 or $p^{-d_2} \leq \|s_w(q)\|_p \leq p^{-d_1}$ then we call the automaton regulated (or more specifically – (d_1, d_2)-regulated).*

A regulated U_pFA can recognize only a regular language. Moreover, we can prove an upper bound on the number of states needed for a *DFA* recognizing the same language.

Theorem 2.7. *If a k-state (d_1, d_2)-regulated U_pFA $M = \langle Q, \Sigma, s_0, \delta, Q_A, Q_R\rangle$ recognizes a language L, then there exists a DFA with $2^{k(d_2-d_1+1)\log_2 p}$ states recognizing L.*

Proof. Recall from the definition of p-adic norm that for a p-adic number $\gamma = (\gamma_i)_{i \geq m}$ the norm $\|\gamma\|_p = p^{-j}$ where j is the index of the rightmost nonzero digit of γ.

Now let us consider an arbitrary amplitude $\gamma = s_w(q)$ in some state $q \in Q$ after having read some word $w \in \Sigma^*$. As the automation is (d_1, d_2)-regulated, all the digits of γ with index smaller than d_1 must be 0s, otherwise the norm $\|\gamma\|_p$ would be greater that p^{-d_1}. If $\gamma \neq 0$ then at least one of the digits from d_1 to d_2 must be non-zero, otherwise $\|\gamma\|_p$ would be less than p^{-d_2}. Therefore the digits with indexes from d_1 to d_2 unambiguously define the norm of the amplitude $\|\gamma\|_p$.

As all the transition amplitudes are p-adic integers, a higher-indexed digit of an amplitude does not influence a lower-indexed digit of any successor amplitude. As in the amplitude of a state all the digits with indexes less than d_1 are

0s and all the digits with indexes greater than d_2 do not influence the norm, a state can have only a finite number of essentially different amplitudes, namely $p^{d_2-d_1+1}$. Hence a configuration of an automaton can be described with one of $\left(p^{d_2-d_2+1}\right)^k = 2^{k(d_2-d_1+1)\log_2 p}$ states of a *DFA*. □

Theorem 2.8. *If a k-state U_pFA $M = \langle Q, \Sigma, s_0, \delta, Q_A, Q_R\rangle$ recognizes a language L with the property that on no word w the equality $\sum_{q\in Q_A} \|s_w(q)\|_p = \sum_{q\in Q_R} \|s_w(q)\|_p$ holds, then there exists a k-state U_pFA M' recognizing the complement language $\bar{L}$.*

Proof. Let $M' = \langle Q, \Sigma, s_0, \delta, Q_R, Q_A\rangle$. By swapping the sets Q_A and Q_R the inequality $\sum_{q\in Q_A} \|s_w(q)\|_p > \sum_{q\in Q_R} \|s_w(q)\|_p$ in the acceptance condition of the automaton M' holds iff it did not hold for automaton M, therefore making automaton M' accept the complement of the language of the automaton M. □

Theorem 2.9. *If a (d_1, d_2)-regulated k-state U_pFA recognizes a language L, then there is a $(d_1, d_2 + 1)$-regulated $k + 1$-state U_pFA that recognizes L with the property that there is no word w for which the equality $\sum_{q\in Q_A} \|s_w(q)\|_p = \sum_{q\in Q_R} \|s_w(q)\|_p$ holds.*

Proof. As the automaton is (d_1, d_2)-regulated, if the sums $\sum_{q\in Q_A} \|s_w(q)\|_p$ and $\sum_{q\in Q_R} \|s_w(q)\|_p$ are not equal, then they differ by at least p^{-d_2}. We can add to the automaton a new isolated rejecting state q_ε which at every step contains amplitude p^{d_2+1} therefore its norm is p^{-d_2-1} (see Fig. 2.2). Therefore it does not change the acceptance or rejectance of any word, but makes the sums $\sum_{q\in Q_A} \|s_w(q)\|_p$ and $\sum_{q\in Q_R} \|s_w(q)\|_p$ different on every word. □

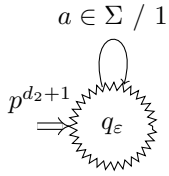

Figure 2.2: A rejecting state with a constant amplitude p^{d_2+1} in every step.

From Theorem 2.8 and Theorem 2.9 follows the next corollary:

Corollary 2.10. *If a (d_1, d_2)-regulated k-state U_pFA recognizes a language L, then there exists a (d_1, d_2+1)-regulated $(k+1)$-state U_pFA recognizing the complement language $\bar{L}$.*

We now give an easy example of a language that shows a gap between the state complexity of deterministic and ultrametric automata.

Let $w = (w_1, \ldots, w_m) \in \{0, 1, \ldots, k-1\}^m$. We consider two operations:

a) a cyclic shift: $f_a^{k,m}(w_1, w_2, \ldots, w_m) = (w_m, w_1, w_2, \ldots, w_{m-1})$.

b) increasing the first element: $f_b^{k,m}(w_1, w_2, \ldots, w_m) = ((w_1 + 1) \bmod k, w_2, \ldots, w_m)$.

Let $x \in \{a,b\}^n$. We define $f^{k,m}_{x_1x_2\ldots x_n}(w) = f^{k,m}_{x_n}\left(\cdots f^{k,m}_{x_2}\left(f^{k,m}_{x_1}(w)\right)\cdots\right)$. We consider the following language

$$L_{k,m} = \left\{x \in \{a,b\}^* \,\middle|\, f^{k,m}_x(0^m) = 0^m\right\}$$

Theorem 2.11. *Every deterministic finite automaton recognizing $L_{k,m}$ needs at least k^m states.*

Proof. For every words $x_1, x_2 \in \{a,b\}^*$ such that $f^{k,m}_{x_1}(0^m) \neq f^{k,m}_{x_2}(0^m)$ the automaton cannot remember words x_1, x_2 in the same state because there exists a suffix $x' \in \{a,b\}^*$ such that $f^{k,m}_{x_1x'}(0^m) = 0^m$, but $f^{k,m}_{x_2x'}(0^m) \neq 0^m$ and therefore $x_1x' \in L_{k,m}$, but $x_2x' \notin L_{k,m}$. Therefore the automaton needs at least k^m states. □

Theorem 2.12. *For every prime p there is a regulated U_pFA recognizing $L_{k,m}$ with $(k+1)\cdot m - 1$ states.*

Proof. The automaton has $k \cdot m$ states w^s_i where $i \in \{1,\ldots,m\}$, $s \in \{0,\ldots,k-1\}$. w^0_i are starting states with amplitude 1 and they are also the accepting states.

The automaton is constructed in such way that after reading x the automaton has amplitude 1 in the state w^s_i if $f^{k,m}_x(0^m)_i = s$ and amplitude 0 in all other w^t_i $(t \neq s)$.

Additionally the automaton has $m-1$ rejecting states $r_1,\ldots,r_{m-1}$ which at every step have amplitude 1. Therefore $\sum_{q\in Q_R} \|s_w(q)\|_p = m-1$.

If the sum of the norms of the amplitudes in the accepting states is m then the word is accepted, otherwise rejected. See Fig. 2.3. □

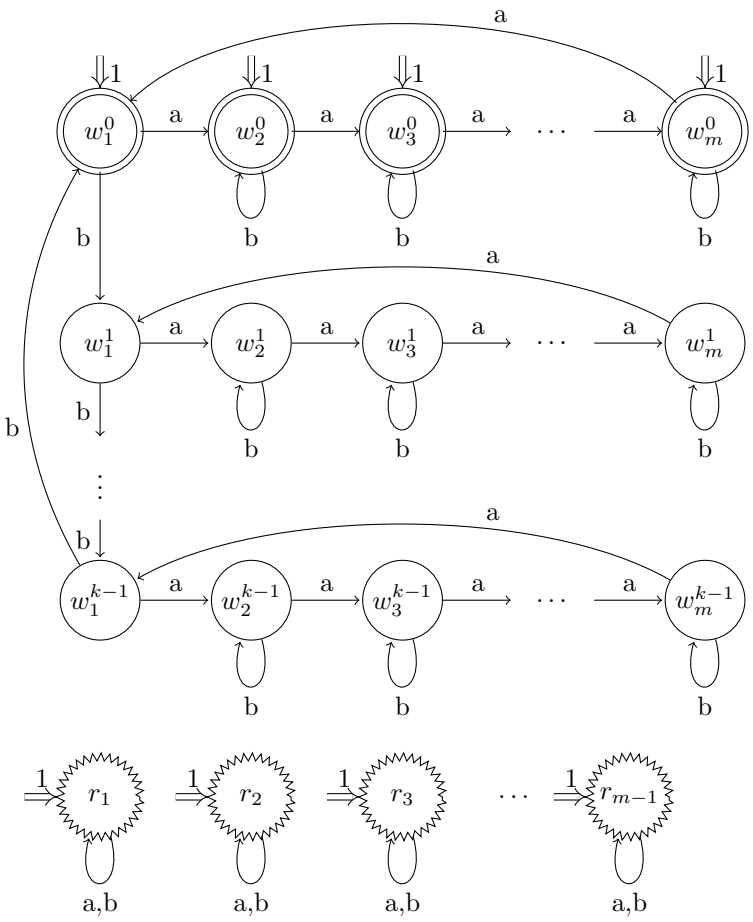

Figure 2.3: U_pFA with $(k+1) \cdot m - 1$ states recognizing $L_{k,m}$. Double circled states are accepting and zigzagged states are rejecting. Double arrows show the starting amplitude distribution. Other arrows show transitions on symbols a and b. All transitions have amplitude 1.

We can do even better if we take advantage of the ultrametric properties of the automaton.

Theorem 2.13. *For every prime $p > m$ there is a U_pFA recognizing $L_{p,m}$ with $m + 1$ states.*

Proof. The automaton has m rejecting states w_i where $i \in \{1, \ldots, m\}$ and an accepting starting state q_a which has always amplitude 1. The automaton is constructed in such way that after reading word x the state w_i contains an amplitude divisible by p iff $f_x^{k,m}(0^m)_i = 0$. Therefore, if $x \in L$ the sum of the norms of the amplitudes in the rejecting states $\sum_{q \in Q_R} \|s_w(q)\|_p$ is at most $m \cdot p^{-1} < 1$. Otherwise that sum is at least 1.

See Fig. 2.4. □

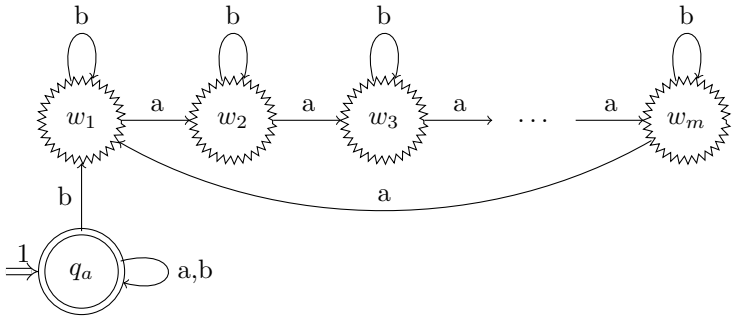

Figure 2.4: U_pFA with $m+1$ states recognizing $L_{k,m}$. The double circled state is accepting and zigzagged states are rejecting. The double arrow shows the starting amplitude distribution. Other arrows show transitions on symbols a and b. All transitions have amplitude 1.

Now we will compare our definition with the original definition that first appeared in [7]. Initially p-ultrametric finite automata were defined as follows (we will call them threshold automata).

Definition 2.14. *A finite p-ultrametric threshold automaton (U_pFTA) is a sextuple $\langle Q, \Sigma, s_0, \delta, F, \Lambda\rangle$, where*

Q *is a finite set – the set of states,*

Σ *is a finite set, ($\$ \notin \Sigma$) – the input alphabet,*

$q_0 : Q \to \mathbb{Q}_p$ *is the initial amplitude distribution,*

$\delta : (\Sigma \cup \{\$\}) \times Q \times Q \to \mathbb{Q}_p$ *is the transition function,*

$F \subseteq Q$ *is the set of final states,*

$\Lambda = (\lambda, \diamond)$ *is the acceptance condition where $\lambda \in \mathbb{R}$ is the acceptance threshold and $\diamond \in \{\leq, \geq\}$.*

The automaton works in the same way as U_pFA with the difference that after the last symbol of the input word w, in the same way as the input symbols, the end marker $ is processed obtaining the final amplitude distribution $s_{w\$}$. If the sum of the p-norms of final amplitudes over final states is at least (or at most) the threshold, i.e., if $\sum_{q\in F} \|s_{w\$}(q)\|_p \diamond \lambda$, then the word w is said to be accepted, otherwise – rejected.

A state of U_pFTA is called regulated if there exist constants ρ, c such that for every input word the p-norm of amplitude γ of this state is either 0 or bounded by $0 < \rho - c < \|\gamma\|_p < \rho + c$. A U_pFTA is called regulated if all the p-adic numbers in its initial distribution q_0 and transition function δ are p-adic integers and all of its states are regulated.

This definition differs from the way we earlier defined U_pFA in Def. 2.3 in two aspects. First is the use of endmarker, and second is the acceptance condition.

As the next theorem shows, the use of endmarker is not needed, as we can construct an equivalent U_pFTA without endmarker by increasing the number of states at most twice:

Theorem 2.15. *For every U_pFTA $M = (Q, \Sigma, q_0, \delta, F, \Lambda)$ there exists a U_pFTA $M' = (Q', \Sigma, q'_0, \delta', F', \Lambda)$ with $|Q| + |F|$ states such that for every word w: $\sum_{q\in F} \|s_{w\$}(q)\|_p = \sum_{q\in F'} \|s'_w(q)\|_p$, where s and s' are the amplitude distributions of U_pFTAs M and M', respectively.*

Proof. The automaton M' is constructed by making a copy q' of each final state $q \in F$. We define the set of states to be $Q' = Q \cup \{q' \mid q \in F\}$. The set of final states of M' now contain only the copied final states: $F' = \{q' \mid q \in F\}$.

The transition function δ' coincides with δ when only states $q \in Q$ are involved, i.e. $\forall a \in \Sigma\ \forall q_1, q_2 \in Q\ \delta(a, q_1, q_2) = \delta'(a, q_1, q_2)$. However for each final state $q' \in F'$ the transition is defined as to simulate reading the current symbol and the endmarker, i.e., $\forall a \in \Sigma\ \forall q_1 \in Q\ \forall q \in F\ \delta'(a, q_1, q') = \sum_{q_2\in Q} \delta(a, q_1, q_2) \cdot \delta(\$, q_2, q)$. From the final states $q' \in F'$ there are no outgoing transitions: $\forall a \in \Sigma\ \forall q' \in F' \forall q_2 \in Q' \delta'(a, q', q_2) = 0$.

Similarly we modify the initial distribution: $\forall q \in Q\ q'_0(q) = q_0(q)$, $\forall q' \in F'\ q'_0(q') = \sum_{q_1\in Q} q_0(q_1) \cdot \delta(\$, q_1, q)$.

Therefore for the automaton M' the amplitudes in the states $q \in Q$ after some word $w \in \Sigma^*$ coincide with the respective amplitudes in the automaton M: $\forall w \in \Sigma^*\ \forall q \in Q\ s'_w(q) = s_w(q)$, but the amplitude in the states $q' \in F'$ coincide with the respective amplitudes in M that would happen if the endmarker \$ would be read: $\forall w \in \Sigma^*\ \forall q' \in F'\ s'_w(q') = s_{w\$}(q)$. □

In the next theorem we prove that U_pFAs are as expressive as U_pFTAs if the threshold can be approximated by a number which can be written in base-p with a finite number of digits.

Theorem 2.16. *If a language L is recognized (without endmarker) by a U_pFTA $M = (Q, \Sigma, q_0, \delta, F, (\lambda, \diamond))$ such that there exists $\lambda' = \sum_{i=a}^{b} l_i \cdot p^i$ such that $\forall w \in \Sigma^*\ \sum_{q\in F} \|s_w(q)\|_p \diamond \lambda \Leftrightarrow \sum_{q\in F} \|s_w(q)\|_p \tilde{\diamond} \lambda'$ (where $\leq$ is $<$ and $\tilde{\geq}$ is $>$) then there exists a U_pFA M' with $|Q| + \sum_{i=a}^{b} l_i$ states which recognizes L.*

Proof. Let $l = \sum_{i=a}^{b} l_i$. If $\diamond$ is $\geq$ then M is modified to a U_pFA M' by setting $Q_A = F$ and adding l rejecting states $r_1, \ldots, r_l$ so that on any word $w \in \Sigma^*$ $\sum_{i=1}^{l} \|s_w(r_i)\|_p = \lambda'$. Therefore a word w is accepted if $\sum_{q\in Q_A} \|s_w(q)\|_p > \lambda'$ which coincides with the assumption in the formulation of the theorem.

If $\diamond$ is $\leq$ then M is modified similarly by setting $Q_R = F$ and adding l accepting states $r_1, \ldots, r_l$ so that on any word $w \in \Sigma^*$ $\sum_{i=1}^{l} \|s_w(r_i)\|_p = \lambda'$. Now a word w is accepted if $\sum_{q\in Q_R} \|s_w(q)\|_p < \lambda'$ which again matches the formulation of the theorem.

The states $r_1, \ldots, r_l$ are constructed so that for each $a \leq i \leq b$ there are l_i states with a starting amplitude p^{-i}. The transitions transform state r_j to r_j with amplitude 1. This ensures that on every $w \in \Sigma^*$ $\sum_{i=1}^{l} \|s_w(r_i)\|_p = \lambda'$. □

As the next theorem shows, we can always apply the previous theorem to a regulated U_pFTA as the required conditions hold.

Theorem 2.17. *If a language L is recognized by a regulated U_pFTA $M = (Q, \Sigma, q_0, \delta, F, (\lambda, \diamond))$ then there exists $\lambda' = \sum_{i=a}^{b} l_i \cdot p^i$ such that $\forall w \in \Sigma^*\ \sum_{q\in F} \|s_w(q)\|_p \diamond \lambda \Leftrightarrow \sum_{q\in F} \|s_w(q)\|_p \tilde{\diamond} \lambda'$.*

Proof. As the automaton is regulated, for every state the norm of its amplitude can take only a finite number of different values. Therefore the sum of the norms of all final states can also take only a finite number of different values. Therefore there is a gap between λ and the closest number to λ which does not satisfy the acceptance condition. □

2.4 One-Way Multi-Head Automata

In this section we prove that for every k there is a language that can be recognized by a one-head ultrametric finite automaton and cannot be recognized by any k-head non-deterministic finite automaton.

We define one-way non-deterministic (or deterministic) finite automaton as a special case of a more general two-way k-head non-deterministic finite automaton. The two-way setting will be considered in the next section.

A two-way k-head non-deterministic finite automaton consists of an input tape containing the input word on which the heads of the automaton can move freely in both directions, not crossing the endmarkers. The tape is read-only. We use the standard definition for the two-way k-head non-deterministic finite automaton:

Definition 2.18 ([28])**.** *A two-way non-deterministic k-head finite automaton ($2NFA(k)$) is a sextuple $\langle Q, \Sigma, k, q_0, \delta, F\rangle$, where*

Q is a finite set – the set of states,

Σ is a finite set ($\triangleright, \triangleleft \notin \Sigma$) – the input alphabet ($\triangleright$ and $\triangleleft$ are the left and right endmarkers, respectively),

$k \geq 1$ is the number of heads,

$q_0 \in Q$ is the starting state,

$\delta : Q \times (\Sigma \cup \{\triangleright, \triangleleft\})^k \to 2^{Q \times \{-1,0,1\}^k}$ is the partial transition function. Whenever $(q', (d_1, \ldots, d_k)) \in \delta(q, (a_1, \ldots, a_k))$ is defined, then $d_i \in \{0, 1\}$ if $a_i = \triangleright$, and $d_i \in \{-1, 0\}$ if $a_i = \triangleleft$, for $1 \leq i \leq k$,

$F \subseteq Q$ is the set of accepting states.

At the beginning of work, a $2NFA(k)$ has all of its heads placed on the left endmarker. A configuration of a $2NFA(k)$ in some moment in time $t \geq 0$ is a 3-tuple $c_t = (w, q, p)$ where w is the input word, $q \in Q$ is the current state and $p = (p_1, \ldots, p_k) \in \{0, \ldots, |w| + 1\}^k$ gives the current head positions. The initial configuration for input w is set to $(w, q_0, (0, \ldots, 0))$. The automaton ends its work when the transition function δ is not defined for the current configuration. A transition from one configuration to the next is denoted by $\vdash$. A transition $(w, q, (p_1, \ldots, p_k)) \vdash (w, q', (p_1 + d_1, \ldots, p_k + d_k))$ is valid iff $(q', (d_1, \ldots, d_k)) \in \delta(q, (a_{p_1}, \ldots a_{p_k}))$ where $w = a_1 a_2 \ldots a_n$ is the input word and $a_0 = \triangleright$ and $a_{n+1} = \triangleleft$. The reflexive transitive closure of $\vdash$ is denoted by $\vdash^*$.

A $2NFA(k)$ accepts a word w iff there exists a sequence of configurations that results in automaton stopping in an accepting state if the input tape contains

$\triangleright w \triangleleft$. The language $L(M)$ accepted by M consists of those and only those words that are accepted by M. More precisely,

$$L(M) = \{w \in \Sigma^* | \, (w, q_0, (0, \ldots, 0)) \vdash^* (w, q, (p_1, \ldots, p_k)) \, , q \in F, \\ \text{and } M \text{ halts in } (w, q, (p_1, \ldots, p_k))\} \, .$$

If for any state and k-tuple of symbols the transition function δ is either undefined or singleton, then the automaton is said to be deterministic ($2DFA(k)$). If the heads of the automaton never move left, then the automaton is defined to be one-way. Nondeterministic and deterministic one-way k-head automata are denoted by $1NFA(k)$ and $1DFA(k)$, respectively.

Strict hierarchies of classes have been shown for both one-way multi-head deterministic and nondeterministic automata with regard to the head count of the automata [28, 62]. In 1978, Yao and Rivest [62] used the language

$$L'_k = \left\{ w_1 \$ w_2 \$ \ldots \$ w_{2k} \middle| w_i \in \{a, b\}^* \wedge w_i = w_{2k+1-i} \text{ for all } 1 \leq i \leq k \right\}$$

to prove the separation of the class of languages that can be recognized by a $1DFA(k)$ from the class that can be recognized by a $1DFA(k+1)$.

We will consider a similar language, L_k.

Theorem 2.19. *For every $k \geq 1 \in \mathbb{N}$, there exists a language L_k such that:*

(1) for every prime p there exists a $1U_pFA(1)$ that recognizes L_k,

(2) L_k cannot be recognized by any $1NFA(k)$.

Proof. Let $n = \binom{k}{2} + 1$. The sought language is

$$L_k = \{w_1 1 w_2 1 \ldots 1 w_{2n} | w_i \in \{0^m | m \geq 1\} \wedge w_i = w_{2n-i+1}\} \, .$$

We will now prove that L_k satisfies the points of our theorem.

(1) We show that for an arbitrary language L_k, a $1U_pFA(1)$ can be built for every prime number p. The automaton starts in n different starting states $q_{1,1,1}, q_{1,2,1}, \ldots, q_{1,n,1}$ with amplitude 1. Each of these states begins a computational path that is intended to accumulate amplitude in one of n different rejecting states $q_{2n,1,2}, q_{2n,2,2}, \ldots, q_{2n,n,2}$. Every branch contains two kinds of states—states of the 1st group $q_{i,j,1}$ are responsible for generating amplitudes, and states of the 2nd group $q_{i,j,2}$ are intended for amplitude accumulation, $i \in [1, 2n] \, , j \in [1, n]$.

If 0 is read from the input and the automaton is in one of the 1st group states $q_{i,j,1}$, where $i \leq n$, then the amplitude of the state remains the same and with amplitude 1 the automaton goes to a 2nd group state, $q_{i,j,2}$. By doing so, the state's accumulated amplitude is added to $q_{i,j,2}$. If 0 is read in a 2nd group state $q_{i,j,2}$, the state's amplitude remains the same. If 1 is read in a 1st group state $q_{i,j,1}$, where $i < n$, then the automaton with amplitude $j+1$ transitions to $q_{i+1,j,1}$, thereby transitioning there with amplitude $(j+1) \cdot |q_{i+1,j,1}|$ (by $|q_i|$, we denote the amplitude of the state q_i). In contrast, if 0 is read in the 1st group state $q_{i,j,1}$, where $i > n$, the amplitude of the state remains unchanged and the transition to $q_{i,j,2}$ is made with amplitude -1. If 1 is read in the 1st group state $q_{i,j,1}$, where $i \geq n$, the transition to $q_{i+1,j,1}$ is made with amplitude $-(j+1)$. If 1 is read in a 2nd group state, a transition is made from $q_{i,j,1}$ to

$q_{i+1,j,1}$, with amplitude 1. The exception is the last column of states, $q_{2n,j,1}$ and $q_{2n,j,2}$, which are responsible for reading in the last block of the word. In this case, the transition if 1 is read is not defined. A schematic representation of the described automaton is presented in Fig. 2.5.

As a result, if a word $0^{a_1}10^{a_2}10^{a_3}1\ldots10^{a_{2n}}$ was read, then each of the rejecting states $q_{2n,j,2}$ has accumulated an amplitude equal to

$$a_1+a_2\cdot(j+1)+a_3\cdot(j+1)^2+\cdots+a_n\cdot(j+1)^{n-1}-a_{n+1}\cdot(j+1)^{n-1}-a_{n+2}\cdot(j+1)^{n-2}-\cdots-a_{2n},$$

which is equal to 0 if the word belongs to the language; i.e. if

$$a_1 = a_{2n} \wedge a_2 = a_{2n-1} \wedge \ldots \wedge a_n = a_{n+1}.$$

It follows that a word is in L_k iff the following equations hold:

$$\begin{cases} a_1 + a_2 \cdot 2 + a_3 \cdot 2^2 + \cdots + a_n \cdot 2^{n-1} - a_{n+1} \cdot 2^{n-1} - a_{n+2} \cdot 2^{n-2} - \cdots - a_{2n} = 0 \\ a_1 + a_2 \cdot 3 + a_3 \cdot 3^2 + \cdots + a_n \cdot 3^{n-1} - a_{n+1} \cdot 3^{n-1} - a_{n+2} \cdot 3^{n-2} - \cdots - a_{2n} = 0 \\ a_1 + a_2 \cdot 4 + a_3 \cdot 4^2 + \cdots + a_n \cdot 4^{n-1} - a_{n+1} \cdot 4^{n-1} - a_{n+2} \cdot 4^{n-2} - \cdots - a_{2n} = 0 \\ \ldots \\ a_1 + a_2 \cdot (n+1) + a_3 \cdot (n+1)^2 + \cdots + a_n \cdot (n+1)^{n-1} - a_{n+1} \cdot (n+1)^{n-1} \\ \qquad -a_{n+2} \cdot (n+1)^{n-2} - \cdots - a_{2n} = 0 \end{cases}$$

rewriting

$$\begin{cases} (a_1 - a_{2n}) + 2 \cdot (a_2 - a_{2n-1}) + 2^2 \cdot (a_3 - a_{2n-2}) + \cdots + 2^{n-1} \cdot (a_n - a_{n+1}) = 0 \\ (a_1 - a_{2n}) + 3 \cdot (a_2 - a_{2n-1}) + 3^2 \cdot (a_3 - a_{2n-2}) + \cdots + 3^{n-1} \cdot (a_n - a_{n+1}) = 0 \\ (a_1 - a_{2n}) + 4 \cdot (a_2 - a_{2n-1}) + 4^2 \cdot (a_3 - a_{2n-2}) + \cdots + 4^{n-1} \cdot (a_n - a_{n+1}) = 0 \\ \ldots \\ (a_1 - a_{2n}) + (n+1) \cdot (a_2 - a_{2n-1}) + (n+1)^2 \cdot (a_3 - a_{2n-2}) + \cdots \\ \qquad +(n+1)^{n-1} \cdot (a_n - a_{n+1}) = 0 \end{cases}$$

We see that the coefficients of the system form a Vandermonde matrix. Therefore, its determinant is non-zero, and since the given system is homogeneous, only the trivial solution exists.

However, if the word does not belong to L_k, then no more than 4 lines can hold true. However, even in this case at least one line will exist that is not equal to 0. Otherwise the system would have a nontrivial solution. Therefore, a word belongs to L_k iff the sum of the final amplitude norms of the rejecting states is greater than 0.

There is an additional set of states $a_1, \ldots, a_{2n}$ which accumulate a small (in terms of its norm) amplitude in an accepting state a_{2n} therefore making the automaton accept the language L_k.

(2) Proven by Freivalds [17], a proof for a similar language can also be found in [62]. The idea of this proof relies on the fact that any two heads that have been used to compare a pair cannot be used to compare another pair. This implies that if the number of block pairs in a word n is greater than the number of pairs of heads $\binom{k}{2}$, then the language cannot be recognized with k heads. □

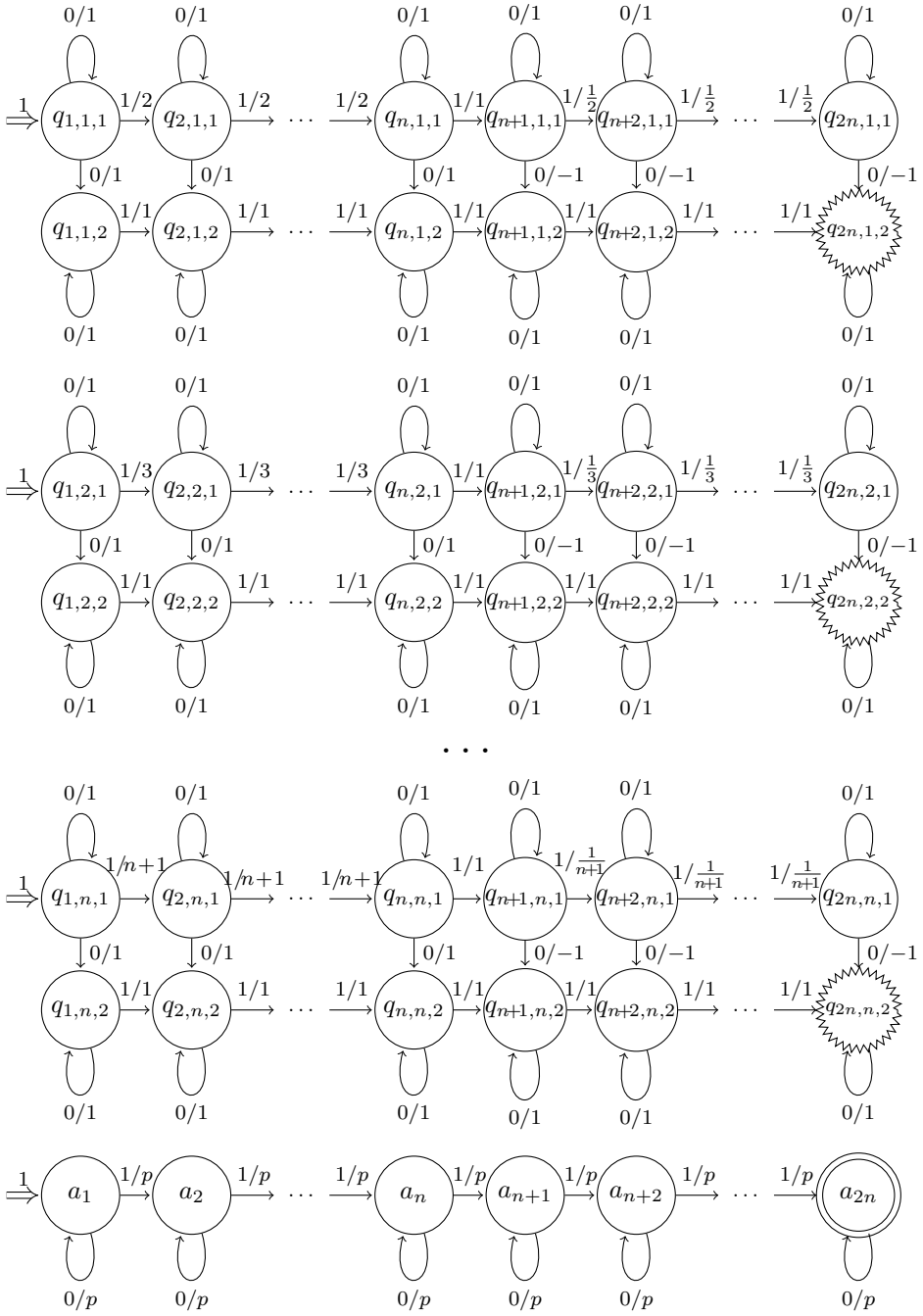

Figure 2.5: U_pFA recognizing $0^n10^m10^h1\cdots10^h10^m10^n$. The double-circled state is accepting and the zigzagged states are rejecting. Double arrows show the starting amplitude distribution. Other arrows show the transitions. Label a/γ on arrow from q to r means that $\delta(a,q,r)=\gamma$.

2.5 Two-Way Multi-Head Automata

Ultrametric two-way multi-head automata are defined by generalizing the definition of ultrametric one-head one-way automata in a natural way. The definition of multi-head automata due to Holzer et al. [28] is used as well.

Definition 2.20. *A finite k-head two-way p-ultrametric automaton ($2\,U_pFA(k)$) is a septuple $\langle Q, \Sigma, k, s_0, \delta, Q_A, Q_R\rangle$ where*

Q is a finite set of states,

Σ is a finite set ($\triangleright, \triangleleft \notin \Sigma$)—the input alphabet ($\triangleright$ and $\triangleleft$ are the left and right endmarkers, respectively),

$k \geq 1$ is the number of heads,

$q_0 : Q \to \mathbb{Q}_p$ is the initial distribution of amplitudes,

$\delta : Q \times (\Sigma \cup \{\triangleright, \triangleleft\})^k \times Q \times \{-1, 0, 1\}^k \to \mathbb{Q}_p$ is the partial transition function. Whenever $\delta\,(q, (a_1, \ldots, a_k), q', (d_1, \ldots, d_k))$ is defined and not equal to 0, then $d_i \in \{0, 1\}$ if $a_i = \triangleright$, and $d_i \in \{-1, 0\}$ if $a_i = \triangleleft$, for $1 \leq i \leq k$,

$Q_A, Q_R \subseteq Q$ are the sets of accepting and rejecting states, respectively.

$2\,U_pFA(k)$ works in a similar way as $1\,U_pFA$, with the exception that the automaton now has k heads that can move freely, as in a $2NFA(k)$. In contrast to $1\,U_pFA$, amplitudes are now given for a pair consisting of the positions of the heads and a state. The amplitude of the state $y \in Q$ with heads in positions $(p_1, \ldots, p_k) \in \{0, \ldots, |w| + 1\}^k$ on the word $w = a_1 \ldots a_n$ after the i-th operation is given by

$$s_i(y, (p_1, \ldots, p_k)) = \sum_{\substack{x \in Q, p'_1, \ldots, p'_k \in \{0,1,\ldots,n+1\}: \\ \delta\left(x, \left(a_{p'_1}, \ldots, a_{p'_k}\right), y, (p_1 - p'_1, \ldots, p_k - p'_k)\right) \text{ is defined}}} s_{i-1}\left(x, (p'_1, \ldots, p'_k)\right) \cdot \delta\left(x, \left(a_{p'_1}, \ldots, a_{p'_k}\right), y, (p_1 - p'_1, \ldots, p_k - p'_k)\right)$$

Similarly as before, the acceptance of a word is determined by comparing the final amplitude p-norm sum of the accepting and rejecting states. That is, the word is accepted iff

$$\sum_{x \in Q_A} \left\| \sum_{\substack{i \in \mathbb{N}, \\ (p_1, \ldots, p_k) \in F_i(x)}} s_i(x, (p_1, \ldots, p_k)) \right\|_p > \sum_{x \in Q_R} \left\| \sum_{\substack{i \in \mathbb{N}, \\ (p_1, \ldots, p_k) \in F_i(x)}} s_i(x, (p_1, \ldots, p_k)) \right\|_p$$

where $(p_1, \ldots, p_k) \in F_i(x)$ iff the automaton with some amplitude halts in the i-th step in the state x with the heads in positions $p_1, \ldots, p_k$.

The class of languages recognized by $2\,U_pFA(k)$ is denoted $\mathcal{L}\,(2\,U_pFA(k))$.

Ultrametric Turing machines are defined to be used as a device to assist in proving results regarding multi-head automata classes. We modify the definition for Turing machine by Hopcroft and Ullman [30].

Definition 2.21. *A p-ultrametric Turing machine with k work tapes ($U_pTM(k)$ or simply U_pTM) is an octuple $M = \langle Q, \Sigma, b, \Gamma, q_0, \delta, Q_A, Q_R\rangle$ where:*

Q *is a nonempty set of states,*

Σ *is a nonempty set—the input alphabet,*

$b \notin \Sigma$ *is the "empty" symbol,*

$\Gamma \supseteq \Sigma \cup \{b\}$ *is the working alphabet,*

$q_0 : Q \to \mathbb{Q}_p$ *is the initial amplitude distribution,*

$\delta : Q \times \Gamma^k \times Q \times (\Gamma \times \{-1, 0, 1\})^k \to \mathbb{Q}_p$ *is a partially-defined transition function where* -1 *denotes moving the head to the left,* 1 *denotes moving the head to the right,* 0 *denotes not moving the head, and* $\mathbb{Q}_p$ *is the amplitude of the transition,*

$Q_A, Q_R \subseteq Q$ *are the sets of accepting and rejecting states, respectively.*

The class of languages recognized by a $U_p TM$ is denoted $\mathcal{L}(U_p TM)$.

Similarly, as with ultrametric multi-head automata, the machine with a given amplitude is in one of its possible configurations. However, now the configuration consists of the state of the finite control, the position of the k heads, and the contents of all k tapes. The amplitude with which the machine is in one of its configurations in the i-th step is computed analogously as in the case with ultrametric multi-head automata. The criteria for acceptance are analogous to those of ultrametric multi-head automata.

Ultrametric multi-register automata are also used as an intermediate device for proofs.

Definition 2.22. *A finite p-ultrametric k register automaton (also referred to as a machine) (*$2U_pRA(k)$*) consists of a p-ultrametric finite control and k registers that can hold natural numbers. The automaton begins with the input number in the first register with the specified starting amplitude distribution in some of its states . A predicate* $\stackrel{?}{=} 0$ *and the operations* $+1$ *or* -1 *can be applied to a register. Each of the transitions of the finite control can have a predicate or an operation associated with it that is triggered when the transition is made. Acceptance criteria are analogous to those of ultrametric automata.*

In the following we will show how to simulate an ultrametric multi-head automaton by an ultrametric Turing machine. The techniques used here are similar to the techniques used by Macarie [45] for probabilistic automata.

Without loss of generality, we can assume that the simulated automaton has at most 2 transitions from each state and input symbol. We will show how to construct a p-ultrametric 2-tape Turing machine that, having received the description of a k-head p-ultrametric automaton as an input, will accept exactly the same words as the automaton. We will consider only automata recognizing unary languages, and we will show how to encode the description of the automaton as a word in the form $1^{2^n}, n \in \mathbb{N}$. Furthermore, we will require that the amplitudes of the simulated automaton are p-constructible sets.

Definition 2.23. *A set* S *of p-adic numbers is p-constructible if there exists a p-ultrametric Turing machine that, having received a description of a number* $x \in S$ *as an input, reaches a marked state* q_u *with amplitude* x*.*

A $2\,U_pFA(k)$ can be described with a binary sequence denoting, in turn: the number of states, the number of heads, the transitions between the states, and their respective amplitudes.

The simulation is performed as follows: The $2\,U_pFA(k)$ $\mathcal{A}$ with transitions from a p-constructible set is simulated by a $U_pTM(2)$ $\mathcal{T}$. $\mathcal{T}$ receives the description of $\mathcal{A}$ in unary alphabet as a word in the form $1^{2^n}, n \in \mathbb{N}$. $\mathcal{T}$ starts by reading the input word and deterministically (making transitions with amplitude 1) writes n in binary on the second tape, where n is the description of the automaton $\mathcal{A}$. Additionally, a space of size $O(k \cdot \log m)$ is reserved (where m is the length of the input word on which the automaton must be simulated) for k counters denoting the positions of the heads of the automaton $\mathcal{A}$. From this point forward, only the second tape will be used (we will refer to it as the work tape).

While processing the input word, the automaton $\mathcal{A}$ can be in different configurations in parallel with different amplitudes. If $\mathcal{A}$ is in a configuration with an amplitude a, $\mathcal{T}$ will simulate it by being in a configuration in which the content of the work tape corresponds to the respective configuration of $\mathcal{A}$ with the same amplitude a.

To show that $\mathcal{T}$ can simulate $\mathcal{A}$ in such a way, we must show that every transition of $\mathcal{A}$ can be realized by $\mathcal{T}$. If $\mathcal{A}$ has a transition from q_1 to q_2 with an amplitude a_1, and from q_1 to q_3 with an amplitude a_2, then $\mathcal{T}$ being in a configuration corresponding to q_1 can make transitions to configurations corresponding to q_2 and q_3 with amplitudes a_1 and a_2, respectively. This simulation is accomplished by $\mathcal{T}$ branching into two branches, and in both of them writing on a special place on the tape d_1 or d_2, respectively, with an amplitude 1, where d_1 and d_2 are the descriptions of the transitions and the amplitudes a_1 and a_2, respectively. This is done deterministically with amplitude 1. Next, a subroutine is called that transitions to a marked state q_u with an amplitude a_1 or a_2. (Because all transitions of $\mathcal{A}$ are p-constructible, there exists such a subroutine.) Subsequently, $\mathcal{T}$ changes the work tape so that it corresponds to the respective transition (again, this is done deterministically with amplitude 1). Because the only transition that is accomplished with an amplitude other than 1 is the transition to the state q_u, after this procedure $\mathcal{T}$ is in a configuration corresponding to q_2 with amplitude a_1, and in a configuration q_3 with amplitude a_2.

Monien [49] has proven that for finite deterministic and nondeterministic automata, the language class that can be recognized using k heads is a proper subclass to the class of languages that can be recognized if $k+1$ heads are allowed. Using similar methods, Macarie [45] has proven the same for finite probabilistic automata. We prove here that the same holds for finite ultrametric automata.

Definition 2.24. *By $\widehat{C}$, we denote the subset of a language class C containing only the words in the form $1^{2^n}, n \in \mathbb{N}$, more precisely $\widehat{C} = \left\{ L \in C | \forall x \in L\ \exists n \in \mathbb{N} : x = 1^{2^n} \right\}$*

Theorem 2.25. *For every natural number k and prime p:*

$$\mathcal{L}\left(2\widehat{U_pFA}(k)\right) \subsetneq \mathcal{L}\left(\widehat{U_pTM}\right).$$

Proof. We will construct a special p-ultrametric Turing machine with 2 tapes and *log*-space space complexity called $\mathcal{T}$. We will show that its recognized

language cannot be recognized by a p-ultrametric automata with k heads for any k.

Similarly as in the previous section, we will construct $\mathcal{T}$ so that it simulates a $2U_pFA(k)$ $\mathcal{A}$ given in its input. The input word on which $\mathcal{A}$ will be simulated will be the input of $\mathcal{T}$, i.e. the description of $\mathcal{A}$.

More precisely, $\mathcal{T}$ 1st tape contains $1^m, m \in \mathbb{N}$, $\mathcal{T}$ checks whether the word is in the form $1^{2^n}, n \in \mathbb{N}$ (if not, the word is rejected), and by taking up to $O(log(n))$ space, writes n's binary representation on the 2nd tape (we will refer to it as the work tape). It then checks whether n's binary representation is syntactically a valid $2U_pFA(k)$; if not, the word is rejected. Then, $\mathcal{T}$ designates a space on the work tape to be used for k counters that will be used to represent $\mathcal{A}$'s head positions. Since the counter values can be in $\{0, \ldots, 2^n + 1\}$, $O(k \cdot log(2^n)) \sim O(k \cdot n)$ space is required. All previous actions are performed deterministically. Next, $\mathcal{T}$ is run on the input string similarly as it is shown in the previous section. (The head of the first tape is not used anymore; instead, a check is performed to determine whether or not the counters corresponding to head positions are inside word boundaries.) Consequently, $\mathcal{T}$ is with respective amplitudes in all of the possible configurations of $\mathcal{A}$ after processing the word. Afterwards, $\mathcal{T}$ checks whether the contents of the work tape suggest that $\mathcal{A}$ is in an accepting state, and halts in a rejecting state if $\mathcal{A}$ would have accepted the word; $\mathcal{T}$ halts in an accepting state if $\mathcal{A}$ would have rejected the word. Therefore, $\mathcal{T}$ yields the opposite result to that of $\mathcal{A}$ with the same amplitudes.

Let us consider the language $L(\mathcal{T})$ recognized by $\mathcal{T}$. We can see that for every $2U_pFA(k)$ denoted by $\mathcal{J}$, there exists a word w such that it is either in $L(\mathcal{T})$ but not in $L(\mathcal{J})$, or it is in $L(\mathcal{J})$ but not in $L(\mathcal{T})$; namely, $\mathcal{J}$'s specification. □

Similarly, as in [45] and [49], in the following proofs we will use the function $f_k : \left\{1^{2^n} | n \in \mathbb{N}\right\} \to \left\{1^{2^n} | n \in \mathbb{N}\right\}$, where $f_k(1^{2^n}) = 1^{2^{k \cdot n}}$.

When f_k is applied to a language, we refer to the following function: $f_k(L) = \{f_k(x) | x \in L\}$.

Lemma 2.26. *For every language $L \in \mathcal{L}(\widehat{U_pTM})$ that is recognized by a 2-tape U_pTM in logarithmic space, there exists a natural number u such that: $f_u(L) \in \mathcal{L}(2\widehat{U_pFA}(3))$.*

Proof. We will show how a U_pTM denoted by $\mathcal{T}$ that recognizes L can be transformed into a U_pTM called $\mathcal{T}'$, which can then be replaced by a p-ultrametric 3 register machine. From this, it easily follows that there exists a $2U_pFA(3)$ that recognizes a "stretched" variant of L, where stretching is done by f_u.

We will construct $\mathcal{T}'$ so that it simulates $\mathcal{T}$. $\mathcal{T}'$ will hold the following information on its work tape:

- the binary information of the input word,
- $\mathcal{T}$ head position on the 1st tape (we will call it the input tape),
- $\mathcal{T}$ 2nd tape contents (we will call it the work tape) (requires $O(logn)$ space),
- $\mathcal{T}$ head position on the work tape.

The simulation of $\mathcal{T}$ on $\mathcal{T}'$'s work tape is less complicated than in the case of ultrametric automata, since $\mathcal{T}'$ can use the original finite control of $\mathcal{T}$, and the transitions are not required to be simulated on the tape. To simulate $\mathcal{T}$, $\mathcal{T}'$ uses only the work tape.

We can see that given $\mathcal{T}'$, a corresponding p-ultrametric 3 register machine can be constructed. If the input is of the form 1^{2^n}, then the register machine starts with n in the first register and with 0 in the remaining registers. The contents of the work tape of $\mathcal{T}'$ can be simulated by manipulating the registers of the first two stored sub-words, v and h^{rev}, and by using the third as an auxiliary register. To do this, we use operations "add 1" and "divide by 2", which can be carried out by using the auxiliary register.

Since a position of a multi-head automaton directly corresponds to a number in a register, and the simulated Turing machine has log-space space complexity, a 3 register machine can be replaced with a p-ultrametric 3 head automaton. However, since its heads cannot cross word boundaries and therefore cannot simulate arbitrarily large numbers, the input words must be sufficiently long. This is achieved by selecting a large enough u. □

Lemma 2.27. *For all languages* $L \in \mathcal{L}(\widehat{U_pTM})$ *and all* $u, v \geq 1, u, v \in \mathbb{N}$*:* $f_u(L) \in \mathcal{L}(2\widehat{U_pFA}(v)) \Rightarrow L \in \mathcal{L}(2\widehat{U_pFA}(u \cdot v))$.

Proof. Let the $2U_pFA(v)$ in the premise be $\mathcal{A}$ and the $2U_pFA(u \cdot v)$ in the conclusion—$\mathcal{A}'$.

Consider the operation of $\mathcal{A}$ on a word $f_u(l), l = 1^{2^n} \in L, n \in \mathbb{N}$. The position of each of v heads of $\mathcal{A}$ can be described with an integer $h_i \in [0, 2^{u \cdot n} - 1]$. h_i can be written in base 2^n with u digits. As the position of each head of $\mathcal{A}'$ can be described with a digit in base 2^n, each head of $\mathcal{A}$ can be simulated with u heads of $\mathcal{A}'$. As each movement of a head of $\mathcal{A}$ corresponds to movement of heads of $\mathcal{A}'$, the respective transitions can be accomplished with equal amplitudes, and the accepting amplitudes of the words remain the same.

Note that this simulation is performed analogously as for deterministic automata in [49] and for probabilistic automata in [45]. □

Lemma 2.28. *For every language* $L \in \mathcal{L}(\widehat{U_pTM})$ *and every* $u > v > 1, u, v \in \mathbb{N}$*:*

$$f_{u+1}(L) \in \mathcal{L}(2\widehat{U_pFA}(v)) \Rightarrow f_u(L) \in \mathcal{L}(2\widehat{U_pFA(v+1)}).$$

Proof. Let the $2U_pFA(v)$ in the premise be $\mathcal{A}$ and the $2U_pFA(v+1)$ in the conclusion—$\mathcal{A}'$.

Consider the operation of $\mathcal{A}$ on a word $f_{u+1}(l), l = 1^{2^n} \in L, n \in \mathbb{N}$.

The position of each of v heads of $\mathcal{A}$ on the input word can be described with an integer $h_i \in \left[0, 2^{(u+1)\cdot n} + 1\right]$.

We will simulate the position h_i in the automaton $\mathcal{A}'$ with a head g_i and an additional number $x_i \in [0, 2^n]$, so that $h_i = g_i + x_i \cdot 2^{u \cdot n}$. It is evident that the values in the necessary interval can be denoted this way:

$$2^{u \cdot n} + (2^n - 1) \cdot 2^{u \cdot n} = 2^{u \cdot n} \cdot (1 + (2^n - 1)) = 2^{u \cdot n + n} = 2^{(u+1) \cdot n}.$$

All v numbers x_i are coded with the $(v+1)$-th head of $\mathcal{A}'$, similarly to [49]. This can be accomplished if sufficient space exists on the tape of $\mathcal{A}'$, specifically if $(2^n)^v < 2^{u \cdot n}$, which holds as $v < u$.

Again, as each movement of a head of $\mathcal{A}$ corresponds to the movement of heads of $\mathcal{A}'$, the respective transitions can be accomplished with equal amplitudes, and the accepting amplitudes of the words remain the same. □

The result concerning the superiority of a $k+1$ head over k heads follows from the previous lemmas and Theorem 2.25.

Theorem 2.29. *For all $k \geq 2 \in \mathbb{N}$:*

$$\mathcal{L}\left(2\widehat{U_pFA}(k)\right) \subsetneq \mathcal{L}\left(2U_p\widehat{FA(k}+1)\right).$$

Proof. We prove from the contrary by showing that if there exists such $h \geq 2$ that $\mathcal{L}\left(2\widehat{U_pFA}(h)\right) = \mathcal{L}\left(2U_p\widehat{FA(h}+1)\right)$, it implies $\mathcal{L}\left(2U_pF\widehat{A(h\cdot}(h+1))\right) = \mathcal{L}\left(\widehat{U_pTM}\right)$, which contradicts 2.25.

Take $L \in \mathcal{L}\left(\widehat{U_pTM}\right)$ for some prime p. Lemma 2.26 implies that there exists $m \in \mathbb{N}$ such that $f_m(L) \in \mathcal{L}\left(2\widehat{U_pFA}(3)\right)$. Consequently, $f_m(L) \in \mathcal{L}\left(2\widehat{U_pFA}(h)\right)$. Lemma 2.28 implies that if $m > h+1$, then $f_{m-1}(L) \in \mathcal{L}\left(2U_p\widehat{FA(h}+1)\right) = \mathcal{L}\left(2\widehat{U_pFA}(h)\right)$. Reduce m by 1 and repeat until we get $f_m(L) \in \mathcal{L}\left(2\widehat{U_pFA}(h)\right)$ and $m = h+1$. Lemma 2.27 implies that if $f_m(L) \in \mathcal{L}\left(2\widehat{U_pFA}(h)\right)$, then $L \in \mathcal{L}\left(2U_p\widehat{FA(h\cdot m}\right) = \mathcal{L}\left(2U_pF\widehat{A(h\cdot}(h+1))\right)$. Contradiction with Theorem 2.25. □

Corollary 2.30. $\mathcal{L}\left(2U_pFA(k)\right) \subsetneq \mathcal{L}\left(2U_pFA(k+1)\right).$

Chapter 3

Counting with Automata

In this chapter, we investigate the descriptional complexity advantages for probabilistic and ultrametric automata compared with deterministic, nondeterministic and alternating automata. We limit our focus to unary languages containing exactly one word. We say that an automaton counts to n if it recognizes the one-word language $C_n = \{1^n\}$. We show that probabilistic and ultrametric automata for the counting problem can be very succinct, requiring only a constant number of states in many models.

In Section 3.1 we summarize the known results about counting with nonprobabilistic (deterministic, nondeterministic and alternating) automata. In Section 3.2 we show probabilistic automata for counting in different settings (one-way, two-way, bounded-error, with probability $1 - \varepsilon$). In Section 3.3 we show an optimal regulated ultrametric automaton for counting.

3.1 Finite Automata

In this section we show known results about deterministic, nondeterministic and alternating automata for counting. For self-containment purposes the simple proofs or the proof ideas are also provided.

Definition 3.1. $C_n = \{1^n\}$.

C_n is the language which corresponds to the problem COUNT_n defined in Chapter 1. It is trivial to construct a 1*DFA* for counting.

Theorem 3.2. *For each n there exists a 1DFA with $n+1$ states that recognizes C_n.*

Proof. Consider the automaton $\mathcal{A}_n = (\{0, 1, \ldots, n\}, \{1\}, \delta_n, 0, \{n\})$ (see Fig. 3.1) where $\delta_n(i, 1) = i + 1$ for all $0 \leq i < n$ and undefined for $i = n$. It is easy to see that it indeed accepts the language C_n. □

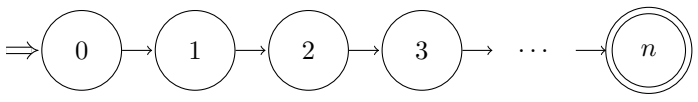

Figure 3.1: 1DFA with $n + 1$ states recognizing C_n. Double arrow shows the starting state. The double-circled state is final.

It is known that $n+1$ is also the necessary number of states even for 1NFA.

Theorem 3.3 ([43])**.** *$n+1$ states are necessary for recognizing C_n with a 1NFA.*

Proof. Assume that there are fewer than $n+1$ states for a 1NFA $\mathcal{A}$ recognizing C_n. An accepting run on $w = 1^n$ must visit at least one state more than once; therefore, it contains a cycle. It therefore follows that more than one word is accepted by $\mathcal{A}$. □

However, alternating automata can count to n with fewer states.

In [44], Leiss proved that any deterministic n-state automaton has an equivalent alternating automaton with $\lceil \log n \rceil$ states. Therefore, as noted in [43], Theorem 3.2 immediately implies the following theorem:

Theorem 3.4. *For each n, there exists a 1AFA with $\lceil \log n \rceil$ states that recognizes C_n.*

A 1AFA is called *one-switch* if it cannot move from an existential state to a universal state. Thus, a one-switch 1AFA alternates between the branching modes at most once.

In [10], Birget proved the following theorem:

Theorem 3.5 ([10])**.** *For each n there exists a one-switch 1AFA that recognizes C_n with $O(\log^2 n / \log\log n)$ states.*

Proof. Consider the automaton in Fig. 3.2. The concept behind this automaton is to count the remainder modulo of each of the first k primes $p_1, p_2, \ldots, p_k$ such that $p_1 \cdot p_2 \cdot \ldots \cdot p_k \geq n$. The first part of the automaton (states a^i_j) accepts the word 1^m iff $\forall 1 \leq i \leq k \;\; m \equiv n \pmod{p_i}$. By the Chinese Remainder Theorem, the unique number less than $p_1 \cdot p_2 \cdot \ldots \cdot p_k$ that satisfies this property is n itself.

The second part of the automaton (states b^i_j) does the opposite – it rejects the subwords whose length is m such that $\forall 1 \leq i \leq k \;\; m \equiv n \pmod{p_i}$. This part is being run on every proper suffix of the input word.

Therefore, the automaton accepts exactly those words $w = 1^m$ for which $\forall 1 \leq i \leq k \;\; m \equiv n \pmod{p_i}$ and no proper suffix of w has this property. Therefore, the only word that is accepted is 1^n.

It is known that the sum of the first k primes is $\sum_{i=1}^{k} p_i = \frac{1}{2}k^2 \ln k + o(k^2 \ln k)$ and the product of the first k primes is $\prod_{i=1}^{k} p_i = \mathrm{e}^{(1+o(1))k \ln k}$ (see for example [6]). From this basis, one can derive that the described automaton has $O(\log^2 n / \log\log n)$ states. □

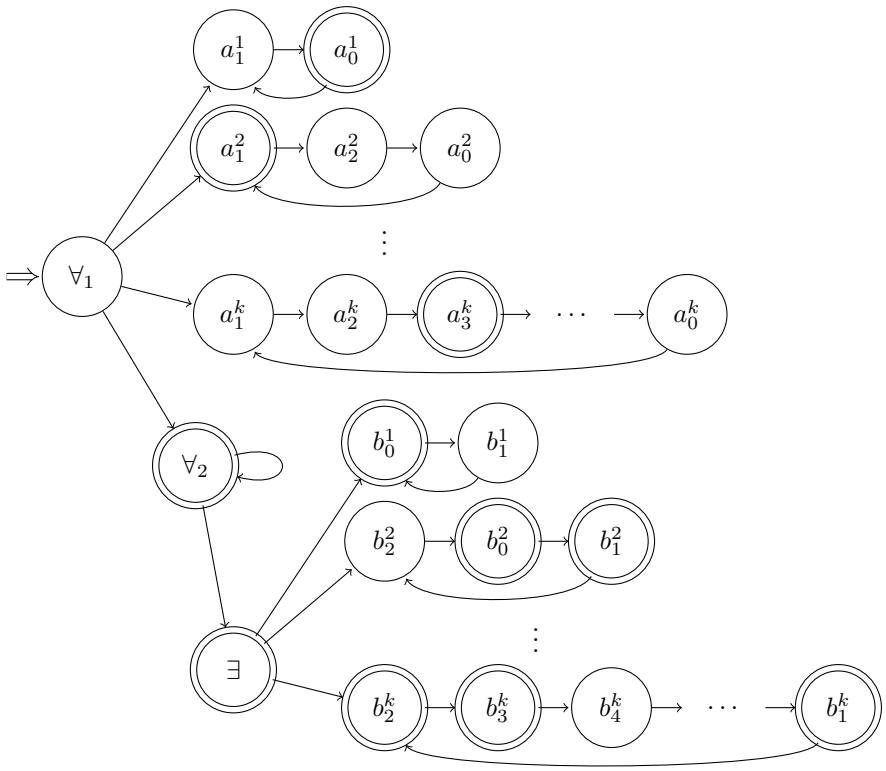

Figure 3.2: 1AFA recognizing C_n. The final states in the first set of cycles are exactly those a_j^i for which $n \equiv j \pmod{p_i}$. State b_j^i is final iff a_j^i is not. States $\forall_1$ and $\forall_2$ are universal; all others states are existential.

3.2 Probabilistic Automata

We show that 3 states are necessary and sufficient to count to n with a 1PFA with an isolated cutpoint.

Theorem 3.6. *For each n, there exists a 1PFA that recognizes C_n with 3 states with an isolated cutpoint.*

Proof. Consider the following automaton $\mathcal{A}_n = (\{1,2,3\}, \{1\}, M_n, (1,0,0), \{2\})$ with the transition matrix

$$M_n = \begin{pmatrix} 1-\varepsilon_1 & \varepsilon_1 & 0 \\ 0 & 1-\varepsilon_2 & \varepsilon_2 \\ 0 & 0 & 1 \end{pmatrix}$$

where ε_1 and ε_2 depend on n (see Fig. 3.3).

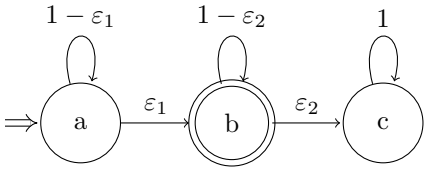

Figure 3.3: 1PFA with 3 states recognizing C_n. The double arrow shows the starting state (with probability 1). The double-circled state is final.

This is the same automaton as in Theorem 1.20 with the exception that the state which shifted the threshold to $\frac{1}{2}$ is dropped (we now allow an arbitrary cutpoint λ instead of $\frac{1}{2}$). As already shown in Theorem 1.20, by setting $\varepsilon_1 = \varepsilon_2 = 1 - \mathrm{e}^{-1/n}$, we get an automaton that has the maximal probability of acceptance on the word 1^n, and the acceptance probability decreases as the length of the word increases or decreases.

Again, note that although as n increases the difference between the probabilities of the automaton $\mathcal{A}_n$ to accept 1^n and 1^{n+1} (or 1^n and 1^{n-1}) decreases, for every n there exist λ and $\delta > 0$ such that the word 1^n is accepted with probability greater than $\lambda + \delta$, and the probability to accept any other word is less than $\lambda - \delta$, i.e. the cutpoint is isolated. □

An example plot of the acceptance probability of words 1^x with $\varepsilon_1 = \varepsilon_2 = 1 - \mathrm{e}^{-\frac{1}{200}}$ is shown in Fig. 3.4.

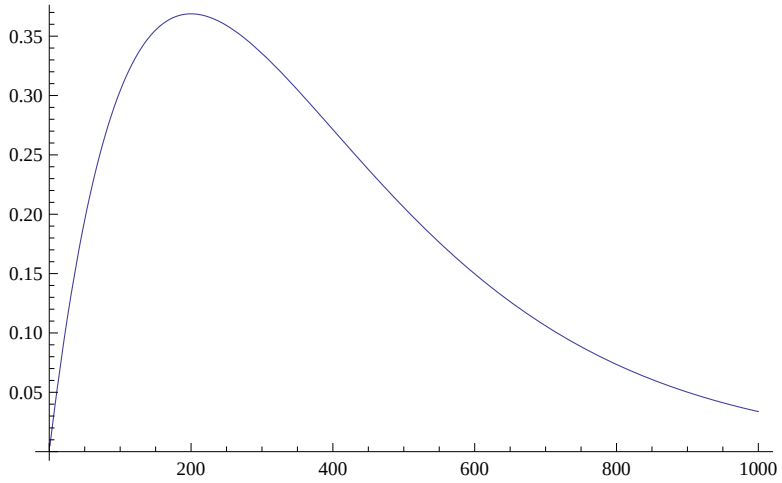

Figure 3.4: The probability of accepting 1^x with $\varepsilon_1 = \varepsilon_2 = 1 - \mathrm{e}^{-\frac{1}{200}}$.

Theorem 3.7. *If $n > 1$ then any 1PFA that recognizes C_n has at least 3 states.*

Proof. Assume there exists a 1PFA with 2 states recognizing C_n for $n > 1$. Exactly one of the states must be final; otherwise, either all or none of the words would be accepted. Without loss of generality, assume that the first state is the non-final state and the second state is the final state. Let

$$M_1 = \begin{pmatrix} 1-p & p \\ m & 1-m \end{pmatrix}$$

be the transition matrix of the automaton.

$$M_1^2 = \begin{pmatrix} 1-p(2-m-p) & p(2-m-p) \\ m(2-m-p) & 1-m(2-m-p) \end{pmatrix}$$

If the automaton is in a probability distribution $\begin{pmatrix} a \\ 1-a \end{pmatrix}$ then after reading two symbols (twice applying M_1), the probability distribution becomes

$$\begin{pmatrix} m(2-m-p)+a(m+p-1)^2 \\ 1-m(2-m-p)-a(m+p-1)^2 \end{pmatrix}$$

Notice that for every $a \in [0,1]$ the probability of being in a final state changes monotonically after every two symbols read; therefore, the automaton cannot recognize C_n for $n > 1$. □

Notice that although each individual automaton $\mathcal{A}_n$ from Theorem 3.6 has an isolated cutpoint, the isolation radius decreases as n increases.

In [19], Freivalds showed how to construct a series of 1PFAs for recognizing C_n with a constant isolation radius.

Theorem 3.8 ([19])**.** *For each n, there exists a 1PFA with $O(\log^2 n / \log\log n)$ states that recognizes C_n with probability $\frac{3}{5}$.*

Proof. The automaton is similar to the 1AFA from the proof of Thm. 3.5. However, now the rejection of long words is performed by a probabilistic clock rather than an alternating behavior. The automaton is defined as follows:

$$\mathcal{A}_n = (\{b, a_0^1, \ldots, a_{p_k-1}^k\}, 1, \delta, (0, \frac{1}{k}, 0, \ldots, 0, \frac{1}{k}, 0, \ldots, 0, \ldots, \frac{1}{k}, 0, \ldots, 0), \{a^1_{n \bmod p_1}, \ldots, a^k_{n \bmod p_k}\})$$

with p_i states $a_0^i, \ldots, a_{p_i-1}^i$ for each of the first k primes $p_1, \ldots, p_k$ (the value of k is to be determined later). The states $a_0^i, \ldots, a_{p_i-1}^i$ form a cycle, which counts the remainder modulo p_i. State a_j^i is final iff $n \equiv j \pmod{p_i}$. The starting probability distribution of the automaton is $\frac{1}{k}$ in each of the states $a_0^1, a_0^2, \ldots, a_0^k$ (see Fig. 3.5).

From every state there is a transition with probability $1-\varepsilon$ to the state b.

By the Chinese Remainder Theorem, if $p_1 < p_2 < \ldots < p_k$ are the first k primes and $p_1 \cdot p_2 \cdot \ldots \cdot p_l \geq n$, then $\forall n' < p_1 \cdot p_2 \cdot \ldots \cdot p_l$ if $n' \neq n$ at most $l-1$ of the following congruences are satisfied:

$$\begin{aligned} n' &\equiv n \pmod{p_1} \\ n' &\equiv n \pmod{p_2} \\ &\ldots \\ n' &\equiv n \pmod{p_k} \end{aligned}$$

Let l be the minimal number such that $p_1 \cdot p_2 \cdot \ldots \cdot p_l \geq 2n$.

Choose $k = 3l$.

Therefore, any word with length $n' < 2n \leq p_1 \cdot p_2 \cdot \ldots \cdot p_l$ will be accepted with probability at most $\frac{l-1}{k} < \frac{l}{3l} = \frac{1}{3}$. Words with length n will be accepted

with probability $1 - E(n)$ where $E(n)$ is the probability to be in state b after reading a word of length n. By the sum of geometric series, $E(n) = \varepsilon + (1-\varepsilon) \cdot \varepsilon + (1-\varepsilon)^2 \cdot \varepsilon + \cdots + (1-\varepsilon)^{n-1} \cdot \varepsilon = \sum_{i=0}^{n-1} \varepsilon \cdot (1-\varepsilon)^i = \varepsilon \frac{1-(1-\varepsilon)^n}{1-(1-\varepsilon)} = 1-(1-\varepsilon)^n$. Words of length $n' \geq 2n$ will be accepted with probability at most $1 - E(2n)$.

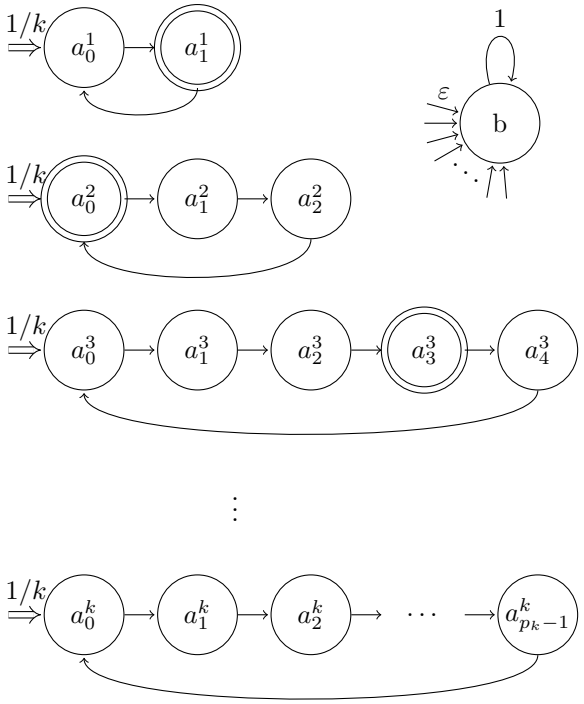

Figure 3.5: 1PFA recognizing C_n. The accepting states in each cycle are exactly those that correspond to n mod p_i For purposes of clarity, the transitions from each state to state b with probability ε are not drawn. All of the unlabeled transitions have probability $1 - \varepsilon$.

By choosing $\varepsilon = 1 - \sqrt[n]{\frac{3}{5}}$ we get $1 - E(n) = \frac{3}{5}$ and $1 - E(2n) = \frac{9}{25} < \frac{2}{5}$. Therefore, 1^n is accepted with probability $\frac{3}{5}$, and every other word is accepted with probability $< \frac{2}{5}$. □

As the next theorem shows, for the two-way probabilistic finite automata, even a constant number of states suffices to recognize C_n with a fixed probability.

Theorem 3.9. *There exists a constant c such that for every $\varepsilon > 0$ and for each n there exists a 2PFA that recognizes C_n with c states with probability $1 - \varepsilon$.*

Proof. In [18], Freivalds showed for every $\varepsilon > 0$ how to construct a 2PFA with c_ε states that recognizes the language EQUAL $= \{0^n 1^n \mid n \geq 1\}$ with probability $1 - \varepsilon$. Yakaryılmaz and Say in [61] showed an improved construction where the number of states for the 2PFA does not depend on ε.

Lemma 1.15 from Chapter 1 shows that if languages $\mathcal{L}, \mathcal{L}' \in S^*$ are such that there exists $u \in S^*$ such that $\forall x \in S^* (x \in \mathcal{L}' \Leftrightarrow ux \in \mathcal{L})$ then a 2PFA $\mathcal{A}$

recognizing $\mathcal{L}$ can be modified into a 2PFA $\mathcal{A}'$ recognizing $\mathcal{L}'$ with the number of states increasing by 1.

One can see that for C_n and EQUAL, there exists u (namely $u = 0^n$) such that $\forall x \in \{1\}^* (x \in C_n \Leftrightarrow ux \in \text{EQUAL})$. We can therefore use Lemma 1.15 to construct an automaton with 1 extra state and then restrict it to a smaller alphabet $\{1\}$ to obtain an automaton recognizing C_n. □

3.3 Ultrametric Automata

In this section we show an optimal U_pFA for counting. Surprisingly only 2 states are enough for a regulated U_pFA to be able to count to n.

Theorem 3.10. *For each n and each prime p there exists a regulated U_pFA that recognizes C_n with 2 states.*

Proof. Consider the automaton $\mathcal{A}_n = (\{a, b\}, \{1\}, ((p-1)p^n, 1), \{\delta_1\}, \{a\}, \{b\})$ (see Fig. 3.6) where

$$\delta_1 = \begin{pmatrix} 1 & 1 \\ 0 & p \end{pmatrix}$$

The accepting state a at every step has amplitude $(p-1)p^n$ which gives norm $\|(p-1)p^n\|_p = p^{-n}$. The amplitude of the rejecting state b is made of two components—one which originated from its starting amplitude distribution (1) and other which "flows" from the state a. In each step the current amplitude of b is multiplied by p which in its p-adic representation means moving every digit one position to left. Once the digit 1 from the starting distribution has moved to position n (this happens on the word 1^n) where the amplitudes from a are positioned they both "collide" making a large decrease in the norm of the state b. This happens exactly because $(p-1)+(p-1)p+(p-1)p^2+\cdots+(p-1)p^k+1 = p^{k+1}$. However on the next (and each subsequent) step the amplitudes flowing from a determine the norm of the amplitude in state b.

See for example how the amplitudes and norms for states a and b change on different words w with $p = 7$ and $n = 5$:

w	$s_w(a)$	$\|s_w(a)\|_p$	$s_w(b)$	$\|s_w(b)\|_p$
ε	...0600000	7^{-5}	...000000000000001	1
1	...0600000	7^{-5}	...000000000600010	7^{-1}
11	...0600000	7^{-5}	...000000006600100	7^{-2}
111	...0600000	7^{-5}	...000000066601000	7^{-3}
1111	...0600000	7^{-5}	...000000666610000	7^{-4}
11111	...0600000	7^{-5}	...000010000000000	7^{-10}
111111	...0600000	7^{-5}	...000100000600000	7^{-5}
1111111	...0600000	7^{-5}	...001000006600000	7^{-5}
11111111	...0600000	7^{-5}	...010000066600000	7^{-5}

Table 3.1: The norms and amplitudes of the states of automaton $\mathcal{A}_n$ on different words w.

Therefore the word $w = 1^n$ is the only one on which $\|s_w(a)\|_p > \|s_w(b)\|_p$.

The maximal norm of the state b is 1 (on the empty word ε), and the minimum norm is p^{-2n} (on the word 1^n) therefore the automaton is regulated. □

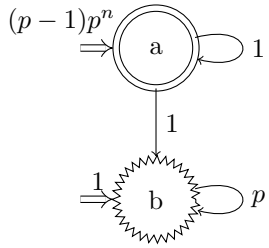

Figure 3.6: Regulated U_pFA with 2 states recognizing C_n. The double circled state is accepting and the zigzagged state is rejecting. Double arrows show the starting amplitude distribution. Edges show the transitions on symbol 1 with labels indicating the transition amplitude.

The next theorem shows that this result is optimal and 2 states are needed even for non-regulated U_pFA.

Theorem 3.11. *If $n > 0$ then any U_pFA that recognizes C_n has at least 2 states.*

Proof. Assume there exists a U_pFA with 1 state a that recognizes C_n. Let $\alpha = \delta(1, a, a)$ be the coefficient that is multiplied to the amplitude of state a for every symbol of the input word. The final amplitude after processing word 1^m in the state a is $q_0(a) \cdot \alpha^m$.

Assume α is a p-adic integer. If its rightmost $p-$adic digit is 0, then multiplying any p-adic number (except 0) by α decreases the norm. If α is not a p-adic integer (it has some non-zero digits to the right of the p-adic comma), then multiplication with α increases the norm. If α is a p-adic integer whose rightmost p-adic digit is not 0, then multiplication with α does not change the norm.

Therefore, as the length of the word increases, the norm increases or decreases monotonically or does not change, which eliminates the possibility to recognize C_n for $n > 0$. □

Chapter 4

Frequency Finite Automata

In this chapter we define and explore two-way frequency finite automata. Frequency finite automata have already been considered earlier in one-way setting. We show that the two-way version exhibits a different behavior and show the relationship between two-way frequency finite automata and automata with linearly bounded counters.

The chapter is organized as follows. In Section 4.1 we give a quick introduction to the frequency computation. In Section 4.2 we define the two-way frequency finite automata and in Section 4.3 we prove results about them and their relationship to automata with linearly bounded counters and finally, prove that any language from *LOGSPACE* can be (m, n)-recognized by a two-way frequency finite automaton with arbitrary high $\frac{m}{n}$.

4.1 Introduction

The notion of frequency computation was introduced by G. Rose [54]. For natural numbers m, n $(1 \leq m \leq n)$ a function $f : \mathbb{N} \to \mathbb{N}$ is (m, n)-computable iff there is a total recursive function $T : \mathbb{N}^n \to \mathbb{N}^n$ such that if $T(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$ for pairwise distinct $x_1, x_2, \ldots, x_n$ then at least m of the equations $f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_n) = y_n$ hold. If the function is of type $f : \mathbb{N} \to \{0, 1\}$ then we speak of a special case of (m, n)-computable sets.

In his survey McNaughton [47] cited a natural question by Myhill, for which values of m, n the (m, n)-computable functions are recursive. It was answered by in 1963 by Trakhtenbrot [57] by showing that a (m, n)-computable function f is recursive iff $2m > n$. In particular there are uncountably many $(1, 2)$-computable functions (and sets).

For the case when $2m \leq n$ and $2m' \leq n'$ Degtev [13] showed some special cases where (m, n)-computable sets are different from (m', n')-computable sets. Later Kummer and Stephan [42] showed that for any $(m, n) \neq (m', n')$ with $2m \leq n$ and $2m' \leq n'$ the classes of (m, n)-computable sets and (m', n')-computable sets are different.

A natural approach to further explore the frequency computation is to add some restrictions on the resources (time, space) of the frequency algorithm. Polynomial-time frequency computations were considered by Kummer and Stephan [42] and later by Hinrichs and Wechsung [27].

Restricting the computation to a deterministic finite automaton gives rise to the notion of regular frequency computation. Frequency computation with a deterministic finite automaton was studied by Kinber [37], and subsequently by Austinat et al. [4, 3, 5]. It turns out that for finite automata the analog of Trakhtenbrot's theorem holds – with frequency (m, n) with $2m > n$ only regular languages can be recognized, and there exist uncountably many $(1, 2)$-computable languages. Freivalds et al. [22] examined the state complexity of frequency finite automata.

All of the previously mentioned research on frequency finite automata considered one-way deterministic finite automaton which in each step simultaneously reads exactly one symbol from every input tape (and the shorter input words are padded with a special symbol \$ to make all input words of the same length). In this chapter we introduce and examine two-way finite automata which can move their heads asynchronously on the input tapes. Although classical two-way finite deterministic automata can only recognize regular languages, it turns out that for frequency computation for any $n > 0$ even with the frequency $(n-1, n)$ a larger class of languages can be $(n-1, n)$-recognized.

4.2 Definitions

Definition 4.1. *For natural numbers m, n $(1 \leq m \leq n)$ a two-way (m, n)-frequency finite automaton ((m, n)-2FFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where*

Q is the finite set of states,

Σ is the input alphabet,

$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\})^n \to Q \times \{L, N, R\}^n$ is the transition function, where $\vdash, \dashv \notin \Sigma$ are the left and right endmarkers, respectively,

$q_0 \in Q$ is the starting state, and

$F : Q \to \{0, 1\}^n$ is the acceptance function.

The automaton has n input tapes and n two-way heads – one for each tape. It can move the heads asynchronously on the input tapes (but the heads must stay within the words boundaries).

The n input words enclosed with the endmarkers are placed on the n tapes of the automaton. The automaton is started in the starting state q_0 with all its heads on the first letters of all input words. At each step the transition function δ is applied.

If q is the current state of the automaton and $a_1, \ldots, a_n$ are the symbols under its heads, and $\delta(q, (a_1, \ldots, a_n)) = (q', (d_1, \ldots, d_n))$ then the next state is q' and for each $1 \leq i \leq n$ the i-th head is moved to the left, stays in the same position or is moved to the right, if d_i is L, N or R, respectively. If $a_i = \vdash$ or $a_i = \dashv$ then $d_i \in \{N, R\}$ or $d_i \in \{L, N\}$, respectively, with the exception that if for all $1 \leq i \leq n$ the $a_i = \dashv$ the automaton is allowed to make a transition with $d_i = R$ for all $1 \leq i \leq n$, in which case we say that the automaton has stopped in state q' and given an output $F(q')$.

Therefore the output of the automaton on input words $x_1, \ldots, x_n$ is $F(\delta^*(q_0, (\vdash x_1 \dashv, \ldots, \vdash x_n \dashv)))$ where δ^* is the natural extension of δ to words.

We say that a language $L \subseteq \Sigma^*$ is recognized by an (m,n)-*2FFA* $\mathcal{A}$ if for every n distinct input words $x_1, \ldots, x_n \in \Sigma^*$ the automaton $\mathcal{A}$ when started on $x_1, \ldots, x_n$ gives an output $(y_1, \ldots, y_n) \in \{0,1\}^n$ such that at least m of the following hold:

$$\begin{aligned} y_1 = 1 &\Leftrightarrow x_1 \in L \\ y_2 = 1 &\Leftrightarrow x_2 \in L \\ &\vdots \\ y_n = 1 &\Leftrightarrow x_n \in L \end{aligned}$$

It should be noted that it is possible for an (m,n)-*2FFA* to recognize more than 1 language.

Definition 4.2. *For an automaton class $\mathcal{X}$ we define $\mathcal{L}(\mathcal{X})$ as the class of languages recognizable by some automaton from the class $\mathcal{X}$:*

$$\mathcal{L}(\mathcal{X}) = \{L \mid \exists \mathcal{A} \in \mathcal{X} \ \mathcal{A} \text{ recognizes } L\}$$

Definition 4.3. $REG = \mathcal{L}(1DFA)$

4.3 Results

As the next theorem shows, with the maximal frequency (n,n) only regular languages can be recognized.

Theorem 4.4. $\mathcal{L}((n,n)\text{-}2FFA) = \mathcal{L}((1,1)\text{-}2FFA) = REG$

Proof. We will show that it is possible to build a $(1,1)$-*2FFA*, say $\mathcal{A}_1$, which utilizes the fact that the language can be recognized by an (n,n)-*2FFA*, say $\mathcal{A}$, therefore $\mathcal{L}((1,1)\text{-}2FFA) \supseteq \mathcal{L}((n,n)\text{-}2FFA)$. Let $w_1, \ldots, w_{n-1} \in \Sigma^*$ be some fixed words. $\mathcal{A}_1$ when given an input x can simulate the computation of $\mathcal{A}$ on $n-1$ fixed inputs $w_1, \ldots, w_{n-1}$ and x as the n-th input and give the same output as $\mathcal{A}$'s n-th output. As the words $w_1, \ldots, w_{n-1}$ are fixed, they can be built in the states of $\mathcal{A}_1$ and therefore do not give any advantage to $\mathcal{A}$ when recognizing L. See Figure 4.1 for a schematic representation.

Also if there is a $(1,1)$-*2FFA* that recognizes L, then it is possible to build an (n,n)-*2FFA* recognizing L which applies the $(1,1)$-algorithm to each of the tapes separately, therefore $\mathcal{L}((n,n)\text{-}2FFA) \supseteq \mathcal{L}((1,1)\text{-}2FFA)$.

As $(1,1)$-*2FFA* is essentially a *2DFA*, it follows that $\mathcal{L}((n,n)\text{-}2FFA) = \mathcal{L}((1,1)\text{-}2FFA) = \mathcal{L}(2DFA) = REG$. □

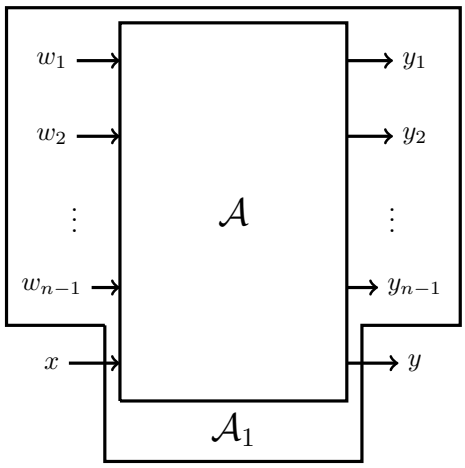

Figure 4.1: A schematic representation of $(1,1)$-*2FFA* $\mathcal{A}_1$ which simulates (n,n)-*2FFA* $\mathcal{A}$.

It turns out that with $m < n$ the (m,n)-*2FFA*s are related to finite automata with linearly bounded counters.

Definition 4.5. *For any $k \geq 1$, a two-way deterministic k-counter automaton ($2CA(k)$) is a 2DFA equipped with k counters. The values of the counters can be incremented and decreased by 1 and checked for 0. The values of the counters are strictly non-negative, i.e. the counter cannot be decreased if the counter already contains 0.*

If the value of the counter never exceeds the length of the input then we call the counter linearly bounded. We denote an automaton with k linearly bounded counters by $2BCA(k)$.

If a language can be recognized with a $2BCA(k)$ then for any $n > k$ it can be recognized with an $(n-k,n)$-*2FFA*.

Theorem 4.6. *For $n > k$:*

$$\mathcal{L}\left((n-k,n)\text{-}2FFA\right) \supseteq \mathcal{L}\left(2BCA(k)\right)$$

Proof. The idea of the proof is that the $(n-k,n)$-*2FFA* can determine which are the k longest words of the n input words and then use them as linearly bounded counters when recognizing each of the $n-k$ other words.

In the beginning the automaton sweeps over the input words to determine k longest words (if there are more than k such words (i.e., some of the words are of equal length), it chooses the first k). Then it processes each of the $n-k$ shortest words separately, one by one, using the chosen k longest words as counters. For each word used as a counter, the automaton positions its head on the left endmarker $\vdash$, representing the counter value 0. Increasing the counter is implemented by moving the head one position to the right, and decreasing – one position to the left (if the symbol under the head is not $\vdash$). Checking for zero can be done by checking if the symbol under the head is $\vdash$. As the counters are linearly bounded, the length of the longest words suffices to store the appropriate value of the counter when using them for each of the shorter words. □

It is well known that an automaton with even 2 unbounded counters can already simulate universal computation of a Turing machine [48]. However the situation with linearly bounded counters is very different.

It can be easily seen that a deterministic multi-head automaton with $k+1$ heads can simulate a $2BCA(k)$ by using the positions of k heads to store the values of the counters. For the other direction, it was proved in [52] that for special class of languages, called bounded languages (see the definition below), a $2BCA(k)$ can recognize the same languages as deterministic automaton with $k+1$ heads.

Definition 4.7. *A language $L \subseteq \Sigma^*$ is* bounded *if $L \subseteq a_1^* a_2^* \dots a_m^*$ for distinct symbols $a_1, a_2, \dots, a_m \in \Sigma$.*

Therefore a $2BCA(1)$ for languages over a unary alphabet are equivalent to an automaton with 2 heads. An alternative representation is by the automata over two-dimensional input tape (picture walking automata) recognizing squares of size $n \times n$ for different n. Such automata (in one representation or another) are studied in [51, 34, 35, 9] and others. It has been proven that such automata can recognize some highly non-trivial languages.

Corollary 4.8. *For any $n > 1$ the languages*

$$\left\{1^{2^m} \mid m \geq 0\right\},$$

$$\left\{1^{2^{2^m}} \mid m \geq 0\right\},$$

$$\left\{1^{42^{m^2}} \mid m \geq 0\right\},$$

$$\left\{1^{11^p} \mid p \text{ is a prime}\right\},$$

$$\left\{0^m 1^{m^2} \mid m \geq 0\right\},$$

$$\left\{0^m 1^{2^m} \mid m \geq 0\right\}$$

can be recognized by an $(n-1, n)$-2FFA.

An open question is whether an $(n-k, n)$-*2FFA* can recognize any language that cannot be recognized by k linearly bounded counters. The following theorem provides a sufficient condition for it.

Theorem 4.9.

$$\left(\mathcal{L}\left(2BCA(k+1)\right) \setminus \mathcal{L}\left(2BCA(k)\right)\right) \cap \mathcal{L}\left((1, k+1)\text{-}2FFA\right) \neq \emptyset \Rightarrow$$
$$\forall n > k \quad \mathcal{L}\left((n-k, n)\text{-}2FFA\right) \supsetneq \mathcal{L}\left(2BCA(k)\right)$$

Proof. We will show that if there is a language L such that:

(a) L is recognizable by a $2BCA(k+1)$,

(b) L is not recognizable by a $2BCA(k)$, and

(c) L is recognizable by a $(1, k+1)$-*2FFA*,

then:

(i) for every $n > k$, L is recognizable by an $(n-k,n)$-*2FFA*

(ii) L is not recognizable by a $2BCA(k)$.

The (ii) part of the conclusion actually is also the assumption (b), therefore we only have to show (i) that L is recognizable by an $(n-k,n)$-*2FFA*.

The idea is again simple. The $(n-k,n)$-*2FFA* uses the $k+1$ longest words as counters to determine whether the $n-k-1$ shortest words belong to L (as L can be recognized with $k+1$ linearly bounded counters by the assumption (a)), similarly as in the proof of Theorem 4.6. When the membership of the $n-k-1$ shortest words are determined, the automaton on the $k+1$ longest words that were used as counters applies the $(1,k+1)$-recognizing algorithm (from the assumption (c)). Therefore the automaton correctly $(n-k,n)$-recognizes L. □

Let $LOGSPACE = DSPACE(\log n)$ be the class of languages recognizable by a deterministic Turing machine in space $O(\log n)$. It is known that automata with linearly bounded counters characterize $LOGSPACE$, i.e. $LOGSPACE = \bigcup_{k>1} \mathcal{L}(2BCA(k))$ (see [29] for the observation of this equality).

Corollary 4.10.

$$\forall L \in LOGSPACE \;\; \exists k \; \forall n > k \;\; L \in \mathcal{L}((n-k,n)\text{-}2FFA)$$

This result shows that the frequency expressed as a fraction $\frac{m}{n}$ does not describe the power of (m,n)-computability because for any $L \in LOGSPACE$ this fraction can be made arbitrary close to 1. In this case the number of allowed errors $n-m$ seems to be more appropriate indicator of the capabilities of (m,n)-computability.

Part II

Unconventional Algorithms

Overview of Part II

In this part we perturbate the definition of an algorithm in two different ways and inspect the properties of the resulting unconventional types of computations.

In Chapter 5 we introduce ultrametric query algorithms which use p-adic numbers to describe the branching of a query algorithm. We investigate the respective complexity measure – ultrametric query complexity. We show that unrestricted ultrametric query algorithms are very powerful, for some functions requiring only 1 query. We also show that the exact ultrametric query complexity of a Boolean function is at most its polynomial degree, which means that it is at most 2 times larger than the exact quantum query complexity.

In Chapter 6 we generalize the notion of frequency computation to structured frequency computation. For the classical frequency computation an important threshold was $\frac{1}{2}$ – if the frequency $\frac{m}{n} \leq \frac{1}{2}$ then a continuum of sets can be (m,n)-computed, but with $\frac{m}{n} > \frac{1}{2}$ the (m,n)-computable sets are exactly the recursive sets. For the structured frequency computation we show a somewhat similar threshold of $\frac{\sqrt{n}}{n}$–we show a structure S of size $O(\sqrt{n})$ such that only recursive sets can be S-computed, and show that any overlapping structure (a condition that implies that only recursive sets can be S-computed) has size at least $\sqrt{n}$. We also examine the graph frequency computation in which the size of the structure is fixed to 2 and therefore conveniently described by the edges of a graph.

Chapter 5

Ultrametric Query Algorithms

In this chapter we introduce ultrametric query algorithms and prove some results about ultrametric query complexity. The notion of ultrametric query algorithms is based on p-adic numbers. The p-adics were also used in Chapter 2 to define ultrametric automata.

The chapter is organized as follows. In Section 5.1 we give some introduction to query algorithms and query complexity. In Section 5.2 we define ultrametric query algorithms, and in Section 5.3 we prove results about them.

5.1 Query Algorithms

Conventionally the query algorithm model considers a Boolean function $f : \{0,1\}^n \to \{0,1\}$ with the values of its arguments $x_1, x_2, \ldots, x_n$ locked in a black box. The definition of the function is known, but the values of the arguments are unknown. Queries can be made to access the values of arguments in the black box, with each query returning the value of one argument. The task of the algorithm is to compute the result of $f(x_1, x_2, \ldots, x_n)$ by making as few queries as possible. Each query can be dependent on the results of previous queries.

Deterministic query algorithms can be represented by binary decision trees. For each query there are exactly 2 branches that represent the next actions carried out by the algorithm depending on whether the queried variable was 0 or 1. The complexity of an algorithm is equal to the greatest number of queries made to compute the result of the function on arbitrary input data (the depth of the decision tree). The deterministic query complexity of a function f (denoted by $D(f)$) is defined as the minimum complexity of an algorithm over all possible algorithms which compute f.

Randomized query algorithms on the other hand are not necessarily binary. Each query can be followed by an arbitrary number of branches each carried out with a probability between 0 and 1 with the sum of probabilities for each result of the query being equal to 1. On each input x the randomized decision tree outputs 0 or 1 with certain probability. We say that randomized decision tree computes f with bounded-error if its output equals $f(x)$ with probability at least $\frac{2}{3}$ for all $x \in \{0,1\}^n$, and the tree computes f exactly if its output

equals $f(x)$ with probability 1 for all $x \in \{0,1\}^n$. $R_2(f)$ denotes the maximum number of queries made by an optimal randomized decision tree that computes f with bounded-error [11]. $R_0(f)$ denotes the expected number of queries (on the worst-case input) made by an optimal randomized decision tree that computes f exactly.

In quantum query complexity the variables are allowed to be queried in superposition. The basis states of an algorithm are $|i, b, z\rangle$. A query of the input is a unitary transformation O which maps a state $|i, b, z\rangle$ to $|i, b \oplus x_i, z\rangle$. A T-query quantum decision tree corresponds to a sequence of unitary transformations $U_0, O, U_1, O, U_2, O, \ldots, O, U_{T-1}, O, U_T$ where $U_0, U_1, \ldots, U_T$ are fixed unitary transformations that do not depend on x. The output is obtained by measuring the final state and outputting the rightmost bit of the observed basis state. We say that a quantum decision tree computes f with bounded-error if the output equals $f(x)$ with probability at least $\frac{2}{3}$, for all $x \in \{0,1\}^n$. The tree computes f exactly if the output equals $f(x)$ with probability 1, for all $x \in \{0,1\}^n$. $Q_E(f)$ denotes the number of queries of an optimal quantum decision tree that computes f exactly, and $Q_2(f)$ is the number of queries of an optimal quantum decision tree that computes f with bounded-error [11].

5.2 p-ultrametric Query Algorithms

The definition of p-ultrametric query algorithms is based on the definition of probabilistic query algorithms in which probabilities are replaced with rational number amplitudes. The amplitude of a state is computed similarly to probability to reach the state in a probabilistic algorithm - when the algorithm transitions from state A to B, the amplitude of state A is multiplied by the transition's amplitude, and the result is used as the amplitude of state B. If B has multiple incoming transitions, multiplication is performed on each state/transition pair separately and the results are added together to form the amplitude of B. The algorithm also has one or more end states, with no outgoing transitions. The result of the algorithm is computed by computing the p-norm of the amplitudes of the end states. The algorithm returns a result 0 or 1 depending on whether the p-norm exceeds a previously defined threshold value.

The algorithm model obtained this way is in many ways similar to the model of quantum query algorithms since amplitudes in both models are treated likewise [1] [60]. Also the complexity of a p-ultrametric query algorithm is defined similarly as the maximum number of consequent queries over all branches.

A p-ultrametric algorithm is described by a directed acyclic graph (DAG). There is exactly one vertex (root) which has no incoming edges. The nodes with no outgoing edges are leafs and they are the final (accepting) states of the algorithm. The inner nodes can be of two types—ones which query the input variables and others which do not.

Definition 5.1 ([21])**.** *The total amplitude of the root is defined to be* 1*. Furthermore, let v be a node at depth d in the DAG, let α be its total amplitude, and let $\beta_1, \ldots, \beta_k$ be the amplitudes corresponding to the outgoing edges $e_1, \ldots, e_k$ of v. Let $v_1, \ldots, v_k$ be the nodes where the edges $e_1, \ldots, e_k$ point to. Then the total amplitude of v_l, $l \in \{1, \ldots, k\}$, is defined as follows.*

1. *If the indegree of v_l is one, then its total amplitude is $\alpha\beta_l$.*

2. *If the indegree of v_l is larger than one, i.e., if two or more computation paths are joined, say m paths, then let $\alpha, \gamma_2, \ldots, \gamma_m$ be the corresponding total amplitudes of the predecessors of v_l and let $\beta_l, \delta_2, \ldots, \delta_m$ be the amplitudes of the incoming edges The total amplitude of the node v_l is then defined to be $\alpha\beta_l + \delta_2\gamma_2 + \cdots + \delta_m\gamma_m$.*

At the end of the algorithm the norms of the amplitudes in the final states are summed to obtain a value which is then compared with the threshold value.

Let L be the set of leaves of the DAG representing the p-ultrametric query algorithm and let $\alpha(v)$ denote the total amplitude of a state v. Then if $\sum_{v \in L} \|v\|_p$ exceeds the threshold λ then the algorithm outputs 1, otherwise 0. We say that an algorithm A computes a function $f : \{0,1\}^n \to \{0,1\}$ if on every input $x \in \{0,1\}^n$ its output coincides with the value of the function on this input.

A special case of p-ultrametric query algorithms are the algorithm for which the DAG contains exactly one leaf.

Definition 5.2. *We say that a p-ultrametric query algorithm is* ***one-endpoint*** *if it has exactly one accepting state.*

Note that if all the transition amplitudes of a p-ultrametric query algorithm are p-adic integers then the amplitudes at any state are p-adic integers as well. Therefore, if such algorithm is one-endpoint then the norm of the final state is always between 0 and 1 (inclusive), similarly as the probability in quantum and probabilistic query algorithms.

Definition 5.3. *We say that a p-ultrametric query algorithm is* ***exact*** *if for every input the sum of norms of the final amplitudes is either* 0 *or* 1.

The p-ultrametric query complexity (denoted by $U_p(f)$) of a function f is the complexity of an optimal p-ultrametric query algorithm which computes f. The exact p-ultrametric query complexity (denoted by $U_{p,E}(f)$) of a function f is the complexity of an optimal p-ultrametric query algorithm which exactly computes f. We denote the corresponding complexities for one-endpoint algorithms by $U_p^1(f)$ and $U_{p,E}^1(f)$. Of course, for every p and f it holds that $U_p(f) \leq U_{p,E}(f) \leq U_{p,E}^1(f)$ and $U_p(f) \leq U_p^1(f)$.

The algorithms described in this chapter are depicted using the following notation. A circle represents a state of the algorithm. If one or more variables are queried by the algorithm at this state, they are indicated in the circle. A double-circled state represents an end state of the algorithm. Lines represent transitions between states with the labels representing the transition amplitudes. If a transition is made only for some specific values of the queried variables, it is specified in the label.

5.3 Results

Ultrametric realizations of query algorithms for different Boolean functions show a dramatic decrease in complexity when compared to corresponding deterministic query algorithms.

It is known that any Boolean function can be represented by a multilinear real polynomial and this representation is unique for any Boolean function. Let $deg(f)$ be the degree of the unique multilinear polynomial representing the

Boolean function f. The degree is polynomially related to the query complexity measures (when considering total Boolean functions). It is known that $deg(f) \leq D(f) \leq 2deg(f)^4$ and also $deg(f) \leq 2Q_E(f)$ [50, 11].

The next theorem shows that $U^1_{p,E}(f) \leq deg(f)$:

Theorem 5.4. *For every function f with $deg(f) = k$, for every prime p there exists a one-endpoint exact p-ultrametric query algorithm computing the function f with complexity k.*

Proof. It is easy to see that the coefficients of the polynomial representation of a Boolean function are always integers. The algorithms makes a branch for every term in the polynomial with an amplitude that corresponds to the coefficient of this term. In that branch it queries all the variables in the term and if they are all equal to 1 then makes a transition to the final state. It follows that the final state contains amplitude 1 if $f(x) = 1$ and 0 otherwise. As for any p it holds that $\|0\|_p = 0$ and $\|1\|_p = 1$, the algorithm exactly computes the function f. □

Corollary 5.5. $U^1_{p,E}(f) \leq 2Q_E(f)$

However for some functions we can do even better. For example let us consider the n-ary XOR function which computes the parity of n bits (let us denote it by XOR_n). It is known that $deg(XOR_n) = n$ and quantumly XOR of n bits can be exactly calculated with $\lceil \frac{n}{2} \rceil$ queries, i.e., $Q_E(XOR_n) = \lceil \frac{n}{2} \rceil$. Most notably a quantum query algorithm can exactly compute the binary XOR_2 with just 1 query. We show that for any n there exists a 2-ultrametric algorithm with complexity 1 which computes XOR_n.

Theorem 5.6. *For every n there exists a one-endpoint 2-ultrametric query algorithm with complexity 1 for XOR_n.*

Proof. The algorithm splits into n branches with amplitude 1 and in each of them queries one variable. If the value of the queried variable x_i is equal to 1, this amplitude goes to the final state. If $XOR_n(x) = 1$ then the rightmost digit in the 2-adic representation of the amplitude in final state is 1 and therefore the norm is equal to 1. Otherwise the rightmost digit is 0 and the norm is less or equal to $\frac{1}{2}$. Therefore the threshold value can be chosen to be between $\frac{1}{2}$ and 1.

The algorithm is depicted in Fig. 5.1. □

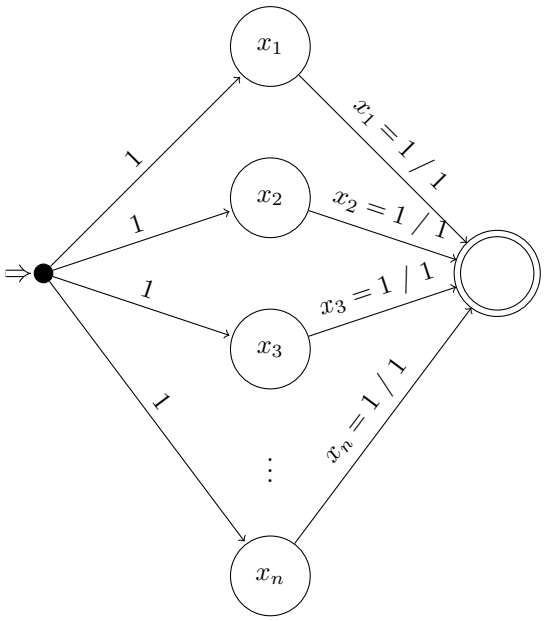

Figure 5.1: 2-ultrametric query algorithm for XOR_n.

The 2-ultrametric algorithm for XOR_n hints for a more general way of constructing 2-ultrametric algorithms with a small complexity.

Let us denote by $deg_2(f)$ the binary polynomial degree of function f, i.e., the minimal degree of a polynomial $p(x)$ such that $p(x) \equiv f(x) \pmod 2$.

Theorem 5.7. *For every function f with $deg_2(f) = k$, there exists a one-endpoint 2-ultrametric query algorithm with complexity k for the function f.*

Proof. The algorithm is similar to the algorithm for the representing polynomial with the difference that now at the final state the amplitude is not equal to the function value but merely congruent to it modulo 2. It is easy to see that the coefficients of the binary polynomial representation of a Boolean function are also integers. The algorithms makes a branch for every term in the polynomial with an amplitude that corresponds to the coefficient of this term. In that branch it queries all the variables in the term and if they are all equal to 1 then makes a transition to the final state. It follows that the final state contains amplitude α with $\alpha \equiv 0 \pmod 2$ if $f(x) = 1$ and $\alpha \equiv 1 \pmod 2$ otherwise.

For any $\alpha \equiv 0 \pmod 2$ it holds that $\|\alpha\|_2 \leq \frac{1}{2}$ and for any $\alpha \equiv 1 \pmod 2$, $\|\alpha\|_2 = 1$. Therefore setting the threshold to a value between $\frac{1}{2}$ and 1 makes the algorithm compute function f. □

However, for some functions we can do even better that that. For example if we consider the n-ary OR function (OR_n) it can be shown that the deterministic query complexity of this function is n and quantum query complexity of this function is $O(\sqrt{n})$. However, for every $p > n$ a corresponding one-endpoint exact p-ultrametric query algorithm can be constructed with complexity equal to 1:

Theorem 5.8. *For every prime $p > n$ there exists a one-endpoint exact p-ultrametric query algorithm with complexity 1 for OR_n.*

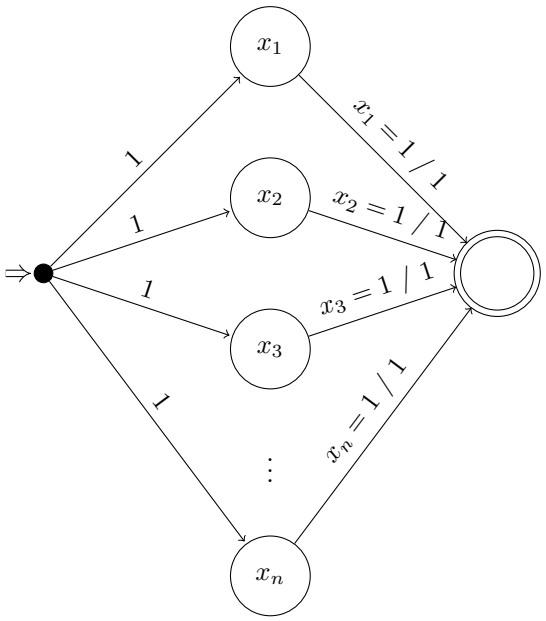

Figure 5.2: p-ultrametric query algorithm for OR_n.

Proof. The algorithm splits into n branches with amplitude 1 and in each of them queries one variable. If the value of the queried variable x_i is equal to 1, this amplitude goes to the final state.

If $OR_n(x) = 1$ then the final state contains amplitude α which can be represented by a natural number k with $1 \leq k < p$, therefore $\|\alpha\|_p = 1$. If $OR_n(x) = 0$ then the final state contains amplitude 0 and $\|0\|_p = 0$. Therefore, the algorithm exactly computes OR_n

The algorithm is depicted in Fig. 5.2. Note that the DAG of the algorithm looks exactly the same as the DAG of the algorithm for XOR_n. The only difference is the choice of the parameter p and the threshold value. □

An interesting class of functions are those for which there exist p-ultrametric query algorithms with a small complexity for a specific p.

Let us consider the following function. Denote by $\overline{x} = \overline{x_n x_{n-1} \dots x_1}$ the number $x_1 + 2x_2 + 2^2 x_3 + \dots + 2^{n-1} x_n$ whose binary representation is $x_n x_{n-1} \dots x_1$. The function is defined as follows:

$$NDIV_{n,p}(x) = \begin{cases} 0, & \text{if the number } \overline{x_n x_{n-1} \dots x_1} \text{ is divisible by } p \\ 1, & \text{otherwise} \end{cases}$$

Theorem 5.9. *For any n and any prime p there exists a one-endpoint p-ultrametric query algorithm with complexity 1 that computes $NDIV_{n,p}$.*

Proof. The algorithm splits into n branches with amplitudes $2^0, 2^1, 2^2, \dots, 2^{n-1}$ and in each of them queries one variable. If the value of the queried variable x_i is equal to 1, this amplitude goes to the final state. Therefore the amplitude in the final state equals $\overline{x} = x_1 + 2x_2 + 2^2 x_3 + \dots + 2^{n-1} x_n$. If it not divisible by p then the norm of the amplitude is equal to 1, otherwise the norm is at most

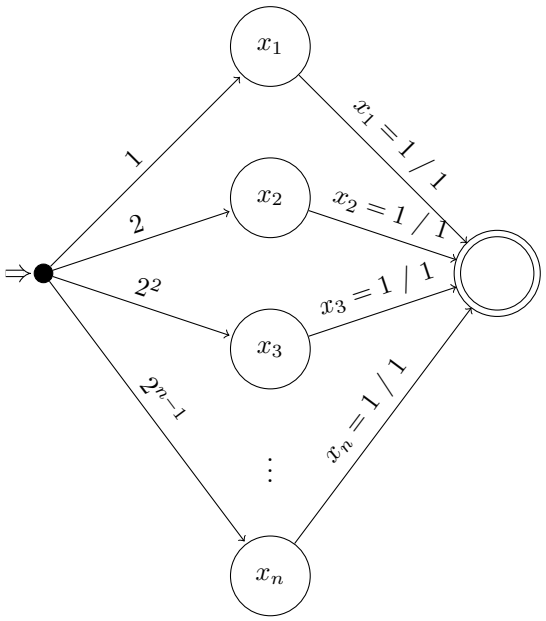

Figure 5.3: p-ultrametric query algorithm for $NDIV_{n,p}$.

$\frac{1}{p}$. Therefore a threshold value between $\frac{1}{p}$ and 1 gives a p-ultrametric query algorithm that computes $NDIV_{n,p}$. The algorithm is depicted in Fig. 5.3. □

Chapter 6

Structured Frequency Algorithms

In this chapter we introduce the notion of structured frequency algorithms by modifying the original definition given by G. Rose in 1960 [54].

The chapter is organized as follows. In Section 6.1 we give an introduction and define the structured frequency computation. In Section 6.2 we show that by using a structure defined by a projective plane we can require only $O(\sqrt{n})$ answers to be correct and still be able to compute only recursive sets – a counterpart to the Trakhtenbrot's result about frequency $\frac{m}{n} > \frac{1}{2}$. In Section 6.3 we consider structures defined by graphs.

The results of this chapter are a joint work in cooperation with Jānis Iraids and Rūsiņš Freivalds.

6.1 Introduction and Definitions

See the introduction of Chapter 4 for an overview of frequency computation. The original definition of frequency (m, n)-computation allows any m of the n outputs to be correct. We generalize the notion of frequency computation by demanding that from some fixed set of subsets of outputs at least 1 subset has only correct answers. Therefore the original frequency (m, n)-computation is a special case where a set containing every subset of size m is taken as the structure.

By $\mathbb{N} = \{0, 1, 2, \dots\}$ we denote the set of nonnegative integers and $\mathbb{B} = \{0, 1\}$. $[n] = \{0, 1, 2, \dots, n-1\}$. We use $|X|$ to denote the cardinality of a set X.

Let $A \subseteq \mathbb{N}$ be a set. By $\chi_A : \mathbb{N} \to \mathbb{B}$ we denote the *characteristic function* of A:

$$\chi_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

We say that a function f is *recursive* if there is an algorithm (Turing machine) that computes f. If χ_A is a total recursive function then we call the set A *recursive*.

Definition 6.1. *A set A is (m, n)-computable iff there is a total recursive function f which assigns to all distinct inputs $x_1, x_2, \ldots, x_n$ a binary vector $(y_1, y_2, \ldots, y_n)$ such that at least m of the equations $\chi_A(x_1) = y_1, \chi_A(x_2) = y_2, \ldots, \chi_A(x_n) = y_n$ hold.*

By a *structure* of a finite set K we call a set of K's subsets $S \subseteq 2^K$.

We assume that the elements of K are ordered under some fixed ordering $\phi : K \to [n]$ where $n = |K|$.

Definition 6.2. *A set A is (S, K)-computable (or computable with a structure S) iff there is a total recursive function f which assigns to all distinct inputs $x_1, x_2, \ldots, x_n$ a binary vector $(y_1, y_2, \ldots, y_n)$ such that $\exists B \in S\ \forall b \in B\ \chi_A(x_{\phi(b)}) = y_{\phi(b)}$*

It can be seen that (m, n)-computability is a special case of (S, K)-computability by taking S to be the set of all subsets of K of size m.

6.2 Projective Plane Frequency Computation

In finite geometry, the *Fano plane* (named after Gino Fano) is the finite projective plane of order 2, having the smallest possible number of points and lines. This plane has 7 points and 7 lines with 3 points on every line and 3 lines through every point. Every two points are on a unique line and every two lines intersect in a unique point. See Fig. 6.1.

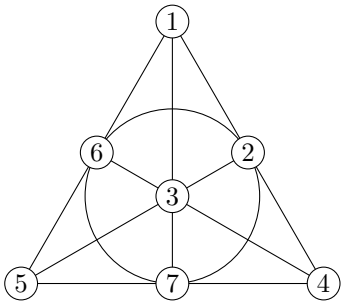

Figure 6.1: The Fano Plane

We consider the first example of a structured frequency computability using the Fano plane.

Definition 6.3. *A set A is* Fano*-computable iff there exists a total recursive function $f : \mathbb{N}^7 \to \mathbb{B}^7$ which assigns to all 7-tuples $(x_1, x_2, \ldots, x_7) \in \mathbb{N}^7$ of distinct inputs a binary vector $\{y_1, y_2, \ldots, y_7\}$ such that*

$$(y_1 = \chi_A(x_1) \wedge y_2 = \chi_A(x_2) \wedge y_4 = \chi_A(x_4)) \vee$$

$$\vee (y_2 = \chi_A(x_2) \wedge y_3 = \chi_A(x_3) \wedge y_5 = \chi_A(x_5)) \vee$$

$$\vee (y_3 = \chi_A(x_3) \wedge y_4 = \chi_A(x_4) \wedge y_6 = \chi_A(x_6)) \vee$$

$$\vee (y_4 = \chi_A(x_4) \wedge y_5 = \chi_A(x_5) \wedge y_7 = \chi_A(x_7)) \vee$$

$$\vee(y_5 = \chi_A(x_5) \wedge y_6 = \chi_A(x_6) \wedge y_1 = \chi_A(x_1))\vee$$
$$\vee(y_6 = \chi_A(x_6) \wedge y_7 = \chi_A(x_7) \wedge y_2 = \chi_A(x_2))\vee$$
$$\vee(y_7 = \chi_A(x_7) \wedge y_1 = \chi_A(x_1) \wedge y_3 = \chi_A(x_3))$$

It can be seen that the required fraction of correct answers for *Fano*-computability is $\frac{3}{7} < \frac{1}{2}$. Contrary to the (m,n)-computability however only recursive sets are *Fano*-computable.

Theorem 6.4. *A set A is* Fano*-computable iff it is recursive.*

The proof of this theorem is a special case of Theorem 6.8 below.

We want to explore further how much smaller can we get this fraction, i.e., for how small fraction of the inputs can we require the algorithm to give the correct answers so that the computed set can still only be recursive? Recall, that for "unstructured" frequency computations the answer is $\frac{1}{2}$ – if $\frac{m}{n} \leq \frac{1}{2}$ then a continuum of sets can be (m,n)-computed even with a finite automaton, but if $\frac{m}{n} > \frac{1}{2}$ then every (m,n)-computable set is recursive. Surprisingly for the structured frequency computation we can get this fraction close to $\frac{\sqrt{n}}{n}$ as n tends to infinity by extending the *Fano*-computability example.

Of course, it is possible "cheat", for example, by requiring that the algorithm on every input $(x_1, x_2, \ldots, x_n)$ outputs $(y_1, y_2, \ldots, y_n)$ such that $y_1 = \chi_A(x_1)$ and have no requirements for the other y_i's therefore attaining even a fraction of $\frac{1}{n}$ while every such computable set A is recursive (as on the first input the algorithm always has to output the correct answer). However we would like to avoid such cases because if we only look at the first input x_1 the fraction of correct answers there is $\frac{1}{1}$ which is the maximal possible. To avoid such "cheating" we introduce the notion of size consistency.

Definition 6.5. *By the* size *of a structure $S \subseteq 2^K$ we denote the size of the smallest subset* - $\min_{A \in S} |A|$. *We call the structure* size consistent *iff* $\neg\exists K' \subseteq K \ \min_{A' \in S} \frac{|A' \cap K'|}{|K'|} > \min_{A \in S} \frac{|A|}{|K|}$

The size consistency means that there is no smaller subset $K' \subseteq K$ such that the minimal subset $A' \in S$ (respective to K') contains a larger fraction of elements of K' than the minimal subset $A \in S$ does for K. Therefore this excludes the unwanted cases mentioned earlier.

Now we introduce a new type of structures for which we prove that the computed set is guaranteed to be recursive.

Definition 6.6. *We call a structure $S \subseteq 2^K$* overlapping *iff* $\forall A, B \in S \ A \cap B \neq \emptyset$.

Theorem 6.7. *For any set K of size $n = q^2 + q + 1$ where q is a prime power there exists a size consistent overlapping structure of size $q + 1$.*

Proof. The reader might already be familiar with the concept of finite projective geometry. A finite projective plane is a finite set of points P and lines $L \subseteq 2^P$, such that

A1) For any two distinct points, there is exactly one line containing these points.

A2) For any two distinct lines, there is exactly one point common to these lines.

A3) There exist four points, no three of which are on a line.

For all $q \geq 2$ these axioms imply

B1) Each line contains exactly $q+1$ points.

B2) Each point is on exactly $q+1$ lines.

B3) There are exactly q^2+q+1 points in the projective plane.

It is known, that if q is a prime power there exists a finite projective plane denoted by $PG(2,q)$ with $|L| = |P| = q^2+q+1$ based on the finite field $\mathbb{F}_q$ with q elements. For a more detailed overview see, for example, [26].

Note, that $S = PG(2,q)$ is an overlapping structure with the points P playing the role of K and the lines L playing the role of S. For $PG(2,q)$: $|K| = |P| = q^2+q+1$ and all the lines contain exactly $q+1$ points. From A2) it follows that the lines of $PG(2,q)$ indeed make an overlapping structure.

To show that the projective plane $S = PG(2,q)$ is a size consistent structure, it is sufficient to count $\sum_{A\in S} |A \cap K'|$.; For any integer $m : 0 < m < n$, if we consider a subset K' of size $n-m$, from B2) follows that $\sum_{A\in S} |A \cap K'| = (q+1)n - (q+1)m$. By the pigeonhole principle, there exists a subset A, such that $|A \cap K'| \leq \frac{(q+1)n-(q+1)m}{n} = \frac{(q+1)(n-m)}{n}$. Therefore $\min_{A'\in S} \frac{|A'\cap K'|}{|K'|} \leq \frac{(q+1)(n-m)}{n(n-m)} = \frac{q+1}{n}$. □

Theorem 6.8. *If A is computable with an overlapping structure then A is recursive.*

The proof goes along the same lines of the proof of B.A.Trakhtenbrot for frequency $\frac{m}{n} > \frac{1}{2}$ [57, 23]. However, this result is more general.

Proof. We will use infinite binary trees whose vertices correspond to binary strings. The root corresponds to the empty string and for every other vertex v the corresponding string $s(v)$ is the string of its parent vertex concatenated with a 0 or 1 depending on which child is v:

$$s(v) = \begin{cases} s(v_p)0, & \text{if } v \text{ is the left child of } v_p \\ s(v_p)1, & \text{if } v \text{ is the right child of } v_p \end{cases}$$

We will use $v(x)$ to denote the x-th (0-based) symbol of $s(v)$. $dom(v) = [|s(v)|]$ Therefore an infinite branch $B = B_0B_1B_2\ldots$ defines a set whose characteristic function $\chi_B(x)$ is given by $B(x) = \lim_{n\to\infty} B_n(x)$. We use the same name for the set as for the branch.

Let $f : \mathbb{N}^n \to \mathbb{B}^n$ be the function that (S,K)-computes A with $|K| = n$ and some overlapping structure $S \subseteq 2^K$.

Consider a tree T which contains all $\sigma \in \{0,1\}^*$ satisfying the property that for all distinct $x_1,\ldots,x_n \in dom(\sigma)$ there exists $P \in S$ such that $(\sigma(x_1),\ldots,\sigma(x_n))$ coincides with $f(x_1,\ldots,x_n)$ in positions P.

T contains A as an infinite branch because f (S,K)-computes A.

Assume that another infinite branch B in T differs from A in n positions $x_1, x_2, \ldots, x_n$. Then $(B(x_1), B(x_2), \ldots, B(x_n))$ coincides with $f(x_1, x_2, \ldots, x_n)$ in some positions $P_1 \in S$. But $(A(x_1), A(x_2), \ldots, A(x_n))$ also coincides with $f(x_1, x_2, \ldots, x_n)$ in some positions $P_2 \in S$. As $\forall P_1, P_2 \in S\ P_1 \cap P_2 \neq \emptyset$ there is an x_i such that $A(x_i) = B(x_i)$ contradicting the assumption that A and B differs in $x_1, x_2, \ldots, x_n$. Therefore every infinite branch of T differs from A in at most $n-1$ positions.

Let B be an infinite branch that differs from A in maximum number of positions and let D be the finite set on which A and B differs. B is also an infinite branch in the subtree

$$T' = \{\sigma \in T \mid \forall x \in dom(\sigma) \cap D\ \ \sigma(x) = B(x)\}$$

Assume that C is another infinite branch in T'. Let x be such that $B(x) \neq C(x)$. From the definition of T' follows that $x \notin D$. But then C and A differ on $D \cup \{x\}$ thus contradicting the choice of B as an infinite branch that differs from A in maximum number of positions. Therefore B is the only infinite branch in T'.

For any vertex v the procedure of deciding whether v is in T' is recursive. The following algorithm computes $B(x)$ for any x:

> Search for the first $t > x$ such that all $\sigma \in T' \cap \{0,1\}^t$ take only a unique value $y = \sigma(x)$ at x. Output this y as the value of $B(x)$.

The returned value cannot be different from $B(x)$ as T' has at every length $t > x$ a string σ with $\sigma(x) = B(x)$. If the algorithm wouldn't terminate for some x then for every $t > x$ there would be σ with $\sigma(x) \neq B(x)$ and there would be an infinite subtree $T'' = \{\sigma \in T' \mid x \in dom(\sigma) \to \sigma(x) \neq B(x)\}$. By König's Lemma this subtree would contain an infinite branch C different from B contradicting the fact that B is the only infinite branch of T'.

Therefore B is recursive and as A differs from B in a finite set of positions A is also recursive. □

The following theorem shows that for overlapping structures the fraction obtained by the finite planes is close to the best possible.

Theorem 6.9. *Every size consistent overlapping structure $S \subseteq 2^K$ has size at least $\sqrt{n}$ where $n = |K|$.*

Proof. If S is size consistent, $\forall K' \subseteq K\ \min_{A' \in S} \frac{|A' \cap K'|}{|K'|} \leq \min_{A \in S} \frac{|A|}{|K|}$. In particular, take K' equal to a set of minimal size in S. Then

$$\frac{|K'|}{|K|} = \min_{A \in S} \frac{|A|}{|K|} \geq \min_{A' \in S} \frac{|A' \cap K'|}{|K'|} \geq \frac{1}{|K'|},$$

where the second inequality follows from the fact that S is overlapping, hence even K' has at least one element common with any other set from S. The size of the structure S is equal to the size of the smallest set in it – $|K'|$ and $|K'|^2 \geq |K| = n$. Therefore the size of the structure is at least $\sqrt{n}$. □

6.3 Graph Frequency Computation

If we consider structures with the sizes of all subsets equal to some $k \geq 1$, the first interesting case is with $k = 2$ (with $k = 1$ either there are some inputs

for which the outputs are not taken into account or it is the same as $(1, n)$-computability). A convenient and well-known way to represent such structures is using graphs.

Definition 6.10. *We call a structure $S \subseteq 2^K$ a* graph structure *iff $\forall A \in S \ |A| = 2$. For a graph $G = (V, E)$ by saying that a set A is G-computable we mean that A is (E, V)-computable.*

A natural question arises – for which graphs G are the G-computable sets recursive?

For some graphs G it is very easy to show that only recursive sets are G-computable.

Proposition 6.11. *If the graph G is either a triangle (C_3) or a star graph (S_k) then every G-computable set is recursive.*

Proof. The internal vertex of a star graph S_k is involved in every edge therefore on the input corresponding to this vertex the algorithm must always output the correct answer (on this vertex the algorithm $(1, 1)$-computes the set).

For a triangle graph C_3 if an algorithm C_3-computes a set A then it also $(2, 3)$-computes A. □

The following theorem shows a sufficient condition for a graph G to allow computability of non-recursive sets.

Theorem 6.12. *If a graph G contains as a subgraph a cycle of length 4 (C_4) or two vertex-disjoint paths of length 3 ($2P_3$) then there is a continuum of G-computable sets, namely, every $(1, 2)$-computable set is also G-computable.*

Proof. Assume there is an algorithm $\mathcal{A}_1$ that $(1, 2)$-computes a set A. For a graph G that contains a cycle of length 4 - $\{(1, 2), (2, 3), (3, 4), (4, 1)\}$ for some vertices $1, 2, 3, 4$ (see Fig. 6.2) consider the following algorithm – on inputs x_1 and x_3 output the values $(y_1, y_3) = \mathcal{A}_1(x_1, x_3)$ and on inputs x_2 and x_4 output the values $(y_2, y_4) = \mathcal{A}_1(x_2, x_4)$. At least one of the outputs y_1 and y_3 is correct and at least one of the outputs y_2 and y_4 is correct, therefore on at least one of the pairs of inputs $\{(1, 2), (2, 3), (3, 4), (4, 1)\}$ the outputs are correct.

Similarly for a graph G containing two vertex-distinct paths of length 3 - $\{(1, 2), (2, 3), (4, 5), (5, 6)\}$ for some vertices $1, 2, 3, 4, 5, 6$. Now the algorithm is to use $\mathcal{A}_1$ on pairs of inputs – (x_1, x_3), (x_4, x_6) and (x_2, x_5). In this case also there exists at least one pair of correct outputs corresponding to an edge of G. □

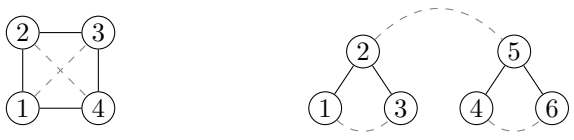

Figure 6.2: Continuum implying subgraphs. Dashed lines show on which pairs of vertices (inputs) to apply the $(1, 2)$-algorithm.

As shown by the following corollaries Theorem 6.12 discards many graphs as the potential candidates for a structure that allows only recursive functions.

Corollary 6.13. *If G contains more than one connected component of size ≥ 3 then every $(1,2)$-computable set is also G-computable.*

Corollary 6.14. *If G contains as a subgraph a cycle of length other than 3 or 5 then every $(1,2)$-computable set is also G-computable.*

The following two theorems show that graph structures two pairs () and three pairs () differ very much.

First we will need some lemmas. Let H be the 4-vertex graph with vertices $1,2,3,4$ and edges $(1,2)$, $(3,4)$ (i.e, the graph ).

Lemma 6.15. *If M is a Turing machine with 4 inputs and 4 outputs* H*-computes two distinct total functions $f(x)$ and $g(x)$ such that there exist d_0 and d_1 with properties $d_0 \neq d_1$, $f(d_0) \neq g(d_0)$ and $f(d_1) \neq g(d_1)$ then there exists an algorithmic procedure computing all the values of the functions f and g with at most one exception.*

Proof. With no restriction to generality, we can assume that $d_0 = 0$ and $d_1 = 1$.

We start with considering $(x_1 = 0, x_2 = 2, x_3 = 1, x_4 = 3)$. Since $f(0) \neq g(0)$ and $f(1) \neq g(1)$ but the two functions f and g are computed correctly, the values y_1, y_2 are to be correct values of one of these functions, and y_3, y_4 are to be correct values of the other function. If $y_1 = f(0)$ then $y_2 = f(2), y_3 = g(1), y_4 = g(3)$. If $y_1 = g(0)$ then $y_2 = g(2), y_3 = f(1), y_4 = f(3)$.

Next, we consider $(x_1 = 0, x_2 = 1, x_3 = 2, x_4 = 3)$. Three cases are possible. First, if $y_1 = f(0), y_2 = f(1)$ then $y_3 = g(2), y_4 = g(3)$. Second, if $y_1 = g(0), y_2 = g(1)$ then $y_3 = f(2), y_4 = f(3)$. Third, if neither $y_1 = f(0), y_2 = f(1)$ nor $y_1 = g(0), y_2 = g(1)$ then $y_3 = f(2) = g(2)$ and $y_4 = f(3) = g(3)$.

In any of these cases we have found either both $f(2)$ and $g(2)$, or both $f(3)$ and $g(3)$. Denote by $a \in \{2,3\}$ the value of x such that we have not yet found both $f(a)$ and $g(a)$. Then we go on considering the 4-tuples $(x_1 = 0, x_2 = a, x_3 = 1, x_4 = 4)$ and $(x_1 = 0, x_2 = 1, x_3 = a, x_4 = 4)$. This way, gradually we get all the values of the functions f and g l values of with at most one exception. □

Lemma 6.16. *If a Turing machine M with 4 inputs and 4 outputs is not correctly* H*-computing some total recursive function $f(x)$ then this property of M can be discovered considering only a finite number of 4-tuples $(x_1, x_2, x_3, x_4) \in \mathbb{N}^4$.*

Proof. By the definition, the machine M produces some result on arbitrary 4-tuple $(x_1, x_2, x_3, x_4) \in \mathbb{N}^4$. Since all such 4-tuples can be algorithmically enumerated, either a contradiction is found after a finite number of steps, or no contradiction is ever found and the function f is computed correctly. □

Lemma 6.17. *If $\alpha \in \{0,1\}^n$ is a finite binary word and if a Turing machine M with 4 inputs and 4 outputs is not correctly* H*-computing any total function $f(x)$ with values $f(0) = \alpha(0), f(1) = \alpha(1), \ldots, f(n-1) = \alpha(n-1)$, then this property of M can be discovered considering only a finite number of 4-tuples $(x_1, x_2, x_3, x_4) \in \mathbb{N}^4$.*

Proof. We consider an infinite binary tree representing all infinite binary sequences. If $\alpha \in \{0,1\}^n$ is a prefix of a function that is not correctly H-computed by M, then, by Lemma 6.16, this can be discovered considering only a finite

number of 4-tuples $(x_1, x_2, x_3, x_4) \in \mathbb{N}^4$. In our infinite binary tree we make a cut corresponding to this prefix α. By formulation of our Lemma, these cuts leave no infinite binary path in the tree. By König's lemma [38], every tree that contains infinitely many vertices, each having finite degree, has at least one infinite simple path. Hence after all the cuts in our infinite binary tree, there remain only a finite number of vertices. □

Now we consider a tree $\mathbb{T}$ of all the total functions H-computed by M. (Since we consider only functions $\mathbb{N} \to \{0, 1\}$, all the vertices of this tree have finite degree.) By Lemma 6.15, every function $g(x)$ correctly H-computed by the machine M differs from $f(x)$ at most for one value of x.

Lemma 6.18. *If a Turing machine M with 4 inputs and 4 outputs* H*-computes at least one total nonrecursive function $f(x)$, then the tree $\mathbb{T}$ either contains only a finite number of functions or $\mathbb{T}$ has only one accumulation point.*

Proof. Accumulation point of the tree $\mathbb{T}$ is an infinite path P such that for every prefix π of the path P there exists an infinite path Q distinct from P but also having the prefix π. Had there been two distinct accumulation points P and Q in $\mathbb{T}$, there would be two functions $f(x)$ and $g(x)$ and values d_0 and d_1 with properties $d_0 \neq d_1$, $f(d_0) \neq g(d_0)$ and $f(d_1) \neq g(d_1)$. However, then, by Lemma 6.15, all the functions H-computed by M would be recursive. □

Theorem 6.19. *If a Turing machine M with 4 inputs and 4 outputs correctly* H*-computes a total function then this function is recursive.*

Proof. Consider a tree $\mathbb{T}$ of all the total functions H-computed by M. If $\mathbb{T}$ contains only a finite number of functions then for each of these functions there is a prefix π which is not not a prefix of any other total function H-computed by M. If $\mathbb{T}$ has only one accumulation point then, by the construction of the tree $\mathbb{T}$ described in the proof of Lemma 6.17, we gradually construct initial fragments of $\mathbb{T}$. Since $\mathbb{T}$ has exactly one accumulation point, the accumulation point is always the path with the maximum other functions branching off this initial fragment of the path. Hence this path can be algorithmically constructed, and the function is recursive. □

Theorem 6.20. *If a graph G contains as a subgraph three vertex-disjoint paths of length 2 ($3P_2$) then there is a continuum of G-computable sets.*

(Computer-assisted) Proof. Consider a complete infinite binary tree T whose vertices are labeled with nonnegative integers. The root is labeled with 0. For each vertex labeled x its right child is labeled $2x + 1$ and its left child is labeled $2x + 2$. Therefore T contains all numbers in $\mathbb{N}$. If we fix an infinite branch B in T, it defines the set $L_B = \{x \mid x \in B\}$.

We will show that if G contains as a subgraph $3P_2$ then there is an algorithm which G-computes any set L_B, irrespective of which branch B is chosen. As there is a continuum different ways to choose a branch B, it will follow that there is a continuum of G-computable sets.

As a side note, we should note that this is also the way how to prove that there is a continuum of $(1, 2)$-computable sets. The algorithm $(1, 2)$-computing L_B is the following:
On inputs (x_1, x_2):

- if there is a branch which goes through both x_1 and x_2, then output $(1, 0)$, if $x_1 < x_2$, and $(0, 1)$, if $x_1 > x_2$
- otherwise, output $(0, 0)$

It can be checked that no matter how the branch B is chosen, at least one of these outputs will be correct.

If, instead of $(1, 2)$-computing, we consider computing with a graph with 3 pairs of connected vertices, the idea of the proof is the same, only now we have to deal with a larger number of different possibilities for the input instances. A single input instance can be represented as a 7-vertex rooted tree I in which all vertices except the root are divided into three pairs. The root of I represents a vertex prepended to the root of T and the 3 pairs of vertices represent the 3 pairs of inputs for the algorithm. For any vertices x_1, x_2 if x_1 is a descendant of x_2 in T then x_1 is also a descendant of x_2 in I. See Fig. 6.3 for an example instance.

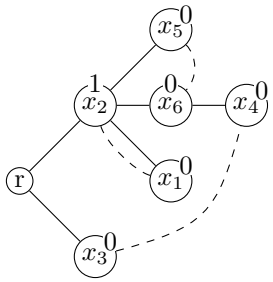

Figure 6.3: An example instance. The tree shows relationships between the inputs in the tree T. The dashed lines show how the inputs are distributed into pairs. The attached output labels $y_i \in \{0, 1\}$ show one possible assignment such that no matter how a branch B is chosen, there exists a pair for which both outputs are correct.

To correctly solve an instance I means to assign outputs $y_i \in \{0, 1\}$ to all non-root vertices so that, no matter which branch B is chosen, on at least one pair both outputs will be correct.

It was checked with a computer program that for each possible instance I there exists an assignment that solves it.

The pseudocode of the program:

for all 7-vertex trees T **do**
  for all possible divisions of T into root and 3 pairs of vertices **do**
    if there exists no assingment such that, no matter which branch B is chosen, there exists a pair with both correct outputs **then**
      return fail
    end if
  end for
end for
return success

□

6.4 Conclusions and Open Problems

We have introduced a new model of computability by extending the previously known frequency (m, n)-computability. We have shown some structures which lead to computability of only recursive sets and some structures which allow a continuum of computable sets. However, we are still far from a complete characterization of all structures.

Some open problems are:

- Are there any size consistent non-overlapping structures of size less than $\sqrt{n}$ that allow only computability of recursive sets? If so then what is the smallest possible fraction of correct answers attainable?
- For graph frequency computation obtain a complete classification of all graphs G and classes of G-computable sets.
- What other types of structures are interesting and worth considering and what classes of sets are computable with them?

Conclusion

In this thesis we have considered several types of unconventional finite automata, ranging from two-way probabilistic and alternating finite automata to ultrametric and frequency finite automata. We have also examined two types of unconventional computation – ultrametric query algorithms and structured frequency computation. In all considered areas we have compared how the unconventional models relate to classical models. Although there are many new results in this thesis, there are still a lot of connections to be found.

In the two-way finite automata size complexity theory we have shown a previously unknown relation between alternating and probabilistic automata. More specifically that there is a family of languages that is recognizable by a family of polynomial-size alternating automata, but for every family of fast probabilistic bounded-error two-way automata the sizes of the automata grow superpolynomially. Although in the literature there are several results showing advantages of probabilistic models of automata over non-probabilistic models of automata, this seems to be one of very rare examples where a class of non-probabilistic automata are shown to be more powerful than a class of probabilistic automata. It seems that the result could be strengthened to include also probabilistic automata that can work superpolynomial time or have non-isolated cutpoint, but it seems hard to prove it with the current techniques, therefore some new approach might be needed.

We have introduced the ultrametric finite automata. We think that for these the most perspective model is the regulated ultrametric automata as they can recognize exactly the regular languages and a bound on the complexity of simulating them with deterministic automata is given. It would be interesting to find some connections of how they relate to alternating and nondeterministic automata.

We have considered all of the above-mentioned types of automata for the counting problem. Most importantly we have shown optimal constant-size probabilistic and ultrametric regulated automata. However it is still open, if we require the two-way probabilistic automaton to be fast and have the error probability bounded by a constant, can it do any better than one-way automaton, i.e., have less than $O(\log^2 n/\log\log n)$ states.

We have introduced the two-way frequency finite automata. We have shown that any language recognizable with two-way automaton with k linearly bounded counters is $(n-k,n)$-recognizable by a two-way frequency finite automaton. This relation shows that many nontrivial languages are $(n-1,n)$-recognizable. However, it is an open question whether there are any other languages $(n-k,n)$-recognizable by a frequency automaton. We have shown a sufficient condition for it. We also have shown that for any

language $L \in LOGSPACE$ there exists k such that for any $n > k$ L can be $(n-k, n)$-recognized by a two-way frequency finite automaton.

Based on the p-adic numbers we have defined ultrametric query algorithms and ultrametric query complexity similar to probabilistic and quantum query complexity. However, the unrestricted ultrametric query model seems to be too powerful because such functions as OR_n and XOR_n can be computed with just 1 query and the complexity never exceeds the polynomial degree of the function. Therefore one should try to find a natural restriction for the model that for some functions gives more interesting query complexity bounds such as $O(\log n)$ and $O(\sqrt{n})$. One could also try to devise some lower bound technique and explore what functions are hard for ultrametric query algorithms.

We have introduced the notion of structured frequency algorithms by modifying the definition of frequency computation. Based on finite planes we have shown a structure of size $O(\sqrt{n})$ that allows only recognition of recursive sets and shown that using overlapping structures this size cannot be decreased. It is an open question whether there are some other (non-overlapping) structures of smaller size that allow only recognition of recursive sets.

Bibliography

[1] A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64, 2002.

[2] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[3] H. Austinat, V. Diekert, and U. Hertrampf. A structural property of regular frequency computations. *Theoretical Computer Science*, 292(1):33 – 43, 2003. Selected Papers in honor of Jean Berstel.

[4] H. Austinat, V. Diekert, U. Hertrampf, and H. Petersen. Regular frequency computations. In *Proc. RIMS Symposium on Algebraic Systems, Formal Languages and Computation*, volume 1166, pages 35–42, 2000.

[5] H. Austinat, V. Diekert, U. Hertrampf, and H. Petersen. Regular frequency computations. *Theoretical Computer Science*, 330(1):15 – 21, 2005. Insightful Theory.

[6] E. Bach and J. Shallit. *Algorithmic Number Theory*. MIT Press, Cambridge, MA, USA, 1996.

[7] K. Balodis, A. Beriņa, K. Cīpola, M. Dimitrijevs, J. Iraids, K. Jēriņš, V. Kacs, J. Kalējs, R. Krišlauks, K. Lukstiņš, R. Raumanis, I. Scegulnaja, N. Somova, A. Vanaga, and R. Freivalds. On the state complexity of ultrametric finite automata. In *SOFSEM 2013: Theory and Practice of Computer Science*, volume 2, pages 1–9, 2013.

[8] P. Berman and A. Lingas. *On complexity of regular languages in terms of finite automata*. Institute of Computer Science, Polish Academy of Sciences, 1977.

[9] M. D. Biasi and A. Yakaryilmaz. Non-trivial unary languages recognized by two-way one-counter machines. *CoRR*, abs/1311.0849, 2013.

[10] J.-C. Birget. Two-way automata and length-preserving homomorphisms. *Mathematical systems theory*, 29(3):191–226, 1996.

[11] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity. *Theoretical Computer Science 288(1)*, pages 21–43, 2002.

[12] M. Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47(0):149 – 158, 1986.

[13] A. N. Degtev. On (m, n)-computable sets. In D. I. Moldavanskij, editor, *Algebraic Systems*, pages 88–99. Ivanovo Gos. Universitet, 1981. In Russian.

[14] B. Dragovich and A. Dragovich. A p-adic model of DNA sequence and genetic code. *p-Adic Numbers, Ultrametric Analysis, and Applications*, pages 1(1):34–41, 2009.

[15] A. Drucker and R. de Wolf. Quantum proofs for classical theorems. *arXiv preprint arXiv:0910.3376*, 2009.

[16] C. Dwork and L. Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal on Computing*, 19(6):1011–1023, 1990.

[17] R. Freivalds. Language recognition using finite probabilistic multitape and multihead automata. *Problemy Peredachi Informatsii*, 15(3):99–106, 1979. (in Russian).

[18] R. Freivalds. Probabilistic two-way machines. In J. Gruska and M. Chytil, editors, *Mathematical Foundations of Computer Science 1981*, volume 118 of *Lecture Notes in Computer Science*, pages 33–45. Springer Berlin Heidelberg, 1981.

[19] R. Freivalds. On the growth of the number of states in result of the determinization of probabilistic finite automata. *Avtomatika i Vicislitelnaja Tehnika*, 3:39–42, 1982. (in Russian).

[20] R. Freivalds. Ultrametric automata and Turing machines. In A. Voronkov, editor, *Turing-100*, volume 10 of *EPiC Series*, pages 98–112. EasyChair, 2012.

[21] R. Freivalds. Ultrametric vs. quantum query algorithms. In A.-H. Dediu, M. Lozano, and C. Martn-Vide, editors, *Theory and Practice of Natural Computing*, volume 8890 of *Lecture Notes in Computer Science*, pages 1–10. Springer International Publishing, 2014.

[22] R. Freivalds, T. Zeugmann, and G. R. Pogosyan. On the size complexity of deterministic frequency automata. In *Language and Automata Theory and Applications*, pages 287–298. Springer, 2013.

[23] W. Gasarch and F. Stephan. A techniques oriented survey of bounded queries. *LONDON MATHEMATICAL SOCIETY LECTURE NOTE SERIES*, pages 117–156, 1999.

[24] V. Geffert. An alternating hierarchy for finite automata. *Theoretical Computer Science*, 445:1–24, Aug. 2012.

[25] A. G. Greenberg and A. Weiss. A lower bound for probabilistic algorithms for finite state machines. *J. Comput. Syst. Sci.*, 33(1):88–105, Aug. 1986.

[26] M. Hall, Jr. *Combinatorial Theory (2^{nd} Ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[27] M. Hinrichs and G. Wechsung. Time bounded frequency computations. In *Computational Complexity, 1997. Proceedings., Twelfth Annual IEEE Conference on (Formerly: Structure in Complexity Theory Conference)*, pages 185–192. IEEE, 1997.

[28] M. Holzer, M. Kutrib, and A. Malcher. Multi-head finite automata: Characterizations, concepts and open problems. *Electronic Proceedings in Theoretical Computer Science*, 1:93–107, Jun 2009.

[29] M. Holzer, M. Kutrib, and A. Malcher. Complexity of multi-head finite automata: Origins and directions. *Theoretical Computer Science*, 412(12):83 – 96, 2011. Complexity of Simple Programs.

[30] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[31] C. A. Kapoutsis. Size complexity of two-way finite automata. In *International Conference on Developments in Language Theory*, pages 47–66. Springer, 2009.

[32] C. A. Kapoutsis. Minicomplexity. In *International Workshop on Descriptional Complexity of Formal Systems*, pages 20–42. Springer, 2012.

[33] C. A. Kapoutsis, R. Královič, and T. Mömke. An exponential gap between LasVegas and deterministic sweeping finite automata. In *International Symposium on Stochastic Algorithms: Foundations and Applications*, pages 130–141. Springer, 2007.

[34] J. Kari and C. Moore. Rectangles and squares recognized by two-dimensional automata. In J. Karhumki, H. Maurer, G. Pun, and G. Rozenberg, editors, *Theory Is Forever*, volume 3113 of *Lecture Notes in Computer Science*, pages 134–144. Springer Berlin Heidelberg, 2004.

[35] J. Kari and V. Salo. A survey on picture-walking automata. In W. Kuich and G. Rahonis, editors, *Algebraic Foundations in Computer Science*, volume 7020 of *Lecture Notes in Computer Science*, pages 183–213. Springer Berlin Heidelberg, 2011.

[36] A. Khrennikov. Non-archimedean analysis. In *Non-Archimedean Analysis: Quantum Paradoxes, Dynamical Systems and Biological Models*, volume 427 of *Mathematics and Its Applications*, pages 101–129. Springer Netherlands, 1997.

[37] E. B. Kinber. Frequency computations in finite automata. *Cybernetics and Systems Analysis*, 12(2):179–187, 1976.

[38] D. König. Sur les correspondances multivoques des ensembles. *Fundamenta mathematicae*, 8(1):114–134, 1926.

[39] S. V. Kozyrev. Ultrametric analysis and interbasin kinetics. In *p-adic mathematical physics: 2nd International Conference*, volume 826, pages 121–128. AIP Publishing, 2006.

[40] R. Královič. *Complexity classes of finite automata*. PhD thesis, ETH Zürich, 2010.

[41] R. Krišlauks, I. Rukšāne, K. Balodis, I. Kucevalovs, R. Freivalds, and I. Nāgele. Ultrametric Turing machines with limited reversal complexity. In *SOFSEM 2013: Theory and Practice of Computer Science*, volume 2, pages 87–94, 2013.

[42] M. Kummer and F. Stephan. The power of frequency computation. In H. Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *Lecture Notes in Computer Science*, pages 323–332. Springer Berlin Heidelberg, 1995.

[43] O. Kupferman, A. Ta-Shma, and M. Y. Vardi. Counting with automata. Technical report, 1999.

[44] E. Leiss. Succinct representation of regular languages by boolean automata. *Theoretical Computer Science*, 13(3):323 – 330, 1981.

[45] I. Macarie. Multihead two-way probabilistic finite automata. In R. Baeza-Yates, E. Goles, and P. Poblete, editors, *LATIN '95: Theoretical Informatics*, volume 911 of *Lecture Notes in Computer Science*, pages 371–385. Springer Berlin Heidelberg, 1995.

[46] D. A. Madore. A first introduction to p-adic numbers. Online, 2000.

[47] R. McNaughton. The theory of automata, a survey. *Advances in Computers*, 2:379–421, 1961.

[48] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[49] B. Monien. Two-way multihead automata over a one-letter alphabet. *RAIRO - Theoretical Informatics and Applications - Informatique Thorique et Applications*, 14(1):67–82, 1980.

[50] N. Nisan and M. Szegedy. On the degree of boolean functions as real polynomials. *computational complexity*, 4(4):301–313, 1994.

[51] H. Petersen. Two-way one-counter automata accepting bounded languages. *SIGACT News*, 25(3):102–105, Sept. 1994.

[52] H. Petersen. Bounded counter languages. In M. Kutrib, N. Moreira, and R. Reis, editors, *Descriptional Complexity of Formal Systems*, volume 7386 of *Lecture Notes in Computer Science*, pages 266–279. Springer Berlin Heidelberg, 2012.

[53] M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230 – 245, 1963.

[54] G. F. Rose. An extended notion of computability. In *International Congress for Logic, Methodology and Philosophy of Science, Stanford, California*, 1960.

[55] W. J. Sakoda and M. Sipser. Nondeterminism and the size of two way finite automata. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 275–286, New York, NY, USA, 1978. ACM.

[56] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition, 2006.

[57] B. A. Trakhtenbrot. On the frequency computation of functions. *Algebra i Logika*, 2(1):25–32, 1963. In Russian.

[58] P. Turakainen. Generalized automata and stochastic languages. *Proceedings of The American Mathematical Society*, 21:303–309, 1969.

[59] I. V. V. V. S. Vladimirov and E. I. Zelenov. p-adic analysis and mathematical physics. *World Scientific*, 1995.

[60] A. Vasilieva and T. Mischenko-Slatenkova. High precision quantum query algorithm for computing and-based boolean functions. In *Proceedings of the 7th ACM International Conference on Computing Frontiers*, CF '10, pages 247–256, New York, NY, USA, 2010. ACM.

[61] A. Yakaryılmaz and C. Say. Succinctness of two-way probabilistic and quantum finite automata. *Discrete Mathematics & Theoretical Computer Science*, 12(4):19–40, 2010.

[62] A. C. Yao and R. L. Rivest. k + 1 heads are better than k. *J. ACM*, 25(2):337–340, Apr. 1978.