CLARKSON UNIVERSITY

Multiplicity Automata, Polynomials and the Complexity of Small-Depth Boolean Circuits

A Thesis
by
Erion Plaku

Department of Mathematics and Computer Science

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science,
Computer Science

April 23, 2002

Accepted by the Graduate School

Date Dean

CLARKSON UNIVERSITY
DEPARTMENT OF
MATHEMATICS AND COMPUTER SCIENCE

The undersigned have examined the thesis entitled "**Multiplicity Automata, Polynomials and the Complexity of Small-Depth Boolean Circuits**" presented by **Erion Plaku** a candidate for the degree of **Master of Science**, and hereby certify that it is worthy of acceptance.

Date: April 23, 2002

Advisor:

Alexis Maciel

Examining Committee:

Christino Tamon

Chris Lynch

Abstract

We are interested in the computational power of Boolean circuits. Our objective is to provide lower bounds for different circuit classes while emphasizing connections to learning and pseudorandomness.

The main contribution of this thesis is the development of a simple method based on multiplicity automata which we use to prove lower bounds on the size of some classes of circuits. We show $2^{\Omega(n)}$ lower bounds for $\mathrm{AC}^0_1 \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ and $\mathrm{AND} \circ \mathrm{MOD}^{\{0\}}_m \circ \mathrm{SYM}$ circuits and $2^{\Omega(n/\log n)}$ lower bounds for $\mathrm{AND} \circ \{\mathrm{MOD}^{\{0\}}_m, \mathrm{EXACT}\} \circ \mathrm{SYM}$ circuits. We also show $2^{\Omega(n)}$ lower bounds for $\mathrm{OR} \circ \mathrm{CC}^0_d[p^k] \circ \mathrm{SYM}_{\epsilon n}$ and $\mathrm{AND} \circ \mathrm{CC}^0_d[p^k] \circ \mathrm{SYM}_{\epsilon n}$ circuits computing the AND_n and OR_n functions, respectively, for any $\epsilon \leq p^{-4kd}$. All our lower bounds remain the same even if $\mathrm{CC}^0[p^k]$ circuits are replaced by $\{\mathrm{CC}^0[p_i^{k_i}] : p_i, k_i \in O(1)\}$ circuits.

We show how to compute a pseudorandom function generator secure against polynomial-time adversaries by $\mathrm{q(SUM} \circ \mathrm{EXACT)}^+$ and $\mathrm{q(OR} \circ \mathrm{EXACT)}^+$ circuits. Consequently, these circuit classes are not weakly learnable in polynomial time even when membership queries are allowed

Other contributions include the use of polynomials to obtain lower bounds on circuits computing the square-free function. We show that the square-free function cannot be computed by depth 2 circuits with $\mathrm{AND}_{\lfloor 0.14 \ln n \rfloor}$ gates at the input and a modular $\mathrm{MOD}^{\{0\}}_{p^k}$ or a weighted threshold gate at the output. We remark that p, k and the weights are not bounded above by any function and can in fact be exponential. These results provide non-trivial lower bounds on the complexity of a number theoretic problem which is closely related to the integer factorization problem.

Acknowledgements

I would like to thank Alexis Maciel, my advisor, for introducing me to the area of Boolean circuits. Without his encouragements and constant support this thesis would have not been possible. I enjoyed working with him.

I am thankful to Christino Tamon for the many helpful discussions on multiplicity automata and learning and his prompt answers to a myriad of questions.

I would also like to thank Chris Lynch for a careful review of this thesis and his valuable comments.

I would like to acknowledge the financial support of NSF through grant CCR-9877150.

I thank everyone who has helped me in the research process and especially my friends at the churches in Berat and Koinonia for their prayers and encouragement.

Potsdam, New York — Erion Plaku
April 6, 2002

Table of Contents

Chapter 1

Introduction

The goal of *complexity theory* is to characterize the amount of resources needed for the computation of specific functions. Common resources include sequential time, sequential space, number of gates in Boolean circuits, parallel time in a multiprocessor machine, etc. The exact complexity of a function is determined by the amount of resources that is both sufficient and necessary for its computation. Sufficiency implies an upper bound on the amount of resources needed to compute the function for every instance of the input. Necessity implies a lower bound, that is, for some instance of the input, at least a certain amount of resources is required to compute the function.

The amount of resources that is needed to compute a function allows for an elegant classification of functions according to their computational complexity. Researchers have developed the notion of *complexity classes*, where a complexity class is defined by specifying (a) the type of computation model M (b) the resource R which we wish to measure in this model, and (c) an upper bound U on this resource. A complexity class, then, consists of all functions requiring at most an amount U of resource R for their computation in the model M. Thus, to find the complexity of a function we determine to which complexity classes it belongs (by providing upper bounds on the resource) and to which complexity classes it does not belong (by providing lower bounds).

Naturally, we say that a function is easy if the upper bound on the amount of resources needed to compute it is small, and we say that the function is hard if the lower bound is large. For instance, functions belonging to P are said to be easy and NP-complete functions are said to be hard, although not provably so.

The most fundamental question in complexity theory is whether P is different from NP. After many years of extensive research the question remains unanswered, although most researchers believe that P $\neq$ NP. No function that can be computed in nondeterministic polynomial time is known to require more than deterministic polynomial time. In fact, no nonlinear lower bounds have been proven on general models of computation for any functions in NP.

This lack of progress has led researchers to consider restricted models of computation with the hope that these restricted models would enable them to constrain the problem and develop methods to derive strong lower bounds. In turn, these methods would provide a better understanding of the model of computation, and, by gradually removing the restrictions, nonlinear lower bounds would be proven for the general model of computation.

In this thesis we will consider the *Boolean circuit* model of computation. A Boolean circuit is a directed acyclic graph whose nodes compute certain Boolean functions from some basis of computation Ω. A Boolean function $f : \{0,1\}^* \to \{0,1\}$ is computed by a sequence of circuits $\{C_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$, circuit C_n computes the function f restricted to inputs of length n. Boolean circuits are a general model of computation, since non-uniform Boolean circuits can compute any functions. Furthermore, a Turing machine running in time $T(n)$ can be simulated by circuits of size $O\left(T(n)\log T(n)\right)$. Consequently, strong lower bounds on the size of Boolean circuits computing a function f imply strong lower bounds on the computational time of f on a Turing machine. Unlike Turing machines, Boolean circuits are easy to define and their fixed structure is more amendable to mathematical analysis. It remains an open problem, however, to derive nonlinear lower bounds on the size of general circuits computing some explicit Boolean function. Another attractive feature of the Boolean circuit model of computation is that it can be easily restricted, and it is in these restricted models that significant progress has been made.

One such restricted model consists of *monotone* circuits. A monotone circuit is a Boolean circuit whose gates compute monotone functions. Razborov [40] was the first to obtain superpolynomial lower bounds of $n^{\Omega(\log n)}$ on the size of monotone circuits computing the clique and perfect-matching functions on a graph of n vertices. Subsequent improvements of Razborov's methods by Alon & Boppana [3] and Andreev [4] resulted in exponential lower bounds for the monotone complexity of the clique function and other functions in NP.

Another way to restrict the Boolean circuit model of computation is to bound the depth of the circuit either by a constant or a slowly growing function of the input length. The first breakthrough result came when Furst, Saxe, and Sipser [21] showed superpolynomial lower bounds on the size of AC^0 circuits (polynomial-size constant-depth circuits with AND, OR, and NOT gates) computing the parity function. Yao [50] showed that a deeper analysis of the method of Furst, Saxe, and Sipser would yield an exponential lower bound. Subsequently, Håstad [26] simplified the proof and obtained near optimal lower bounds. This series of results led researchers to consider an interesting question: Since parity is so hard to compute, what happens if in addition to AND, OR, and NOT gates we allow parity gates, or in general modular $\mathrm{MOD}_m^{\{0\}}$ gates, where a modular gate outputs 1 iff modulo m, the sum of its inputs is zero? This question was partially answered by Smolensky [45], who showed that polynomial-size constant-depth circuits with AND, OR, NOT, and $\mathrm{MOD}_p^{\{0\}}$ gates cannot compute the $\mathrm{MOD}_q^{\{0\}}$ function if p and q are distinct primes. As a consequence, many researchers considered bounded-depth circuits with different types of gates such as threshold, exact, symmetric, etc. Håstad & Goldmann [27] showed an exponential lower bound for

depth 3 threshold circuits with AND gates of fan-in $\frac{1}{3}\log n$ at the input level. Razborov & Wigderson [42] were able to remove the restriction on the fan-in of the AND gates, but their lower bound is only quasipolynomial. Despite the many efforts, to this day, no superpolynomial lower bounds are known for depth 3 threshold circuits, not even for depth 2 circuits consisting entirely of modular gates MOD_m^A, where the modulus m is not a prime power. There are two approaches to explaining the lack of lower bounds for these small-depth classes of circuits: (1) although seemingly restricted, these classes of circuits may be quite powerful and might indeed compute all of NP, or (2) the current lower bound methods are not powerful enough.

Few researchers believe that these small-depth circuit classes might be powerful enough to compute all of NP. Motivated by the second approach, Razborov & Rudich [41] studied the current lower bound proofs and were able to categorize them as *natural proofs*. Furthermore, they were able to show that if a circuit class contains a pseudorandom function generator secure against strong statistical tests, then there are no natural proofs for separating this class from P/poly, the class of unbounded-depth polynomial-size circuits over a complete basis. The containment of pseudorandom function generators in a circuit class is also evidence that the circuit class cannot be learned. These results provide interesting connections amongst the fields of circuit complexity, computational learning theory, and cryptography. Let $\mathcal{C}$ be a circuit class. First, a natural lower bound proof on the size of the circuits from the class $\mathcal{C}$ implies that the class $\mathcal{C}$ does not contain pseudorandom function generators secure against statistical tests. Therefore, the class $\mathcal{C}$ may be learnable. Second, a learning algorithm for the class $\mathcal{C}$ implies that $\mathcal{C}$ does not contain pseudorandom function generators. Consequently, it may be possible to find a natural lower bound proof on the size of the circuits from $\mathcal{C}$. In trying to separate different circuit classes the following three tasks are related: (1) show lower bounds for new circuit classes; (2) develop learning algorithms for new circuit classes;(3) show that certain circuit classes contain pseudorandom function generators.

1.1 Content and Results of the Thesis

In this thesis we are interested in the computational power of Boolean circuits. Our objective is to provide lower bounds for different circuit classes while emphasizing connections to learning and pseudorandomness.

The main contribution of this thesis is the development of a simple method based on *multiplicity automata* which we use to prove lower bounds for some circuit classes. We also show how to compute pseudorandom function generators by some closely related circuit classes.

We show how to use multiplicity automata to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits, *i.e.,* polynomial-size constant-depth circuits with $\mathrm{MOD}_{p^k}^A$ gates anywhere and symmetric gates at the input level. Our simulation is carried out in steps. First, we show how to simulate symmetric gates whose inputs are the Boolean variables $x_1, \ldots, x_n$, their negations, and the

constants 0 or 1. We then show how to simulate $\mathrm{MOD}_{p^k}^A$ gates whose inputs are Boolean functions of $x_1, \ldots, x_n$. Finally, by repeatedly applying the simulation of the modular gates we can simulate $\mathrm{CC}^0[p^k]$ circuits whose inputs are symmetric gates.

Using modular restriction techniques, Grolmusz & Tardos [24] characterized the functions computable by $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A$ circuits. Using the Degree-Decreasing Lemma and random restriction, they also showed a superpolynomial lower bound for $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A \circ \mathrm{AND}$ circuits computing the AND function of n Boolean variables, where the fan-in of the AND gates is $O(1)$ and the fan-in of the MOD_m^A gates is $o(n^2/\log n)$. Using communication matrices, Krause & Waack [30] showed an exponential lower bound for $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits computing the sequence identity function. Note that $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A \subseteq \mathrm{CC}^0[p^k] \circ \mathrm{SYM} \subseteq \mathrm{MOD}_p^{\{0\}} \circ \mathrm{SYM}$.

Based on the simulation of $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits by multiplicity automata, we simplify the complex technique used to prove the lower bound for $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits and prove exponential lower bounds for some other simple functions. We derive lower bounds for other circuit classes by exploiting their similarities to $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits. Using the properties of OR and AND gates, we show exponential lower bounds for $\mathrm{OR} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ and $\mathrm{AND} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits computing the sequence identity function and its complement respectively. By using the prime factorization of m to express the $\mathrm{MOD}_m^{\{0\}}$ gate as an AND of modular gates with prime power moduli, we obtain exponential lower bounds for $\mathrm{AND} \circ \mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits. Allowing the primes to grow logarithmically yields a lower bound of $2^{\Omega(n/\log n)}$ for $\mathrm{AND} \circ \{\mathrm{MOD}_m^{\{0\}}, \mathrm{EXACT}\} \circ \mathrm{SYM}$ circuits. We also show exponential lower bounds for $\mathrm{OR} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ and $\mathrm{AND} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ circuits computing the AND_n and OR_n functions, respectively, for any $\epsilon \leq p^{-4kd}$.

All our lower bounds remain the same even if $\mathrm{CC}^0[p^k]$ circuits are replaced by $\{\mathrm{CC}^0[p_i^{k_i}] : p_i, k_i \in O(1)\}$ circuits.

We modify the pseudorandom function generator (PRFG) of Naor & Reingold [37] while still preserving its security against polynomial-time adversaries. We then show how to compute this PRFG by $\mathrm{q(SUM} \circ \mathrm{EXACT)}^+$ circuits, *i.e.,* quasipolynomial-size circuits with a SUM gate at the output, EXACT gates at the second level and AND gates of polylogarithmic fan-in at the input level. From our proof it is clear that the output gate can also be an OR gate, and thus, the modified PRFG can be computed by $\mathrm{q(OR} \circ \mathrm{EXACT)}^+$ circuits. Consequently, these circuit classes are not weakly learnable in polynomial time even when membership queries are allowed. The original PRFG of [37] is computable by TC_5^0 circuits and previous to our work TC_4^0 circuits were the simplest circuit class containing a PRFG secure against polynomial-time adversaries (see [28]).

Other contributions include the use of polynomials to obtain lower bounds on circuits computing the square-free function. We show that the square-free function cannot be computed by depth 2 circuits with $\mathrm{AND}_{\lfloor 0.14 \ln n \rfloor}$ gates at the input and a modular $\mathrm{MOD}_{p^k}^{\{0\}}$ or a weighted threshold gate at the output. We remark that p, k and the weights are not

bounded above by any function and can in fact be exponential. These results provide non-trivial lower bounds on the complexity of a number theoretic problem which is closely related to the integer factorization problem.

Table 1.1 summarizes of the results obtained in this thesis.

1.2 Outline of the Thesis

In Chapter 2 we study the Boolean circuit model of computation and the properties of the different operations which are used as the basis of computation. We then present strong lower bounds for some simple circuit classes. In Chapter 3 we use polynomials to prove lower bounds on the complexity of depth 2 circuits with AND gates at the input level and a modular or a threshold gate at the output. We use random restriction techniques (as in [23]) to obtain lower bounds on the size of $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{AND}$ circuits. We also study the relation between $\mathrm{MOD}_m^A \circ \mathrm{AND}$ circuits and the threshold of representing polynomials. We provide trade-off results between the fan-in of the AND gates and the threshold. Finally, we prove several interesting results on the complexity of the square-free function by using a number-theoretic sieve method similar to that of [13, 14, 15]. In Chapter 4 we focus on the multiplicity automata model of computation. We prove several important properties of multiplicity automata. Then, using these properties, we describe how to construct multiplicity automata to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits. We then derive lower bounds for several circuit classes. In Chapter 5, based on a modification of the work of Naor & Reingold [37], we show how to compute a pseudorandom function generator by some simple circuit classes of quasipolynomial size and discuss the consequences of such constructions.

The results on the complexity of the square-free function appeared in [39] and are based on joint work with Igor E. Shparlinski. The multiplicity automata method (Chapter 4) and the hardness results (Chapter 5) are based on joint work with Alexis Maciel [33, 34].

1.3 Notation and Basic Definitions

In this section we provide some of the notation and basic definitions we use throughout the thesis.

Common Symbols Unless otherwise specified, p denotes a fixed prime, m denotes a fixed composite integer, and k denotes a fixed integer.

Sets $\mathbb{N}$ denotes the set of natural numbers $\{1, 2, 3, \ldots\}$, $\mathbb{Z}$ denotes the set of integers $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$, and $\mathbb{R}$ denotes the set of real numbers. For integers a, b with $a \leq b$, $[a, b]$ denotes the set $\{a, a+1, \ldots, b-1, b\}$. If A is a set, then $\mathcal{P}(A)$ denotes the power set of A, *i.e.,* $\mathcal{P}(A) = \{B : B \subseteq A\}$, and A^+, A^- denote the set of positive and negative

- Lower Bounds: Multiplicity Automata Method
  - A $2^{\Omega(n)}$ lower bound for $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits computing some simple functions in $\{\mathrm{AND}_n, \mathrm{OR}_n, \mathrm{MOD}_2^{\{0\}}\} \circ \{\mathrm{AND}_2, \mathrm{OR}_2\}$.
  - A $2^{\Omega(n)}$ lower bound for $\mathrm{OR} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ and $\mathrm{AND} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits computing the sequence identity function and its complement, respectively.
  - A $2^{\Omega(n)}$ lower bound for $\mathrm{AND} \circ \mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits computing the complement of the sequence identity function.
  - A $2^{\Omega(n/\log n)}$ lower bound for $\mathrm{AND} \circ \{\mathrm{MOD}_m^{\{0\}}, \mathrm{EXACT}\} \circ \mathrm{SYM}$ circuits computing the complement of the sequence identity function.
  - A $2^{\Omega(n)}$ lower bound for $\mathrm{OR} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ and $\mathrm{AND} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ circuits computing the AND_n and OR_n functions, respectively, for any $\epsilon \leq p^{-4kd}$.
- Lower Bounds: Polynomial Method
  - The square-free function is not computable by depth 2 circuits with
    - $\mathrm{AND}_{0.14 \ln n}$ gates at the input and
    - a $\mathrm{MOD}_{p^k}^{\{0\}}$ or a weighted threshold gate at the output,

    where p, k and the weights are not bounded above by any function and can in fact be exponential.
- Cryptographic Results
  - If factoring is 2^{n^δ}-hard, then the circuit classes $\mathrm{q}(\mathrm{SUM} \circ \mathrm{EXACT})^+$ and $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ contain a pseudorandom function generator secure against any polynomial-time adversary.
- Learning Results
  - If factoring is 2^{n^δ}-hard, then the circuit classes $\mathrm{q}(\mathrm{SUM} \circ \mathrm{EXACT})^+$ and $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ are not weakly learnable in polynomial time even when membership queries are allowed.

Table 1.1: Summary of the results obtained in this thesis. In this table, p is prime, m and k are integers and δ, $0 < \delta < 1$, is a real number. Unless otherwise specified, they are all constants. The subscript of the symmetric gates indicates their fan-in. The lower bounds remain the same even if $\mathrm{CC}^0[p^k]$ circuits are replaced by $\{\mathrm{CC}^0[p_i^{k_i}] : p_i, k_i \in O(1)\}$ circuits.

elements of A, respectively, *i.e.,* $A^+ = \{a : a \in A \text{ and } a > 0\}$, $A^- = \{a : a \in A \text{ and } a < 0\}$. We write $A \subset B$ to indicate that A is a proper subset of B. We write $A - B$ to denote the set of elements of A that are not in B, *i.e.,* $A - B = \{x : x \in A \wedge x \notin B\}$.

Vectors and Strings We use the terms binary vector of dimension n and binary string of length n interchangeably. We use bold lowercase letters to denote vectors. If $\mathbf{x}$ is a vector of dimension n, then, for $1 \leq i \leq n$, both $\mathbf{x}_i$ and $[\mathbf{x}]_i$ denote the element of $\mathbf{x}$ in the ith position. For a set $A \subseteq [1, n]$, $\mathbf{x}_A$ denotes a binary vector of dimension n such that for $1 \leq i \leq n$, $[\mathbf{x}_A]_i = 1$ if $i \in A$ and $[\mathbf{x}_A]_i = 0$ if $i \notin A$. We use $\odot$ to denote the inner product modulo 2 of two vectors $\mathbf{x}$ and $\mathbf{y}$ of the same dimension, *i.e.,* $\mathbf{x} \odot \mathbf{y} = \sum_i x_i y_i \bmod 2$, and $\circ$ to denote string concatenation.

Matrices We use uppercase or Greek letters to denote matrices. Let M be an $m \times n$ matrix, where $m, n \in \mathbb{N}$. The indexing of rows and columns starts from 1. For $1 \leq i \leq m$ and $1 \leq j \leq n$, $M_{i,*}$ denotes the ith row of M, $M_{*,j}$ denotes the jth column of M, and $M_{i,j}$ denotes the element of matrix M in the ith row and jth column. I_n denotes the $n \times n$ identity matrix.

Predicates For any predicate f, define $[f] = 1$ if f is *true*, and $[f] = 0$ if f is *false*. Thus, if a is an integer greater than 7, then $[a \in \mathbb{Z}] + [a > 7] = 2$.

Logarithms and Exponentials We denote by $\log x$ the binary logarithm of x, by $\ln x$ the natural logarithm of x, and by $\exp(x)$ the expression e^x.

Function Complexity Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}^+$. Then,

— $f(n) \in O(g(n))$ if $\exists c \in \mathbb{N}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) \leq cg(n)$.

— $f(n) \in o(g(n))$ if $\forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : cf(n) < g(n)$.

— $f(n) \in \Omega(g(n))$ if $\exists c \in \mathbb{N}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : cf(n) \geq g(n)$.

— $f(n) \in \omega(g(n))$ if $\forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) > cg(n)$.

— $f(n) \in \theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

	Upper Bound	Lower Bound
f is polynomial	$\exists c \in \mathbb{R}^+ : f(n) \in O(n^c)$	$\exists c \in \mathbb{R}^+ : f(n) \in \Omega(n^c)$
f is quasipolynomial	$\exists c \in \mathbb{R}^+ : f(n) \in O\left(n^{\log^c n}\right)$	$\exists c \in \mathbb{R}^+ : f(n) \in \Omega\left(n^{\log^c n}\right)$
f is polylogarithmic	$\exists c \in \mathbb{R}^+ : f(n) \in O(\log^c n)$	$\exists c \in \mathbb{R}^+ : f(n) \in \Omega(\log^c n)$
f is exponential	$\exists c \in \mathbb{R}^+ : f(n) \in O\left(2^{n^c}\right)$	$\exists c \in \mathbb{R}^+ : f(n) \in \Omega\left(2^{n^c}\right)$

Restriction A restriction ρ is a mapping of the Boolean variables $x_1, \ldots, x_n$ to the set $\{0, 1, *\}$, where

- $\rho(x_i) = 0$ means that x_i is substituted by 0;
- $\rho(x_i) = 1$ means that x_i is substituted by 1;
- $\rho(x_i) = *$ means that x_i remains a variable.

Random Restriction A random restriction ρ with parameters p, q is a mapping of the Boolean variables $x_1, \ldots, x_n$ to the set $\{0, 1, *\}$, where $0 < p, q < 1$, $p + q < 1$, and for each x_i independently,

- $\rho(x_i) = 0$ with probability p;
- $\rho(x_i) = 1$ with probability q;
- $\rho(x_i) = *$ with probability $1 - p - q$.

Threshold Boolean Functions A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a *threshold* if and only if there exists an integer θ such that

$$f(x_1, \ldots, x_n) = 1 \iff x_1 + \cdots + x_n \geq \theta.$$

Modular Boolean Functions A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *modular* if and only if there exists an integer m and a set $A \subseteq \{0, 1, \ldots, n\}$ such that

$$f(x_1, \ldots, x_n) = 1 \iff (x_1 + \cdots + x_n) \bmod m \in A.$$

Symmetric Boolean Functions A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *symmetric* if and only if for every permutation π of the set $\{1, \ldots, n\}$,

$$f(x_1, \ldots, x_n) = f\left(x_{\pi(1)}, \ldots, x_{\pi(n)}\right).$$

Thus, the output of a symmetric function depends only on the sum of its inputs. Consequently, a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is symmetric if and only if there exists a set $A \subseteq \{0, 1, \ldots, n\}$ such that

$$f(x_1, \ldots, x_n) = 1 \iff (x_1 + \cdots + x_n) \in A.$$

The Set of Boolean Functions We denote the set of all Boolean functions on n variables and outputs of length m by $\mathcal{B}_{n,m}$, *i.e.,*

$$\mathcal{B}_{n,m} = \{f \mid f : \{0,1\}^n \to \{0,1\}^m\}.$$

We also let $\mathcal{B}_n = \mathcal{B}_{n,1}$.

Chapter 2

Boolean Circuits

The goal of circuit complexity is to determine the minimum number of operations from the basis Ω of computation that are needed to compute some explicit Boolean function $f : \{0,1\}^n \to \{0,1\}^m$. In this chapter we give the necessary background on the Boolean circuit model of computation. We study the properties of the different operations which are used as the basis of computation Ω. We then present strong lower bounds for some simple circuit classes.

2.1 Model of Computation

Basis of Computation The basis of computation, denoted by Ω, is a collection of functions from the set $\{0,1\}^*$ to the set $\{0,1\}$. A basis Ω is called complete if every Boolean function can be expressed as a combination of functions from Ω. The canonical basis $\{\text{AND}, \text{OR}, \text{NOT}\}$ is complete, while the monotone basis $\{\text{AND}, \text{OR}\}$ is not.

Structure of Boolean Circuits A Boolean circuit $C_{n,m}$ over the basis of computation Ω with input $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ and output $\mathbf{y} = \langle y_1, \ldots, y_m \rangle \in \{0,1\}^m$ is a *directed acyclic graph* whose nodes are divided into *inputs*, *gates*, and *outputs*. Input nodes are labeled with a variable x_i, its negation $\neg x_i$ or a constant 0 or 1 and have in-degree 0. Gates are labeled with Boolean operations from the basis set Ω and have positive in-degree. The in-degree of a node is referred to as the fan-in, and the out-degree as the fan-out. A node p is a predecessor of some gate g_i if there is a directed edge from p to g_i. Each gate g_i is defined by its type $\omega_i \in \Omega$ and a k_i-tuple $(p_1, \ldots, p_{k_i})$ of predecessor nodes. The function out_{g_i} computed by gate g_i is defined inductively: If $g_i = (\omega_i, p_1, \ldots, p_{k_i})$, then

$$\text{out}_{g_i}(\mathbf{x}) = \omega_i \left(\text{out}_{p_1}(\mathbf{x}), \ldots, \text{out}_{p_{k_i}}(\mathbf{x}) \right).$$

Output nodes can be inputs or gates and have out-degree 0. The function $f_{n,m} : \{0,1\}^n \to \{0,1\}^m$ computed by the circuit $C_{n,m}$, where $f_{n,m} = \langle f_1, \ldots, f_m \rangle$, is determined by choosing m nodes $\langle p_1, \ldots, p_m \rangle$ as outputs and letting f_i be the function computed at node p_i.

Families of Circuits In this thesis, we consider circuits with n inputs and a single output, which we denote by C_n. As such, the circuit model is very restricted for it can handle only inputs of a fixed length. To compute functions of arbitrary input length we consider *families of circuits*, that is, a sequence $C_1, C_2, C_3, \ldots$ of circuits, one for each input length. Thus, a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ computes a function $f : \{0,1\}^* \to \{0,1\}$ if for every $n \in \mathbb{N}$, circuit C_n computes the function f_n, where f_n denotes the function f restricted to inputs of length n.

Generating a Circuit Description Given a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ that computes a certain function f, a natural question to consider is the following:"What is the amount of resources required to generate a detailed description of C_n, once the length n of the input is known?". Imposing an upper bound on this amount of resources allows for a *uniformity condition* on the circuits of a family. We say that a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ is *uniform* if and only if there exists an algorithm that, on input n, generates a description of C_n using no more than a certain amount of resources. Specific uniformity conditions are obtained by appropriately limiting the amount of resources allowed for generating circuit descriptions.

A family of circuits $\{C_n\}_{n\in\mathbb{N}}$ is *nonuniform* if the amount of resources that can be used to generate a circuit description is unlimited, that is, no assumption is made about the difficulty of constructing C_n, once the length n of the input is known.

We consider only nonuniform families of circuits, and thus, our lower bounds will be as strong as possible.

Complexity Measures The main complexity measures for a circuit are its *size* and *depth*. The size of a circuit is the number of its wires, *i.e.* the number of edges of the directed acyclic graph. The *size complexity* of a Boolean function $f_n : \{0,1\}^n \to \{0,1\}$ with respect to a basis Ω equals the smallest size of a circuit over Ω computing f_n. The depth of a circuit is the length of the longest directed path from an input to an output of the circuit. The *depth complexity* of a Boolean function f_n with respect to a basis Ω equals the smallest depth of a circuit over Ω computing f_n.

The size and depth complexities can be extended to families of circuits. Let $f : \{0,1\}^* \to \{0,1\}$, $s : \mathbb{N} \to \mathbb{N}$, and $d : \mathbb{N} \to \mathbb{N}$. The size complexity of f with respect to a basis Ω is s if and only if, for every $n \in \mathbb{N}$, the size complexity of f_n is $s(n)$, where f_n is f restricted to inputs of length n. Similarly, the depth complexity of f is d if and only if, for every $n \in \mathbb{N}$, the depth complexity of f_n is $d(n)$.

Boolean circuits can be used for sequential or parallel computations – we consider only parallel computations. For sequential computations the size of the circuit corresponds to the computation time. For parallel computations the size of the circuit corresponds to the number of processors, and the depth corresponds to the computation time.

Common Gates In this thesis we consider not only canonical Boolean gates, *i.e.,* AND, OR, NOT, but also many other gates computing interesting Boolean functions.
A modular gate MOD_m^A, where $m \in \mathbb{N}$ and $A \subseteq \{0, \ldots, m-1\}$, is defined as

$$\mathrm{MOD}_m^A(x_1, \ldots, x_n) = 1 \iff (x_1 + \cdots + x_n) \bmod m \in A.$$

A symmetric gate SYM^A, where $A \subseteq \{0, \ldots, n\}$, is defined as

$$\mathrm{SYM}^A(x_1, \ldots, x_n) = 1 \iff (x_1 + \cdots + x_n) \in A.$$

An exact gate EXACT_k, where $k \in \{0, \ldots, n\}$, is defined as

$$\mathrm{EXACT}_k(x_1, \ldots, x_n) = 1 \iff x_1 + \cdots + x_n = k.$$

A threshold gate $\mathrm{TH}_{\geq\theta}$, where $\theta \in \mathbb{N}$ is called the threshold, is defined as

$$\mathrm{TH}_{\geq\theta}(x_1, \ldots, x_n) = 1 \iff x_1 + \cdots + x_n \geq \theta.$$

Similarly, a threshold gate $\mathrm{TH}_{\leq\theta}$ is defined as

$$\mathrm{TH}_{\leq\theta}(x_1, \ldots, x_n) = 1 \iff x_1 + \cdots + x_n \leq \theta.$$

Some Complexity Classes for Small-Depth Circuits The class $\Omega\text{-}\mathcal{C}^k$ is defined as the class of Boolean function families $\{f_n\}_{n\in\mathbb{N}}$ that can be computed by a sequence of circuits $\{C_n\}_{n\in\mathbb{N}}$ over the basis Ω such that, for every $n \in \mathbb{N}$, the size of C_n is polynomial in n and the depth of C_n is $O\left(\log^k n\right)$. A subscript denotes the exact depth of the circuits, *i.e.,* $\Omega\text{-}\mathcal{C}_3^0$ circuits have depth 3.

The class NC^k is defined as the class of Boolean function families $\{f_n\}_{n\in\mathbb{N}}$ that can be computed by a sequence of constant fan-in Boolean circuits $\{C_n\}_{n\in\mathbb{N}}$ over the basis $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$ such that, for every $n \in \mathbb{N}$, the size of C_n is polynomial in n and the depth of C_n is $O\left(\log^k n\right)$. We also define

$$\mathrm{NC} = \bigcup_{k\in\mathbb{N}} \mathrm{NC}^k.$$

Since the notion of depth in circuits corresponds to computation time in parallel models, the class NC, first defined by Nick Pippenger[1] in 1979, can be described as the collection of functions efficiently computable in parallel with a reasonable number of processors.

Similarly, the class AC^k is defined as the class of Boolean function families $\{f_n\}_{n\in\mathbb{N}}$ that can be computed by a sequence of *unbounded* fan-in Boolean circuits $\{C_n\}_{n\in\mathbb{N}}$ over the basis

[1] NC is the abbreviation Cook made up for "Nick's class".

$\{\text{AND}, \text{OR}, \text{NOT}\}$ such that, for every $n \in \mathbb{N}$, the size of C_n is polynomial in n and the depth of C_n is $O\left(\log^k n\right)$. We also define

$$\text{AC} = \bigcup_{k \in \mathbb{N}} \text{AC}^k.$$

We define Σ_d as the subclass of AC^0_d circuits with an OR gate at the output and Π_d as the subclass of AC^0_d circuits with an AND gate at the output. We also define Δ_d as the class of Boolean functions in $\Sigma_d \cap \Pi_d$.

Other interesting complexity classes of small depth circuits are defined in a similar way. Thus, $\text{ACC}^k[m]$ is similar to AC^k, but in addition to AND, OR, NOT gates, MOD^A_m gates are allowed. $\text{CC}^k[m]$ is a subclass of $\text{ACC}^k[m]$ that allows AND, OR gates of constant fan-in only. Also, $\text{ACC}^k = \bigcup_{m \in \mathbb{N}} \text{ACC}^k[m]$ and $\text{ACC} = \bigcup_{k \in \mathbb{N}} \text{ACC}^k$. CC^k and CC are defined similarly. TC^k is the class of polynomial-size and depth-$O\left(\log^k n\right)$ circuits with threshold gates.

For convenience, we let a type G gate denote the class of polynomial-size circuits consisting of a single level of gates of type G. Thus, SYM denotes the class of polynomial-size circuits consisting of a single level of symmetric gates and EXACT denotes the class of polynomial-size circuits consisting of a single level of exact gates. The class POL is defined as the class of Boolean function families $\{f_n\}_{n \in \mathbb{N}}$ that can be computed by a sequence of polynomials $\{P_n\}_{n \in \mathbb{N}}$ with integer coefficients such that, for every $n \in \mathbb{N}$, the number of terms of P_n and the absolute values of each coefficient are bounded by a polynomial in n, and the degree of P_n is bounded by a constant. The class SUM is the subclass of POL corresponding to polynomials of degree one.

We will often use the same notation reserved for polynomial-size circuit classes to express circuit classes of other sizes. When we do so, the size of the circuit class will be mentioned explicitly. For example, we may write that AC^0 circuits require exponential size to compute the parity function instead of writing that constant-depth circuits with AND, OR and NOT gates require exponential size to compute the parity function.

In many cases polynomial-size circuit families $\{C_n\}_{n \in \mathbb{N}}$ from a certain circuit class $\mathcal{C}$ cannot compute a family of functions $\{f_n\}_{n \in \mathbb{N}}$. Following standard notation, we define $q\mathcal{C}$ as the class of Boolean function families $\{f_n\}_{n \in \mathbb{N}}$ that can be computed by a sequence of circuits $\{C_n\}_{n \in \mathbb{N}}$ such that, for every $n \in \mathbb{N}$, $C_n \in \mathcal{C}$, but the size of C_n is quasipolynomial in n. Thus, $q\text{AC}^0$ is the class of constant-depth and quasipolynomial-size circuits over the canonical basis, and $q\text{TC}^0$ is the class of constant-depth and quasipolynomial-size threshold circuits.

Composition of Circuit Complexity Classes Maciel & Thérien [35] developed a convenient notation to describe classes of circuits whose levels consist of various types of gates. Their notation is similar to that used in Calculus to denote composition of functions. Thus, if Γ and Λ are circuit classes, then $\Gamma \circ \Lambda$ circuits denote the class of functions f of the form

$f(x) = g(h(x))$, where $g \in \Gamma$ and $h \in \Lambda$. For example, $\mathrm{SYM} \circ \{\mathrm{AND}, \mathrm{MOD}\} \circ \mathrm{TC}^0_1$ is the class of functions computed by polynomial-size circuits with symmetric gates at the output, a mixture of AND and modular gates at level two, and threshold gates at the input level.

For convenience, Γ^+ denotes the composition of Γ with AND gates of polylogarithmic fan-in.

2.2 Modular Gates

As the name suggests, a modular gate counts its inputs modulo an integer. Since the gate computes a Boolean function, the result must be mapped to a Boolean value. Smolensky [45] defined the output of a modular gate MOD_m to be 1 if and only if the sum of the inputs was congruent to 0 modulo m. Other researchers have defined the output of a modular gate MOD_m to be 1 if and only if the sum of the inputs is not congruent to 0 modulo m. In this thesis, we use a more general definition: the output of a modular gate MOD_m is 1 if and only if the sum of the inputs modulo m is in some set $A \subseteq [0, m-1]$. We denote these general modular gates by MOD^A_m, and when $A = \{0\}$, we refer to the gate $\mathrm{MOD}^{\{0\}}_m$ as a special or ordinary modular gate. It is an open problem to determine whether circuits with general modular gates are stronger than circuits with ordinary modular gates. Another challenging problem is to determine whether counting modulo 6 is more powerful than counting modulo some prime power. Intuitively, counting modulo a prime power is the same as counting modulo a composite integer which is not a prime power; however, in some circuit settings they seem to be different. If the moduli are prime powers, exponential lower bounds are known for constant-depth circuits over the basis $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathrm{MOD}^A_{p^k}\}$, but if the moduli are not prime powers, no superpolynomial lower bounds are known even for depth-2 circuits over the basis $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathrm{MOD}^A_6\}$. The machinery of lower bound proofs for circuits with modular gates where the moduli are prime powers is based on Fermat's Little Theorem, which cannot be applied to composite moduli. In this section we study the properties of the modular gates and show that a function computed by a general modular gate MOD^A_m cannot be computed by any ordinary modular gate $\mathrm{MOD}^{\{0\}}_m$.

2.2.1 Properties of Modular Gates with Prime Power Moduli

The following lemma, which seems to be folklore, has been used in [45, 9, 24] and many other papers. We include a standard proof of this lemma since we will use a slightly different version of it in Chapter 4.

Lemma 2.2.1 (Folklore)**.** *Let p be a prime, k an integer, and $A \subseteq [0, p-1]$. Then, the*

function $f : \{0,1\}^{\upsilon} \to \{0,1\}$ *computed by a* $MOD_{p^k}^A$ *gate can be computed by a* $MOD_p^{\{0\}} \circ AND$ *circuit, where the number of AND gates is at most* $\left(\upsilon + p^k\right)^{|A|p^k}$ *and the fan-in of each AND gate is at most* $|A|\, p^k$.

The proof is based on the following facts.

Fact 2.2.1 (Kummer's Theorem)**.** *The largest power of a prime* p *that divides the binomial coefficient* $\binom{n}{m}$ *is equal to the number of carries when* m *and* $n-m$ *are added in base* p.

Fact 2.2.2. *Let* p *be a prime,* k *a positive integer, and* $x_1, \ldots, x_n \in \{0,1\}$. *Then,*

$$\begin{aligned}\sum_{i=1}^{n} x_i \equiv 0 \pmod{p^k} \quad &\iff \quad \forall i \in [0, k-1] \quad \binom{\sum_{i=1}^{n} x_i}{p^i} \equiv 0 \pmod{p} \\ &\iff \quad \forall i \in [0, k-1] \quad \sum_{\substack{S \subseteq [1,n] \\ |S| = p^i}} \prod_{j \in S} x_j \equiv 0 \pmod{p}.\end{aligned}$$

Proof of Lemma 2.2.1. Let $\mathbf{x} = \langle x_1, \ldots, x_{\upsilon} \rangle \in \{0,1\}^{\upsilon}$ be the input to the $\mathrm{MOD}_{p^k}^A$ gate. For every $a \in A$, let $\mathbf{y}_a$ be a $(\upsilon + p^k)$-tuple such that $[y_a]_1 = x_1, \cdots, [y_a]_{\upsilon} = x_{\upsilon}; [y_a]_{\upsilon+1} = \cdots = [y_a]_{\upsilon+p^k-a} = 1$; and $[y_a]_{\upsilon+p^k-a+1} = \cdots = [y_a]_{\upsilon+p^k} = 0$. Note that,

$$\begin{aligned}\mathrm{MOD}_{p^k}^A(x_1, \ldots, x_{\upsilon}) &= \bigvee_{a \in A} \mathrm{MOD}_{p^k}^{\{a\}}(x_1, \ldots, x_{\upsilon}) \\ &= \bigvee_{a \in A} \mathrm{MOD}_{p^k}^{\{0\}}([y_a]_1, \ldots, [y_a]_{\upsilon+p^k}) \\ &= \mathrm{NOT}\left(\bigwedge_{a \in A} \mathrm{NOT}\left(\mathrm{MOD}_{p^k}^{\{0\}}([y_a]_1, \ldots, [y_a]_{\upsilon+p^k})\right)\right) \\ &= 1 - \prod_{a \in A}\left(1 - \mathrm{MOD}_{p^k}^{\{0\}}([y_a]_1, \ldots, [y_a]_{\upsilon+p^k})\right).\end{aligned}$$

From Fermat's Little Theorem and Fact 2.2.2, it is easy to see that for every $a \in A$,

$$\begin{aligned}
&\mathrm{MOD}_{p^k}^{\{0\}}([y_a]_1, \ldots, [y_a]_{v+p^k}) = 1 \\
&\qquad \iff \quad \sum_{i=1}^{v+p^k} [y_a]_i \equiv 0 \pmod{p^k} \\
&\qquad \iff \quad \forall i \in [0, k-1] \quad \left(\begin{matrix} \sum_{i=1}^{v+p^k} [y_a]_i \\ p^i \end{matrix} \right) \equiv 0 \pmod{p} \\
&\qquad \iff \quad \prod_{i=0}^{k-1} \left(1 - \left(\sum_{S \subseteq [1, v+p^k],\ |S|=p^i} \prod_{j \in S} [y_a]_j \right)^{p-1} \right) \equiv 1 \pmod{p}.
\end{aligned}$$

Thus, a $\mathrm{MOD}_{p^k}^A$ gate can be expressed as a $\mathrm{MOD}_p^{\{0\}}$ gate where the input is a polynomial. Since each term of the polynomial can be computed by an AND gate, the general modular gate $\mathrm{MOD}_{p^k}^A$ can be computed by $\mathrm{MOD}_p^{\{0\}} \circ \mathrm{AND}$ circuits. A simple analysis shows the claimed bounds on the number of terms (the fan-in of the modular gate) and the degree (the fan-in of the AND gate) of the input polynomial. □

Repeated applications of Lemma 2.2.1 and noting that $\mathrm{AND} \circ \mathrm{MOD}_p^{\{0\}} \subseteq \mathrm{MOD}_p^{\{0\}} \circ \mathrm{AND}$ yield the following well-known fact:

Corollary 2.2.2. *Let p be a fixed prime. Every constant-depth circuit of size s with general modular gates where the moduli are constant prime powers of p can be expressed as a $MOD_p^{\{0\}} \circ AND_{O(1)}$ circuit of size $s^{O(1)}$.*

2.2.2 General vs. Regular Modular Gates

It will be interesting to know whether a similar result as that of Lemma 2.2.1 holds when the modulus is not a prime power. A positive answer would be a breakthrough result in circuit complexity since it would imply strong lower bounds for the class of constant-depth circuits over the basis $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathrm{MOD}_m^A\}$. A negative answer would imply a strong lower bound for $\mathrm{MOD}_m^A \circ \mathrm{AND}$ circuits – currently, no superpolynomial lower bounds are known for this class of circuits. We show that in some weak sense general modular gates are more powerful than ordinary modular gates.

Theorem 2.2.3. *For every positive integer m there exists a set $A \subseteq [0, m-1]$ such that the function computed by a MOD_m^A gate cannot be computed by a $MOD_m^{\{0\}}$ gate.*

Proof. Given the positive integer m, we choose a positive integer t such that $t \leq m$, and $A = \{a_1, \ldots, a_t\} \subseteq [0, m-1]$, where $0 \leq a_1 < \cdots < a_t < m$. Define the Boolean function $f : \{0,1\}^n \to \{0,1\}$ as follows: for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$,

$$f(x_1, \ldots, x_n) = 1 \iff (x_1 + \cdots + x_n) \in A \pmod{m}.$$

If we allow each input wire to be connected several times to the gate, then the Boolean function $h : \{0,1\}^n \to \{0,1\}$ computed by the $\mathrm{MOD}_m^{\{0\}}$ gate can be expressed as

$$h(x_1, \ldots, x_n) = 1 \iff c_0 + c_1 x_1 + \cdots + c_n x_n \equiv 0 \pmod{m},$$

where $0 \leq c_0, \ldots, c_n < m$ are fixed for all input values.

Assume that $f(\mathbf{x}) = h(\mathbf{x})$ for all $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$. Let k_0, k_1 be two arbitrary integers, where $k_0 \neq k_1$ and $1 \leq k_0, k_1 \leq n$. Assuming $n > a_1 + 1$, choose $a_1 - 1$ integers $1 \leq k_2, \ldots, k_{a_1} \leq n$ where $k_i \neq k_j$ for all $0 \leq i < j \leq n$. Let $\mathbf{x}', \mathbf{x}'' \in \{0,1\}^n$ such that $\mathbf{x}'$ has 1 in positions $k_0, k_2, \ldots, k_{a_1}$, and 0 everywhere else, and $\mathbf{x}''$ has 1 in positions $k_1, k_2, \ldots, k_{a_1}$, and 0 everywhere else. Since $|\mathbf{x}'| = |\mathbf{x}''| = a_1$, then $f(\mathbf{x}') = f(\mathbf{x}'') = 1$. It follows that, $c_0 + c_{k_0} + c_{k_2} + \cdots + c_{k_{a_1}} \equiv 0 \pmod{m}$ and $c_0 + c_{k_1} + c_{k_2} + \cdots + c_{k_{a_1}} \equiv 0 \pmod{m}$. Therefore, $c_{k_0} \equiv c_{k_1} \pmod{m}$. Since k_0, k_1 were arbitrarily chosen, then $c_1 \equiv \cdots \equiv c_n \pmod{m}$.

Up to this point, we have shown that if $f(\mathbf{x}) = h(\mathbf{x})$ for all $\mathbf{x} = \langle x_1 \ldots x_n \rangle \in \{0,1\}^n$, then

$$h(x_1, \ldots, x_n) = 1 \iff c_0 + c_1(x_1 + \cdots + x_n) \equiv 0 \pmod{m}.$$

It is clear that, for all $1 \leq i \leq t$, $f(\mathbf{x^i}) = 1$ for all $\mathbf{x^i} \in \{0,1\}^n$ such that $\left|\mathbf{x^i}\right| = a_i$. Thus, c_0, c_1 must satisfy $c_0 + a_i c_1 \equiv 0 \pmod{m}$ for all $1 \leq i \leq t$. This is true, if and only if $(m/\gcd(m, c_1)) \mid (a_i - a_j)$ for all $1 \leq i, j \leq t$. That is, $f \equiv h$ if and only if $A = \{a + m'q_1, \ldots, a + m'q_t\}$, where $m' = m/\gcd(m, c_1)$, $0 \leq a < m$ and $1 < q_1 < \cdots < q_t < \frac{m-a}{m'}$. □

2.3 Threshold Gates

A *threshold gate* is labeled with a threshold function. A threshold function $\mathrm{TH}_{\geq\theta}$ is 1 if and only if the sum of the inputs is greater than or equal to some integer θ called the threshold. Similarly, the function $\mathrm{TH}_{\leq\theta}$ is 1 if and only if the sum of the inputs is less than or equal to the threshold θ. Thus, the AND, OR and MAJ functions are the threshold functions $\mathrm{TH}_{\geq n}$, $\mathrm{TH}_{\geq 1}$, $\mathrm{TH}_{\geq n/2}$, respectively. We can generalize threshold functions by associating to each input variable x_i a real weight w_i. The output of these *weighted threshold* functions is determined by comparing $\sum_{i=1}^{n} w_i x_i$ to the threshold θ.

Many arithmetic operations such as addition, multiplication, division and exponentiation can be computed by TC_4^0 circuits. Strong lower bounds are known for depth 2 threshold

circuits, but not for depth 3 threshold circuits. As an example of a lower bound proof, we show an exponential lower bound for a simple depth 2 threshold circuit computing the parity function, where the parity of n Boolean variables $x_1, \ldots, x_n$ is 1 if and only if $\sum_{i=1}^{n} x_i \equiv 1 \pmod 2$.

Theorem 2.3.1. *AND $\circ$ TH circuits computing the parity function of n Boolean variables must have size $2^{n-1}+1$.*

Proof. Suppose there exists an AND $\circ$ TH circuit that computes the parity function. Consider one of the threshold gates. To generalize, suppose the gate computes a weighted threshold function. Let $\mathbf{w} = \langle w_1, \ldots, w_n \rangle \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$. Without loss of generality, we can assume that the threshold gate is a $\mathrm{TH}^{\mathbf{w}}_{\leq \theta}$ gate. Therefore,

$$\mathrm{TH}^{\mathbf{w}}_{\leq \theta}(x_1, \ldots, x_n) = 1 \iff w_1 x_1 + \cdots + w_n x_n \leq \theta.$$

Furthermore, it is easy to see that, on all inputs of odd parity, the threshold gate outputs 1. If it outputs 1 on all inputs of even parity, it would compute a constant function, and thus, could be removed from the circuit. Therefore, for some input $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ of even parity, the threshold gate outputs 0. Let $A = \{i_1, \ldots, i_{2k}\} = \{i : \mathbf{x}_i = 1, 1 \leq i \leq n\}$. Thus,

$$w_{i_1} + \cdots + w_{i_{2k}} > \theta \tag{2.3.1}$$

For each $1 \leq \ell \leq 2k$, let $B_\ell = A - \{i_\ell\}$, and let $\mathbf{x}_{B_\ell}$ be a binary vector of length n which has a 1 in position j for all $j \in B_\ell$, and 0 in the other positions. Clearly, each vector $\mathbf{x}_{B_\ell}$ has odd parity, and thus,

$$w_{i_1} + \cdots + w_{i_{\ell-1}} + w_{i_{\ell+1}} + \cdots + w_{i_{2k}} \leq \theta \tag{2.3.2}$$

From Eq. 2.3.1 and Eq. 2.3.2, it follows that $w_\ell > 0$ for every $\ell \in A$.

For each $\ell \in [1,n] - A$, let $C_\ell = A \cup \{\ell\}$, and let $\mathbf{x}_{C_\ell}$ be a binary vector of length n which has a 1 in position j for all $j \in C_\ell$, and 0 in the other positions. Since each vector $\mathbf{x}_{C_\ell}$ has odd parity, and proceeding as above, we conclude that $w_\ell < 0$ for every $\ell \in [1,n] - A$.

Suppose this threshold gate outputs 0 on some other input $\mathbf{y}$ of even parity which has 1 in position j for all $j \in D$, where $D \subseteq [1,n]$ has even cardinality, and 0 in the other positions. Following the same line of reasoning, we conclude that $w_\ell > 0$ if $\ell \in D$ and $w_\ell < 0$ if $\ell \in [1,n] - D$. Therefore, $A = D$, and as a result, $\mathbf{x} = \mathbf{y}$.

Since the threshold gate was arbitrarily chosen, we have shown that, each threshold gate outputs 0 on *exactly* one input of even parity, and it outputs 1 on all other inputs. Since there are 2^{n-1} inputs of even parity, there must be 2^{n-1} threshold gates.

This lower bound is optimal, since we can construct an AND $\circ$ TH circuit with 2^{n-1} threshold gates that computes the parity function as follows: Associate a threshold gate to

each set $A = \{i_1, \ldots, i_{2k}\} \subseteq [1, n]$ of even cardinality. Let $\theta = 2k$, and if $\ell \in A$, let $w_\ell = 1$, otherwise, let $w_\ell = -1$. It is easily seen that, for every $B \subseteq \{1, \ldots, n\}$,

$$\sum_{j \in B} w_j > \theta \iff B = A.$$

Therefore, the threshold gate outputs 1 if and only if the input vector is $\mathbf{x}_A$, and as a result, the circuit computes the parity function correctly. □

2.4 Exact Gates

Recall that an exact gate computes a symmetric Boolean function which is 1 if and only if the sum of the inputs is equal to some integer. Again, we can generalize exact gates in the same way as threshold gates by associating to each input variable x_i a real weight w_i. Then, the output of the exact gate is 1 if and only if $\sum_{i=1}^{n} w_i x_i$ is equal to some real number.

It is easily seen that an exact gate can be expressed as a binary AND of threshold gates. Hence, in combination with AND gates, exact gates appear to be weaker than threshold gates. To illustrate, we show that AND∘EXACT circuits cannot compute the parity function, while from Section 2.3, the parity function of n Boolean variables can be computed by AND ∘ TH circuits of size 2^{n-1}.

Theorem 2.4.1. *AND ∘ EXACT circuits cannot compute the parity function of n Boolean variables.*

Proof. Suppose that there exists an AND ∘ EXACT circuit that computes the parity function. Consider one of the EXACT and assume it is a weighted gate. The function it computes can be expressed as

$$\text{EXACT}_k(x_1, \ldots, x_n) = 1 \iff w_1 x_1 + \cdots + w_n x_n = k,$$

for some $k \in \mathbb{R}$ and $\mathbf{w} = \langle w_1, \ldots, w_n \rangle \in \mathbb{R}^n$. On all inputs of odd parity, the exact gate outputs 1. For each $1 \leq i \leq n$, let $\mathbf{e}_i$ denote the binary vector of length n which has 1 in position i, and 0 everywhere else. Since $\mathbf{e}_i$ has odd parity, then $w_1 = \cdots = w_n = k$. The vector $\langle 1, 1, 1, 0, \ldots, 0 \rangle \in \{0, 1\}^n$ has odd parity, and therefore, $w_1 = \cdots = w_n = k = 0$. Thus, the exact gate computes a constant function, and, on all inputs of even parity, it outputs 1. Since the exact gate was arbitrarily chosen, we see that the output of the circuit is always 1, a contradiction. □

2.5 Symmetric Gates

A *symmetric gate* is labeled with a symmetric function, and therefore it generalizes modular, exact, and threshold gates. We can associate weights to the input variables of symmetric gates in a similar fashion as with the other types of gates, however the Boolean function that the gate will compute may no longer be symmetric. Symmetric gates are even more interesting than the threshold gates. It has been shown in [25] that any symmetric function can be computed by TC^0_2 circuits. We also know that $\mathrm{ACC}^0[m]$ circuits can be simulated by depth 2 circuits with a symmetric gate of quasipolynomial fan-in at the output and AND gates of polylogarithmic fan-in at the input level (see [50, 9]). There are strong lower bounds for classes of circuits with symmetric gates at the input level, *i.e.,* $\mathrm{AC}^0_1 \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ (see Chapter 4) and $\mathrm{MOD}^{\{0\}}_m \circ \mathrm{SYM}$ (see [30]), but no lower bounds are known for circuits with symmetric gates at other levels with the exception of $\mathrm{TC}^0_1 \circ \mathrm{SYM} \circ \mathrm{AC}^0_1$ circuits (see [27]).

We now state some important properties of the symmetric gates which we use in Chapter 5 to compute pseudorandom function generators. The first property shows that any symmetric function can be computed by $\mathrm{SUM} \circ \mathrm{TC}^0_1$ circuits. The second property shows that depth 2 symmetric circuits, where the fan-in of the output gate is small, are equivalent to a single symmetric gate. Combining these properties with the results of Subsection 2.2.1 shows that $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits can be simulated by $\mathrm{MOD}^{\{0\}}_p \circ \mathrm{TC}^0_1$ circuits.

Lemma 2.5.1 (Maciel & Thérien [35])**.** *$SYM \subseteq SUM \circ TC^0_1$.*

Lemma 2.5.2 (Maciel & Thérien [35])**.** *Any Boolean function of c symmetric gates of fan-in m can be computed by a single symmetric gate of fan-in $(4m)^c$.*

Corollary 2.5.3. *Let p be a fixed prime. Every $CC^0[p^k] \circ SYM$ circuit of size s can be simulated by $MOD^{\{0\}}_p \circ TC^0_1$ circuits of size $s^{O(1)}$.*

Proof. Using Corollary 2.2.2, Lemma 2.5.2 and Lemma 2.5.1, we obtain:

$$\mathrm{CC}^0[p^k] \circ \mathrm{SYM} = \mathrm{MOD}^{\{0\}}_p \circ \mathrm{AND}_{O(1)} \circ \mathrm{SYM} = \mathrm{MOD}^{\{0\}}_p \circ \mathrm{SYM} = \mathrm{MOD}^{\{0\}}_p \circ \mathrm{SUM} \circ \mathrm{TC}^0_1.$$

The result follows by connecting the inputs of the SUM gates directly to the modular gates. □

Chapter 3

Polynomial Method in Circuit Complexity

Historically, the study of polynomials has been an important tool in proving lower bounds on the circuit complexity of explicit Boolean functions. Polynomials were first used by Minsky & Papert [36] to show that perceptrons cannot compute the parity function. Their results were strengthened by Aspnes *et al.* [6] who used voting polynomials to show how well perceptrons can approximate the parity function. Polynomials were also used by Smolensky [45] to obtain lower bounds on the size of $\mathrm{ACC}^0[p]$ circuits. Yao [50] used representing polynomials to show that $\mathrm{ACC}^0[m]$ circuits can be simulated by depth 2 circuits with a symmetric gate of quasipolynomial fan-in at the output and AND gates of polylogarithmic fan-in at the input level.

We use polynomials to obtain lower bounds on circuits computing the square-free function. We show that the square-free function cannot be computed by depth 2 circuits with $\mathrm{AND}_{\lfloor 0.14 \ln n \rfloor}$ gates at the input and a modular $\mathrm{MOD}_{p^k}^{\{0\}}$ or a weighted threshold gate at the output. We remark that p, k and the weights are not bounded above by any function and can in fact be exponential. These results provide non-trivial lower bounds on the complexity of a number theoretic problem which is closely related to the integer factorization problem. The work on the complexity of the square-free function has appeared in [39] and is based on a number theoretic sieve method similar to that of [13, 14, 15]. We also show how to use random restriction to obtain lower bounds on the size of $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{AND}$ circuits using a lower bound result on the fan-in of the AND gates. Attempting to answer an open problem by Barrington & Tardos [46], we define the *threshold* of a polynomial P to be the smallest integer α such that P^α has the maximum number of terms that can be obtained by raising P to any power. Then, we obtain an interesting trade-off result between the fan-in of the AND gates in $\mathrm{MOD}_m^A \circ \mathrm{AND}$ circuits and the threshold of m-representing polynomials.

3.1 Representing Polynomials

Researchers have translated the problems of showing lower bounds on the circuit complexity of some explicit Boolean function f into showing lower bounds on some appropriate complexity measure on the polynomials which in some sense represent f. One such measure is the degree of a polynomial representing f. If the degree of the representing polynomial is high, the Boolean function is considered hard. For Boolean inputs $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$, we simply need to consider multilinear polynomials since for every integer $\alpha > 0$, $x_i^\alpha = x_i$. We now consider the different polynomial representations used in this thesis.

3.1.1 Polynomials over $\mathbb{Z}_m$

In the last years, attention has been focused on polynomials over $\mathbb{Z}_m$. A polynomial of degree d over $\mathbb{Z}_m$ can be represented by a circuit consisting of an unbounded fan-in MOD_m^A gate at the output and AND gates at the input level, where each AND gate has fan-in at most d.

Obtaining lower bounds on the degree of polynomials over $\mathbb{Z}_m$ is a difficult task. When m is prime, we can make use of the fact that $\mathbb{Z}_m$ is a field, and use well-established techniques to obtain the desired results. However, when m is composite other techniques must be developed.

A polynomial P over $\mathbb{Z}_m$ is of the form

$$P(x_1, \ldots, x_n) = \sum_{H \in \mathcal{H}} c_H \prod_{i \in H} x_i, \tag{3.1.1}$$

where

$$\mathcal{H} \subseteq \mathcal{P}(\{1, \ldots, n\}) \quad \text{and} \quad c_H \in \{0, \ldots, m-1\}.$$

We call the largest value of $|H|$ in the representation (3.1.1) the *degree* of P and write $\deg(P)$. We call the number of coefficients c_H, or equivalently $|\mathcal{H}|$, the *size* of P and write $\mathrm{size}(P)$.

Let us define what it means for a polynomial P over $\mathbb{Z}_m$ to represent a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Let $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ be an arbitrary vector.

— *Strong Representation*

$$f(x_1, \ldots, x_n) \equiv P(x_1, \ldots, x_n) \pmod{m}.$$

— *One-Sided Representation*

$$f(x_1, \ldots, x_n) = 0 \iff P(x_1, \ldots, x_n) \equiv 0 \pmod{m}.$$

— *Weak Representation*

There exists a set $A \subseteq \{0, 1, \ldots, m-1\}$ such that,

$$f(x_1, \ldots, x_n) = 0 \iff P(x_1, \ldots, x_n) \bmod m \in A.$$

We say the polynomial P *strongly*, *one-sidedly*, or *weakly* m-represents the Boolean function f.

Representations of Boolean functions via polynomials over $\mathbb{Z}_m$ have been studied in [19, 44, 7, 22, 47, 46, 23, 2]. In [19, 44] some lower bounds are obtained for polynomials over $\mathbb{Z}_2$ strongly 2-representing the Boolean function QR deciding the quadratic residuacity of an n-bit integer. Alon & Beigel [2] show a quasipolynomial lower bound on the size of depth 3 circuits of the form $\mathrm{MAJ} \circ \mathrm{MOD}_m^{\{0\}} \circ \mathrm{AND}_2$ by proving that polynomials of degree 2 over $\mathbb{Z}_m$ cannot approximate parity well. In [7, 22, 47] lower bounds are obtained for polynomials one-sidedly m-representing the OR, $\mathrm{MOD}_m^{\{0\}}$, and $\mathrm{MOD}_m^{\{1,\ldots,m-1\}}$ Boolean functions. Tardos & Barrington [46] show a lower bound of $\Omega\left((\log n)^{1/(r-1)}\right)$ on the degree of polynomials weakly m-representing the OR function, where r is the number of distinct primes dividing m and m is constant. Subsequent to their work Grolmusz [23] using lower bounds on the communication complexity of the generalized inner product (GIP) function, showed that polynomials weakly m-representing GIP must have $\Omega(\log n)$ degree. To this day, no superlogarithmic lower bounds are known on the weak degree of m-representing polynomials for some explicit Boolean function.

3.1.2 Threshold Representation

Similarly to the case of polynomials over $\mathbb{Z}_m$, for a polynomial P in n variables over the real numbers we define the *total degree* $\deg(P)$ as the largest sum $\sum_{i \in H} \alpha_{H,i}$ and the *size* $\mathrm{size}(P)$ as the number of coefficients c_H in the representation

$$P(x_1, \ldots, x_n) = \sum_{H \in \mathcal{H}} c_H \prod_{i \in H} x_i^{\alpha_{H,i}}, \tag{3.1.2}$$

where

$$\mathcal{H} \subseteq \mathcal{P}(\{1, 2, \ldots, n\}) \quad \text{and} \quad 0 \neq c_H \in \mathbb{R}, \alpha_{H,i} \in \mathbb{R}.$$

For a real number w we define the sign function as

$$\mathrm{sign}(w) = \begin{cases} 1, & \text{if } w \geq 0, \\ 0, & \text{if } w < 0. \end{cases}$$

A polynomial P over $\mathbb{R}$ provides a *threshold* representation of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$,

$$\operatorname{sign}(P(x_1, \ldots, x_n)) = f(x_1, \ldots, x_n).$$

Furthermore, in the case of real polynomials, the Boolean values 0 and 1 can be interpreted as two arbitrary real values ν_0 and ν_1, not necessarily $\nu_0 = 0$ and $\nu_1 = 1$. It is easy to see that the degree of the corresponding polynomials does not depend on the particular choice of ν_0, ν_1 because they are equivalent under a linear transformation of variables [29]. However, the size of the corresponding polynomials depends on the particular choice of ν_0 and ν_1. In fact, there are examples of Boolean functions demonstrating that for $(\nu_0, \nu_1) = (0, 1)$ and $(\nu_0, \nu_1) = (1, -1)$ the gap between the numbers of monomials of the corresponding polynomials for these two representations can be exponentially large [29].

Threshold representations of Boolean functions via real polynomials have been studied in a number of works [16, 17, 29, 38]. These papers contain many general estimates together with lower bounds for some particular Boolean functions.

3.2 Random Restrictions

The random restriction technique — introduced in [21] — provided the first breakthrough lower bounds on the size of small depth circuits computing the parity function. The idea behind the proofs is that by randomly assigning some of the input variables to 0 or 1, the circuit is simplified with high probability while it still computes the same function on a subset of the variables. Other circuit lower bounds have been proven in [26, 42, 24] by using variations of the random restriction technique. Grolmusz [23] obtained quasipolynomial lower bounds on the size of $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{AND}$ circuits computing the OR function by using the random restriction technique in combination with the lower bound of [46] on the fan-in of the AND gates. To illustrate this important technique, we show how to obtain an exponential lower bound on the size of $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{AND}$ circuits computing the majority function MAJ by using a result of Tsai [48] on the degree of the one-sided m-representing polynomials. Similar results can be shown for the MidBit and MOD_q functions.

Lemma 3.2.1 (Tsai [48])**.** *Assume that a polynomial P over $\mathbb{Z}_m$* one-sidedly m-represents *the threshold function $TH_{\geq\theta}$ of n variables, that is, for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$,*

$$TH_{\geq\theta}(x_1, \ldots, x_n) = 0 \iff P(x_1, \ldots, x_n) \equiv 0 \pmod{m}.$$

Then, for sufficiently large n,

$$deg(P) \geq \theta.$$

Theorem 3.2.2. *Assume that a polynomial P over $\mathbb{Z}_m$* one-sidedly m-represents *the majority function MAJ of n variables, that is, for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$,*

$$MAJ(x_1, \ldots, x_n) = 0 \iff P(x_1, \ldots, x_n) \equiv 0 \pmod{m}.$$

Then, for sufficiently large n,

$$size\,(P) = 2^{\Omega(n)}.$$

Proof. Let

$$P(x_1, \ldots, x_n) = \sum_{H \in \mathcal{H}} a_H \prod_{i \in H} x_i,$$

where $\mathcal{H} \subseteq \mathcal{P}\,(\{1, \ldots, n\})$ and $0 \neq a_H \in \mathbb{Z}_m$, be a polynomial over $\mathbb{Z}_m$ one-sidedly m-representing the majority function. Let $p = \frac{1}{2}$ and randomly and independently restrict each variable x_i, $i = 0, 1, \ldots, n$, according to the mapping $\rho : \{0,1\}^n \to \{0, *\}$, where

$$\rho(x_i) = \begin{cases} 0, & \text{with probability } 1-p, \\ \star, & \text{with probability } p. \end{cases}$$

Recall that, if $\rho(x_i) = *$, then x_i remains unrestricted.

For every $|H| \geq \frac{n}{4}$,

$$\Pr\left[\prod_{i \in H} \rho(x_i) \neq 0\right] = p^{|H|} \leq 2^{-\frac{n}{4}}. \tag{3.2.1}$$

Suppose $|H| = 2^{\epsilon n/4}$ for a small enough $0 < \epsilon < 1$. Then, because of Eq. 3.2.1, the degree of the polynomial after the restriction will be small with high probability:

$$\Pr\left[\deg\left(P|_\rho\right) < \frac{n}{4}\right] \geq 1 - o(1)$$

while $P|_\rho$ represents $\mathrm{TH}_{\geq n/2}$ of at least $pn = \frac{n}{2}$ variables. From Lemma 3.2.1, $\deg\left(P|_\rho\right) \geq n/2 > n/4$, hence a contradiction. □

3.3 The Weak Degree and the Threshold

As we have mentioned, obtaining a superlogarithmic lower bound on the weak-degree of m-representing polynomials is a challenging open problem. In this section we show trade-off results of the form

$$\deg\left(P\right) = \frac{\Omega(n)}{\mathrm{thresh}\,(P)}$$

between the weak-degree and the *threshold* of polynomials weakly m-representing some specific Boolean functions. Thus, to obtain a lower bound on the weak-degree we can try to obtain an upper bound on the threshold.

3.3.1 The Threshold of a Polynomial

Let P be a polynomial. When P is raised to a power α, we expect the size of P^α to be larger than the size of P. However, for Boolean variables this is not always true. There exists an integer α such that P^α has the maximum number of terms that can be obtained by raising P to any power. More precisely, let $\text{thresh}(P)$ denote the smallest positive integer α such that for any integer $\beta > \alpha$,

$$\text{size}(P^\alpha) \geq \text{size}\left(P^\beta\right).$$

The threshold measures how long the size of the polynomial will grow. For example, the polynomial

$$x_1x_2x_3 + x_2x_4x_5 + x_1x_5 + x_5$$

has threshold 2. We now prove an upper bound on the threshold of a polynomial.

Lemma 3.3.1. *Let P be a polynomial over the Boolean variables $x_1, \ldots, x_n$ with coefficients from some set S. Then,*

$$\textit{thresh}(P) \leq n.$$

Proof. P can be written as

$$P(x_1, \ldots, x_n) = \sum_{H \in \mathcal{H}} a_H X_H,$$

where $\mathcal{H} \subseteq \mathcal{P}(\{1, \ldots, n\})$, $X_H = \prod_{i \in H} x_i$, and $0 \neq a_H \in S$. We need to show that, if k is a nonnegative integer, then

$$\text{size}\left(P^k\right) \leq \text{size}(P^n).$$

Obviously, the lemma is true for $k \leq n$. Assume $k > n$. Assume to the contrary that $\text{size}\left(P^k\right) > \text{size}(P^n)$. Let ℓ be the smallest integer such that $\text{size}\left(P^\ell\right) > \text{size}(P^n)$. Clearly, $n < \ell \leq k$, and $\text{size}\left(P^{\ell-1}\right) \leq \text{size}(P^n)$. Hence, there exists a monomial $M = X_{H_1} \cdots X_{H_\ell} = X_{H_1 \cup \cdots \cup H_\ell}$ in the expansion of P^ℓ which is not found in the expansion of $P^{\ell-1}$. It means that, for every $i_1, \ldots, i_{\ell-1} \in \{1, \ldots, \ell\}$,

$$H_1 \cup \cdots \cup H_\ell \supset H_{i_1} \cup \cdots \cup H_{i_{\ell-1}}.$$

Therefore, for every $i \in \{1, \ldots, \ell\}$, H_i has an element which is not an element of any of the sets H_j for $j \in \{1, \ldots, \ell\}$ and $j \neq i$. This implies that the monomial M has $\ell > n$ variables, a contradiction. □

3.3.2 The $\mathrm{MOD}_q^{\{r\}}$ Function

The following lemma is due to Green [22].

Lemma 3.3.2 (Green [22])**.** *Let q and m be relatively prime. Let P be a polynomial over the Boolean variables $x_1, \ldots, x_n$ with integer coefficients such that P is not a constant function modulo m. Suppose that for some integer $0 \leq r \leq q-1$ and for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ it holds that,*

$$\sum_{i=1}^{n} x_i \equiv r \pmod{q} \Rightarrow P(x_1, \ldots, x_n) \equiv 0 \pmod{m},$$

Then, for sufficiently large n,

$$deg(P) \geq \left\lfloor \frac{n}{2(q-1)} \right\rfloor.$$

We now show how to relate the weak degree to the threshold of the weak m-representing polynomial.

Theorem 3.3.3. *Let q and m be relatively prime. Let P be a polynomial over the Boolean variables $x_1, \ldots, x_n$ with integer coefficients such that P is not a constant function modulo m. Suppose that for some set $A \subseteq \{0, \ldots, m-1\}$, some integer $0 \leq r \leq q-1$, and for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ it holds that,*

$$\sum_{i=1}^{n} x_i \equiv r \pmod{q} \iff P(x_1, \ldots, x_n) \in A \pmod{m}.$$

Then, for sufficiently large n,

$$deg(P) \geq \left\lfloor \frac{n}{2(q-1)thresh(P)} \right\rfloor.$$

Proof. Let $V = \{P(x_1, \ldots, x_n) : x_1, \ldots, x_n \in \{0,1\}\}$. Define

$$Q(x_1, \ldots, x_n) = \prod_{\substack{v \in V \\ v \bmod m \in A}} (P(x_1, \ldots, x_n) - v).$$

It follows from Lemma 3.3.1 and the definition of the threshold that

$$\deg(Q) \leq \deg\left(P^{\text{thresh}(P)}\right) \leq \text{thresh}(P)\deg(P). \tag{3.3.1}$$

Furthermore,

$$[P(x_1,\ldots,x_n) \bmod m] \in A \iff Q(x_1,\ldots,x_n) = 0.$$

Therefore, there exists a natural number m relatively prime to q such that Q is not constant modulo m and

$$\sum_{i=1}^{n} \equiv r \pmod{q} \Rightarrow Q(x_1,\ldots,x_n) \equiv 0 \pmod{m}.$$

Note that Q satisfies the conditions of Lemma 3.3.2. Hence, from Lemma 3.3.2 and Eq. 3.3.1 it follows that

$$\deg(Q) \geq \left\lfloor \frac{n}{2(q-1)} \right\rfloor \quad \text{and} \quad \deg(P) \geq \left\lfloor \frac{n}{2(q-1)\text{thresh}(P)} \right\rfloor.$$

□

3.3.3 The Quadratic Residuacity Function

An integer x is a *quadratic residue modulo p* if there exists an integer $a \in \{0,\ldots,p-1\}$ such that, $x \equiv a^2 \pmod{p}$. Define the function

$$\text{QR}(x) = \begin{cases} 0, & \text{if } x \text{ is a quadratic residue modulo } p, \\ 1, & \text{otherwise.} \end{cases}$$

For a given integer $n \geq 1$, we can identify x, $0 \leq x \leq 2^n - 1$, and its binary representation $x_1 \ldots x_n$ (if necessary we add several leading zeros) and consider $\text{QR}(x)$ as a Boolean function of n variables, *i.e.,* think of it as $\text{QR}(x_1,\ldots,x_n)$.

A careful examination of Theorem 5 in [19] shows that it can be stated as follows.

Lemma 3.3.4 (Coppersmith & Shparlinski [19])**.** *Let $n = \lfloor p \rfloor$, where p is a prime number. Let P be a polynomial over the Boolean variables $x_1,\ldots,x_n$ with rational coefficients. Suppose that for every $\mathbf{x} = \langle x_1,\ldots,x_n\rangle \in \{0,1\}^n$ it holds that,*

$$QR(x_1,\ldots,x_n) = P(x_1,\ldots,x_n).$$

Then, for sufficiently large n,

$$deg(P) \geq 0.04n.$$

In a similar way to the proof of the Theorem 3.3.3 we relate the weak degree of the polynomial P weakly m-representing QR to the threshold of P.

Theorem 3.3.5. *Let $n = \lfloor p \rfloor$, where p is a prime number. Let m be a natural number and let P be a polynomial over the Boolean variables $x_1, \ldots, x_n$ with integer coefficients. Suppose that for some set $A \subseteq \{0, \ldots, m-1\}$, and for every $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ it holds that,*

$$QR(x_1, \ldots, x_n) = 0 \iff P(x_1, \ldots, x_n) \in A \pmod{m}$$

Then, for sufficiently large n,

$$deg\,(P) \geq \frac{0.04n}{thresh\,(P)}.$$

Proof. Let $V = \{P(x_1, \ldots, x_n) : x_1, \ldots, x_n \in \{0,1\}\}$. For $v \in V$, define

$$L_v(x_1, \ldots, x_n) = \frac{\prod_{u \in V - \{v\}} (P(x_1, \ldots, x_n) - u)}{\prod_{u \in V - \{v\}} (v - u)}.$$

For every $u, v \in V$, it is easy to see that L_u and L_v when expanded have exactly the same monomials.

Define

$$Q(x_1, \ldots, x_n) = \sum_{\substack{v \in V \\ v \bmod m \notin A}} L_v(x_1, \ldots, x_n)\ .$$

It follows from Lemma 3.3.1 and the definition of the threshold that

$$\deg(Q) \leq \deg\left(P^{\mathrm{thresh}(P)}\right) \leq \mathrm{thresh}\,(P) \deg(P)\,. \tag{3.3.2}$$

By inspection, one can verify that, for $v = P(x_1, \ldots, x_n)$,

$$L_v(x_1, \ldots, x_n) = 1 \quad \text{and} \quad L_u(x_1, \ldots, x_n) = 0 \text{ for } u \in V - \{v\}.$$

Thus,

$$P(x_1, \ldots, x_n) \bmod m \in A \Rightarrow Q(x_1, \ldots, x_n) = 0,$$

and

$$P(x_1, \ldots, x_n) \bmod m \notin A \Rightarrow Q(x_1, \ldots, x_n) = 1.$$

Note that Q satisfies the conditions of Lemma 3.3.4. Hence, from Lemma 3.3.4 and Eq. 3.3.2 it follows that

$$\deg(Q) \geq 0.04n \quad \text{and} \quad \deg(P) \geq \frac{0.04n}{\mathrm{thresh}\,(P)}.$$

□

3.4 Complexity of the Square-Free Function

In this section we obtain lower bounds on the degree and the size of polynomials over $\mathbb{Z}_m$ which strongly or one-sidedly m-represent the Boolean function SF which decides if a given n-bit integer is square-free. These results provide non-trivial lower bounds on the complexity of a number theoretic problem which is closely related to the integer factorization problem. Similar lower bounds are also obtained for polynomials over the reals which provide a threshold representation of the above Boolean function. Expressing our results in terms of circuits shows that the square-free function cannot be computed by depth 2 circuits with $\mathrm{AND}_{\lfloor 0.14 \ln n \rfloor}$ gates at the input and a modular $\mathrm{MOD}_{p^k}^{\{0\}}$ or a weighted threshold gate at the output. We remark that p, k and the weights are not bounded above by any function and can in fact be exponential. We also show that some simple number theoretic observations allow us to obtain quite strong lower bounds on several other complexity characteristics of testing if a given integer is square-free.

We recall that an integer x is *square-free* if there is no prime p such that $p^2 \mid x$. We define the function

$$\mathrm{SF}(x) = \begin{cases} 1, & \text{if } x \text{ is square-free,} \\ 0, & \text{otherwise.} \end{cases}$$

For a given integer $n \geq 1$, we can identify x, $0 \leq x \leq 2^n - 1$, and its binary representation $x_1 \ldots x_n$ (if necessary we add several leading zeros) and consider $\mathrm{SF}(x)$ as a Boolean function of n variables.

3.4.1 Previous Work

In the series of papers [13, 14, 15] lower bounds were obtained on the circuit complexity, sensitivity, degree of polynomial representations, and other complexity characteristics of testing square-free numbers and computing the greatest common divisor. As in [19, 44], the method of [13, 14, 15] is based on the uniformity of distribution of long patterns of $0, 1$ in the values of $\mathrm{SF}(x)$. For the quadratic residuacity a similar property was established in [19, 44] by using the very powerful Weil estimate; in [13, 14, 15] a sieve method was used for this purpose. In particular, for a strongly 2-representing polynomial P the lower bound

$$\deg(P) \geq 0.165 \ldots n$$

was obtained in [13]. The sieve method was also applied to obtain a lower bound of order $n^{1/2}$ on the degree of real polynomials P which approximate SF in the following sense: for all $1 \leq x \leq 2^n - 1$,

$$|\mathrm{SF}(x) - P(x_1, \ldots, x_n)| \leq 1/3$$

where $x = x_1 \ldots x_n$ is the binary representation of x. These lower bounds are derived from the asymptotic formula for the sensitivity of the function SF obtained in [13]. Unfortunately,

there is no known link between the sensitivity and the degrees of m-representing polynomials, $m \geq 3$, and of threshold representations.

Alternative methods of [1] yield stronger but less explicit complexity results (which apply to primality testing as well). However, these approaches work neither for m-representing polynomials, $m \geq 3$, nor for threshold representations.

Here we use the technique of [13, 14, 15] to obtain several new results about the different polynomial representations of the function SF.

3.4.2 Number of Primes and Binomial Bound

Let $\mathbb{P}_x$ denote the set of primes less than or equal to x.

We use the following well-known asymptotic formulae (see [20] for example):

$$\ln\left(\prod_{p\in\mathbb{P}_x} p\right) = x + O\left(\frac{x}{\ln x}\right), \qquad x \to \infty. \tag{3.4.1}$$

and

$$|\,\mathbb{P}_x| = \frac{x}{\ln x} + O\left(\frac{x}{\ln^2 x}\right), \qquad x \to \infty. \tag{3.4.2}$$

The following estimate can be found in [44].

Lemma 3.4.1. *For any integers L and N, $0 \leq L < N/2$, it holds that*

$$\sum_{K=0}^{L} \binom{N}{K} \leq 2^{H(L/N)N},$$

where $H(\gamma) = -\gamma\log\gamma - (1-\gamma)\log(1-\gamma)$, $0 < \gamma < 1$, is the binary entropy function.

3.4.3 An Important Property of the Square-Free Function

The following result is due to Shparlinski.

Lemma 3.4.2. *Let $\eta \geq 1$ be an integer and define k from the inequalities*

$$2^k \geq \eta^2 > 2^{k-1}.$$

Let $\eta < p_1 < \ldots < p_\eta$ be the first η primes which are greater than η. Then, for any η-dimensional binary vector $\langle \sigma_1, \ldots, \sigma_\eta \rangle$ there exists an integer y, such that $0 \le y \le \exp(4\eta \ln \eta + O(\eta \ln \ln \eta))$ and

$$SF(2^k y + p_i) = \sigma_i, \qquad i = 1, \ldots, \eta.$$

Proof. Let

$$Q = \prod_{\substack{p \le \eta \\ p \in \mathbb{P}}} p \qquad \text{and} \qquad R = 2^k Q.$$

From Eq. 3.4.1 we see that $Q = \exp(O(\eta))$. Thus, it is enough to show that there exists an integer u such that $0 \le u \le \exp(4\eta \ln \eta + O(\eta \ln \ln \eta))$ and

$$\mathrm{SF}(Ru + p_i) = \sigma_i, \qquad i = 1, \ldots, \eta. \tag{3.4.3}$$

We remark that $\gcd(p_i, R) = 1$, $i = 1, \ldots, \eta$.

Let $\mathcal{I}$ be the set of subscripts i for which $\sigma_i = 0$ and let $\mathcal{J}$ be the set of subscripts j for which $\sigma_j = 1$. Let

$$q = \prod_{i \in \mathcal{I}} p_i^2.$$

From the Chinese Remainder Theorem we conclude that there exists an integer a, $0 \le a \le q - 1$, such that $Ra \equiv -p_i \pmod{p_i^2}$ for all $i \in \mathcal{I}$. Therefore, $R(qz + a) + p_i \equiv 0 \pmod{p_i^2}$ for all $i \in \mathcal{I}$ and any integer z. Now we show that one can select a small z for which

$$\mathrm{SF}\left(R(qz + a) + p_j\right) = 1, \qquad j \in \mathcal{J}.$$

For $Z \ge 1$, we denote by $L_j(Z)$ the number of not square-free numbers of the form $R(qz + a) + p_j$ with $1 \le z \le Z$, $j \in \mathcal{J}$. To prove the lemma it is sufficient to show that for some appropriate Z,

$$\sum_{j \in \mathcal{J}} L_j(Z) < Z. \tag{3.4.4}$$

First of all, we remark that, for $i \in \mathcal{I}$ and $j \in \mathcal{J}$,

$$R(qz + a) + p_j \not\equiv 0 \pmod{p_i^2}.$$

Otherwise, we have $p_i^2 \mid (p_j - p_i)$ which is impossible.

For any prime p with $\gcd(p, q) = 1$, the congruence

$$R(qz + a) + p_j \equiv 0 \pmod{p^2}, \qquad 1 \le z \le Z,$$

has at most Z/p^2+1 solutions. Obviously, it does not have solutions for $p^2 > Rq(Z+1)+R$. Let $V = (3RqZ)^{1/2}$.

The smallest prime divisor of any number $R(qz+a)+p_j$ exceeds η. Therefore,

$$\begin{aligned} L_j(Z) &\le \sum_{\substack{\eta<p\le V \\ \gcd(p,q)=1}} \left(\frac{Z}{p^2}+1\right) \le Z\sum_{p>\eta}\frac{1}{p^2} + O\left(\frac{V}{\ln V}\right) \\ &\le Z\sum_{\nu\ge\lfloor\log\eta\rfloor}\sum_{2^{\nu+1}>p\ge 2^\nu}\frac{1}{p^2} + O\left(\frac{V}{\ln V}\right) \\ &\le Z\sum_{\nu\ge\lfloor\log\eta\rfloor}\frac{\pi(2^{\nu+1})}{2^{2\nu}} + O\left(\frac{V}{\ln V}\right) \\ &\le O\left(Z\sum_{\nu\ge\lfloor\log\eta\rfloor}\frac{1}{2^\nu\nu} + \frac{V}{\ln V}\right) = O\left(\frac{Z}{\eta\ln\eta}+\frac{V}{\ln V}\right). \end{aligned}$$

Letting $Z = \eta^2 Rq$ we obtain the inequality (3.4.4), provided that η is large enough. Therefore, there exists an integer u satisfying condition (3.4.3) and $0 \le u \le q(Z+1) \le 2\eta^2 Rq^2$.

Now, from the Prime Number Theorem (Eq. 3.4.2), we conclude that $p_\eta = \eta\ln\eta + O(\eta)$. Therefore, we have $q \le \exp(2\eta\ln\eta + O(\eta\ln\ln\eta))$. Finally, from Eq. 3.4.1 we see that $R = \exp(O(\eta))$, and the result follows. □

3.4.4 Strong Representation

First of all we consider the strong representation of the square-free function.

Theorem 3.4.3. *Let m be an arbitrary integer not necessarily bounded by some function. Assume that a polynomial P over $\mathbb{Z}_m$* strongly m-represents *$SF(x)$, that is, it is such that for any x, $1 \le x \le 2^n - 1$,*

$$P(x_1,\ldots,x_n) \equiv SF(x) \pmod m,$$

where $x = x_1\ldots x_n$ is the binary representation of x. Then, for sufficiently large n,

$$deg(P) \ge 0.14\ln n \qquad and \qquad size(P) \ge \frac{n}{5\ln n}.$$

Proof. Assuming that n is large enough, let

$$\eta = \left\lceil \frac{n}{5 \ln n} \right\rceil .$$

Let $p_1, \ldots, p_\eta$ and k be defined as in Lemma 3.4.2.

Denote by τ the number of monomials $\mu_j(\mathbf{w})$ in $\mathbf{w}$, such that for every k-dimensional vector $\mathbf{w} = \langle w_1, \ldots, w_k \rangle \in \{0,1\}^k$, the polynomial P is written as

$$P(x_1, \ldots, x_{n-k}, w_1, \ldots, w_k) = \sum_{j=1}^{\tau} \mu_j(\mathbf{w}) P_j(x_1, \ldots, x_{n-k})$$

with some polynomials $P_j(x_1, \ldots, x_{n-k})$ over $\mathbb{Z}_m$.

Obviously,

$$\tau \le \sum_{l=0}^{\deg(P)} \binom{k}{l} \qquad \text{and} \qquad \tau \le \operatorname{size}(P). \tag{3.4.5}$$

As in the proof of Lemma 3.4.2, we note that $p_1 < \cdots < p_\eta < \eta^2 \le 2^k$. For every $i = 1, \ldots, \eta$, we add several leading zeros to the binary representation of p_i to obtain binary strings $\mathbf{w}_i$ of length k.

If $\tau < \eta$, then there exist η integer coefficients $c_1, \ldots, c_\eta$, not all equal to zero, such that

$$\sum_{i=1}^{\eta} c_i \, \mu_j(\mathbf{w}_i) = 0, \qquad j = 1, \ldots, \tau.$$

Therefore we have the identity:

$$\sum_{i=1}^{\eta} c_i P\left(x_1, \ldots, x_{n-k}, [\mathbf{w}_i]_1, \ldots, [\mathbf{w}_i]_k\right) = 0.$$

Without loss of generality we can also assume that

$$\gcd(c_1, \ldots, c_\eta) = 1.$$

Then, for some $1 \le i_0 \le \eta$ we have $c_{i_0} \not\equiv 0 \pmod{m}$.

One easily verifies that $2^{n-k} = \exp(5\eta \ln \eta + O(\eta))$. Hence, from Lemma 3.4.2 we derive that there exists x, $0 \le x \le 2^{n-k}$, such that for $i = 1, \ldots, \eta$,

$$P\left(x_1, \ldots, x_{n-k}, [\mathbf{w}_i]_1, \ldots, [\mathbf{w}_i]_k\right) \equiv \mathrm{SF}(2^k x + p_i) \equiv \begin{cases} 1, & \text{if } i = i_0, \\ 0, & \text{if } i \ne i_0, \end{cases} \pmod{m}$$

where $x = x_1 \dots x_{n-k}$ is the binary representation of x (with several leading zeros, if necessary, to make it of length $n-k$). Hence,

$$\sum_{i=1}^{\eta} c_i P\left(x_1, \dots, x_{n-k}, [\mathbf{w}_i]_1, \dots, [\mathbf{w}_i]_k\right) \equiv c_{i_0} \not\equiv 0 \pmod{m}.$$

From the obtained contradiction we see that $\tau \geq \eta \geq 2^{(k-1)/2}$. Taking into account that $H(0.1) < 1/2$ and $0.1/\ln 2 \geq 0.14$, from the inequalities (3.4.5) and Lemma 3.4.1 we obtain the desired result. □

3.4.5 One-Sided Representation

Theorem 3.4.4. *Let $m = p^{\nu}$ be a prime power not necessarily bounded by some function. Assume that a polynomial P over $\mathbb{Z}_m$* one-sidedly m-represents *$SF(x)$, that is, it is such that for any x, $1 \leq x \leq 2^n - 1$,*

$$P(x_1, \dots, x_n) \equiv 0 \pmod{m} \iff SF(x) = 0,$$

where $x = x_1 \dots x_n$ is the binary representation of x. Then, for sufficiently large n,

$$deg\,(P) \geq 0.14 \ln n \qquad and \qquad size\,(P) \geq \frac{n}{5 \ln n}.$$

Proof. As in the proof of Theorem 3.4.3 we obtain that, for some $1 \leq i_0 \leq \eta$, and some $u \not\equiv 0 \pmod{m}$,

$$P(x_1, \dots, x_{n-k}, [\mathbf{w}_i]_1, \dots, [\mathbf{w}_i]_k) \equiv \begin{cases} u, & \text{if } i = i_0, \\ 0, & \text{if } i \neq i_0, \end{cases} \pmod{m}.$$

Also $c_{i_0} \not\equiv 0 \pmod{p}$, and hence, $\gcd(c_{i_0}, m) = 1$. Therefore,

$$\sum_{i=1}^{\eta} c_i P(x_1, \dots, x_{n-k}, [\mathbf{w}_i]_1, \dots, [\mathbf{w}_i]_k) \equiv c_{i_0} u \not\equiv 0 \pmod{m},$$

and as in the proof of Theorem 3.4.3 we obtain the desired result. □

3.4.6 Threshold Representation

We now consider the threshold representation of the SF function.

Theorem 3.4.5. *Let α_0, α_1 be two distinct real numbers, and $n \geq 1$ be an integer. Assume that a polynomial P over $\mathbb{R}$ with coefficients not necessarily bounded by some function, provides a* threshold *representation of $SF(x)$, that is, it is such that for any x, $1 \leq x \leq 2^n - 1$,*

$$sign\,(P\,(\alpha_{x_1}, \ldots, \alpha_{x_n})) = SF(x),$$

where $x = x_1 \ldots x_n$ is the binary representation of x. Then, for sufficiently large n,

$$deg\,(P) \geq 0.14 \ln n \qquad and \qquad size\,(P) \geq \frac{n}{5 \ln n}.$$

Proof. We proceed as in the proof of Theorem 3.4.3. Assuming that n is large enough, let

$$\eta = \left\lceil \frac{n}{5 \ln n} \right\rceil.$$

Let $p_1, \ldots, p_\eta$ and k be defined as in Lemma 3.4.2.

Denote by τ the number of monomials $\mu_j(\mathbf{w})$ in $\mathbf{w}$, such that for every k-dimensional vector $\mathbf{w} = \langle w_1, \ldots, w_k \rangle \in \{\alpha_0, \alpha_1\}^k$, the polynomial P is written as

$$P(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}}, w_1, \ldots, w_k) = \sum_{j=1}^{\tau} \mu_j(\mathbf{w}) P_j(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}})$$

with some polynomials $P_j(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}})$ over $\mathbb{R}$.

Obviously,

$$\tau \leq \binom{\deg(P) + k}{\deg(P)} \quad \text{and} \quad \tau \leq \text{size}\,(P)\,. \tag{3.4.6}$$

As in the proof of Lemma 3.4.2, we note that $p_1 < \ldots < p_\eta < \eta^2 \leq 2^k$. For every $i = 1, \ldots, \eta$, we add several leading zeros to the binary representation of p_i to obtain a binary string of length k. In this string we replace 0 by α_0 and 1 by α_1 and denote by $\mathbf{w}_i \in \{\alpha_0, \alpha_1\}^k$ this new vector.

If $\tau < \eta$, then there exist η real coefficients c_i, $i = 1, \ldots, \eta$, not all equal to zero, at least one of them negative, such that

$$\sum_{i=1}^{\eta} c_i \mu_j(\mathbf{w}_i) = 0, \qquad j = 1, \ldots, \tau.$$

Therefore, we have the identity:

$$\sum_{i=1}^{\eta} c_i P(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}}, [\mathbf{w}_i]_1, \ldots, [\mathbf{w}_i]_k) = 0.$$

One can easily verify that

$$2^{n-k} = \exp\left(5\eta \ln \eta + O(\eta)\right).$$

Hence, from Lemma 3.4.2 we derive that there exists x, $0 \leq x \leq 2^{n-k}$, such that:

$$c_i P(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}}, [\mathbf{w}_i]_1, \ldots, [\mathbf{w}_i]_k) > 0, \text{ for every } c_i < 0,$$

$$c_i P(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}}, [\mathbf{w}_i]_1, \ldots, [\mathbf{w}_i]_k) \geq 0, \text{ for every } c_i \geq 0,$$

where $x = x_1 \ldots x_{n-k}$ is the binary representation of x (with several leading zeros, if necessary, to make it of length $n-k$). Thus,

$$\sum_{i=1}^{\eta} c_i P(\alpha_{x_1}, \ldots, \alpha_{x_{n-k}}, [\mathbf{w}_i]_1, \ldots, [\mathbf{w}_i]_k) > 0.$$

From the obtained contradiction we see that $\tau \geq \eta \geq 2^{(k-1)/2}$ and as in the proof of Theorem 3.4.3 we obtain the desired result. □

3.4.7 Some Remarks and Other Properties

It is not hard to see that the constants in our estimates can be improved, for example, by using tighter estimates on the binomial bound and on the product of primes. On the other hand, we do not know how to obtain more substantial improvements of our lower bounds. In particular, they are exponentially weaker than those which are known for polynomials over $\mathbb{Z}_2$ (see [13]).

In addition, it would be very interesting to obtain analogues of the results of this section for other Boolean functions related to various number theoretic problems, such as the Boolean function deciding primality or the parity of the number of prime divisors of x. Unfortunately, even more advanced sieve techniques (see [19]) than those used in Lemma 3.4.2 are still not powerful enough to produce such results.

Finally, it would be very interesting to extend Theorem 3.4.4 to arbitrary composite moduli m.

Another important complexity characteristic of the square-free function can be obtained from quite simple considerations. Let us define the *additive complexity* $\mathcal{C}_{\pm}(P)$ of a polynomial P over $\mathbb{R}$ as the smallest number of '+' and '−' signs necessary to write down a polynomial. Obviously, for any univariate polynomial P

$$\mathcal{C}_{\pm}(P) \leq \text{size}\,(P) - 1 \leq \deg(P)$$

but neither size (P) nor $\deg(P)$ can be estimated in terms of $\mathcal{C}_{\pm}(P)$. However, it is shown in [43] that if a non-zero polynomial $P(X) \in \mathbb{R}[X]$ has at least N real zeros, then

$$\mathcal{C}_{\pm}(P) \geq \left(\frac{1}{5}\log N\right)^{1/2}.$$

The notion of additive complexity is related to the straight-line complexity of P (see [43]).

Now, let $P(x) \in \mathbb{R}(x)$ be such that

$$\operatorname{sign}(P(x)) = \mathrm{SF}(x), \qquad 0 \leq x \leq 2^n - 1.$$

If $4x+1$ is not a square-free number and $p > 1$ is a prime number such that $p^2 \mid (4x+1)$, then $p^2 \equiv 1 \pmod 4$ and $4x + 1 = (4q+1)p^2$ for some positive integer q. For a fixed prime p, there are at most $2^n/p^2 + 1$ integers q that satisfy the above condition. Hence, there are at most

$$\sum_{3 \leq p \leq (2^n-1)^{1/2}} \left(\frac{2^n}{p^2} + 1\right) \leq 2^{n-1}$$

non square-free numbers of the form $4x+1$. It follows then, that there is a constant $c > 0$ such that there are at least $c2^n$ square-free numbers of the form $4x+1$ and, thus, $P(4x)P(4x+1) \leq 0$ for them. Therefore, $P(x)$ has at least $c2^n$ zeros. This immediately provides the same bound on the degree of f and the lower bound

$$\mathcal{C}_{\pm}(P) \geq (0.2n)^{1/2} + O(1).$$

Chapter 4

Multiplicity Automata and Boolean Circuits

We develop a new method based on multiplicity automata to prove lower bounds for some circuit classes.

Using modular restriction techniques, Grolmusz & Tardos [24] characterized the functions computable by $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A$ circuits. Using the Degree-Decreasing Lemma and random restriction, they also showed a superpolynomial lower bound for $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A \circ \mathrm{AND}$ circuits computing the AND function of n Boolean variables, where the fan-in of the AND gates is $O(1)$ and the fan-in of the MOD_m^A gates is $o(n^2/\log n)$. Using communication matrices, Krause & Waack [30] showed an exponential lower bound for $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits computing the sequence identity function. Note that $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A \subseteq \mathrm{CC}^0[p^k] \circ \mathrm{SYM} \subseteq \mathrm{MOD}_p^{\{0\}} \circ \mathrm{SYM}$.

We show how to use multiplicity automata to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits. Based on the simulation of $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits by multiplicity automata, we simplify the complex technique used to prove the lower bound for $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits and prove exponential lower bounds for some other simple functions. We derive lower bounds for other circuit classes by exploiting their similarities to $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits. Using the properties of OR and AND gates, we show exponential lower bounds for $\mathrm{OR} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ and $\mathrm{AND} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits computing the sequence identity function and its complement respectively. By using the prime factorization of m to express the $\mathrm{MOD}_m^{\{0\}}$ gate as an AND of modular gates with prime power moduli, we obtain exponential lower bounds for $\mathrm{AND} \circ \mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits. Allowing the primes to grow logarithmically yields a lower bound of $2^{\Omega(n/\log n)}$ for $\mathrm{AND} \circ \{\mathrm{MOD}_m^{\{0\}}, \mathrm{EXACT}\} \circ \mathrm{SYM}$ circuits. We also show exponential lower bounds for $\mathrm{OR} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ and $\mathrm{AND} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ circuits computing the AND_n and OR_n functions, respectively, for any $\epsilon \leq p^{-4kd}$.

All our lower bounds remain the same even if $\mathrm{CC}^0[p^k]$ circuits are replaced by $\{\mathrm{CC}^0[p_i^{k_i}] : p_i, k_i \in O(1)\}$ circuits.

This chapter is organized as follows. In Section 4.1 we provide some background on multiplicity automata. Many researchers define the multiplicity automata model of computation as extended finite automata while others define it as the product of certain matrices. It is a well-known fact that the two definitions are equivalent. An important property of multiplicity automata that has been used in many papers (see [10]) relates the size of the smallest automaton computing some function f to the rank of the Hankel matrix of f. Using this relation, we show exponential lower bounds for multiplicity automata computing some simple functions. In [11] it was shown that multiplicity automata are closed under some common mathematical operations. These closure properties are fundamental for the simulation of a circuit class, which we study in Section 4.2.

In Section 4.2 we show how to use multiplicity automata to simulate symmetric gates where the inputs are Boolean variables and modular gates where the moduli are prime powers and the inputs are Boolean functions computable by multiplicity automata. Using these simulations and the closure properties of multiplicity automata, we show how to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits.

In Section 4.3 we show lower bounds for some circuit classes. We make use of the simulations of Section 4.2 and the relations amongst various circuit classes.

4.1 Multiplicity Automata

Multiplicity automata were successfully used in computational learning theory in the *exact* learning model of Angluin [5]. In this model, the task of the learner is to find a function h which agrees with the unknown target function f on all inputs. The learner is allowed to ask *membership* and *equivalence* queries. In a membership query, the learner asks for the value of the target function f at arguments x of its choice, and the teacher answers with $f(x)$ in one computational step. In an equivalence query, the learner suggests a hypothesis h. If the hypothesis h is equivalent to the target function f, the learning algorithm terminates successfully. Otherwise, the teacher provides a counterexample x, *i.e.,* $h(x) \neq f(x)$.

The learnability of multiplicity automata was first shown by Bergadano & Varricchio [12]. Subsequently, Beimel *et al.* [10] simplified the learning algorithm, reduced its complexity and used it to learn other complexity classes. In this chapter, we use multiplicity automata to derive lower bounds for some circuit classes. We now define the model of computation and important properties of multiplicity automata which we use as building blocks in our lower bound proofs.

4.1.1 Model of Computation

A multiplicity automaton is very similar to a finite automaton, the only differences being the multiplicities associated with each edge and accepting path, and the output associated with each input string. In [11], multiplicity automata are defined as extensions of finite automata. Since a finite automaton contains hard to control features such as the state transition, such definition is not always convenient for mathematical analysis. In [10], multiplicity automata are defined as products of certain matrices. It is a known fact that the two definitions are equivalent.

Multiplicity Automata as Finite Automata with Multiplicities

Let $\mathcal{K}$ be a field. A multiplicity automaton M over $\mathcal{K}$, for short $\mathcal{K}$-automaton, is a five-tuple $M = (\Sigma, Q, \delta, \mathrm{S}, \mathrm{F})$ where Σ is a finite alphabet, Q is the finite set of states, $\mathrm{S}, \mathrm{F} : Q \to \mathcal{K}$ are two mappings associated with the set of states, and $\delta : Q \times \Sigma \times Q \to \mathcal{K}$, the transition function, is a mapping that associates a multiplicity to each edge of the automaton. The multiplicity of a path is the product over the field $\mathcal{K}$ of the multiplicites of the edges of that path. The $\mathcal{K}$-automaton M computes a function which associates to each input the sum over the field $\mathcal{K}$ of the multiplicites of the accepting paths for the input. More precisely, let $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \Sigma^*$ be the input string. A path $\mathcal{P}$ for $\mathbf{x}$ is a sequence of edges

$$(q_1, x_1, q_2), (q_2, x_2, q_3), \ldots, (q_n, x_n, q_{n+1}),$$

where for every $1 \leq i \leq n$, there is an edge from state q_i to state q_{i+1} on alphabet symbol x_i. The multiplicity of the path is defined as

$$\Delta(\mathcal{P}) = \prod_{i=1}^{n} \delta(q_i, x_i, q_{i+1}).$$

Let $\mathcal{P}_s$ denote the first state of the path $\mathcal{P}$ and let $\mathcal{P}_f$ denote the last state of the path $\mathcal{P}$. The $\mathcal{K}$-automaton computes a function $f_M : \Sigma^* \to \mathcal{K}$ defined as

$$f_M(\mathbf{x}) = \sum_{\mathcal{P} \in \mathrm{Path}_M(\mathbf{x})} \mathrm{S}(\mathcal{P}_s)\Delta(\mathcal{P})\mathrm{F}(\mathcal{P}_f),$$

where $\mathrm{Path}_M(\mathbf{x})$ is the set of all possible paths of M on input $\mathbf{x} \in \Sigma^*$.

The size of the automaton is the number of states.

Multiplicity Automata as Product of Certain Matrices

We define a $\mathcal{K}$-automaton M of size ℓ as follows: let $\mu_\epsilon = \mathrm{I}$, where I is the $\ell \times \ell$ identity matrix over $\mathcal{K}$, let $\{\mu_\sigma : \sigma \in \Sigma\}$ be a set of $\ell \times \ell$ matrices over $\mathcal{K}$, let $\lambda = \langle \lambda_1, \ldots, \lambda_\ell \rangle \in \mathcal{K}^{1 \times \ell}$ be a row vector, and let $\gamma = \langle \gamma_1, \ldots, \gamma_\ell \rangle \in \mathcal{K}^{\ell \times 1}$ be a column vector. Define a mapping

μ which associates to each string from Σ^* an $\ell \times \ell$ matrix over $\mathcal{K}$ such that for every $\mathbf{x} = \langle \sigma_1, \ldots, \sigma_n \rangle \in \Sigma^*$,

$$\mu(\mathbf{x}) = \mu_{\sigma_1} \cdots \mu_{\sigma_n}.$$

The $\mathcal{K}$-automaton M computes a function $f_M : \Sigma^* \to \mathcal{K}$ such that for every $\mathbf{x} = \langle \sigma_1, \ldots, \sigma_n \rangle \in \Sigma^*$,

$$f_M(\mathbf{x}) = \lambda \mu(\mathbf{x}) \gamma.$$

Equivalency of the Definitions

The following theorem, which seems to be folklore, shows that the two definitions are equivalent.

Theorem 4.1.1 (Folklore)**.** *Let $\mathcal{K}$ be any field. Given a $\mathcal{K}$-automaton of size ℓ as in one definition we can construct an automaton of size ℓ as in the second definition such that the functions computed by the $\mathcal{K}$-automata are the same.*

Proof.

Direction I. Let M be a $\mathcal{K}$-automaton of size ℓ as in the first definition. We will construct a $\mathcal{K}$-automaton N as in the second definition such that for every $\mathbf{x} \in \Sigma^*$,

$$f_M(\mathbf{x}) = f_N(\mathbf{x}).$$

We now define the matrices μ_σ and the vectors λ and γ. For every $\sigma \in \Sigma$, every $1 \leq i, j \leq \ell$, if the $\mathcal{K}$-automaton M has an edge from state q_i to state q_j on the alphabet symbol σ, set the element $[\mu_\sigma]_{i,j}$ equal to the multiplicity of the edge, otherwise, set it equal to 0. For every $1 \leq i \leq \ell$, let $\lambda_i = \mathrm{S}(q_i)$ and $\gamma_i = \mathrm{F}(q_i)$. Then,

$$f_M(\mathbf{x}) \;=\; \sum_{\mathcal{P} \in \mathrm{Path}_M(\mathbf{x})} \mathrm{S}(\mathcal{P}_s) \Delta(\mathcal{P}) \mathrm{F}(\mathcal{P}_f) = \sum_{q_i, q_j \in Q} \lambda_i \left[\mu(\mathbf{x})\right]_{i,j} \gamma_j = \lambda \mu(\mathbf{x}) \gamma = f_N(\mathbf{x}),$$

where the second equality follows from the following lemma:

Lemma 4.1.2. *For every $\mathbf{x} \in \Sigma^*$, every $1 \leq i, j, \leq \ell$,*

$$\left[\mu(\mathbf{x})\right]_{i,j} = \sum_{\mathcal{P} \in Path_M^{i,j}(\mathbf{x})} \Delta(\mathcal{P}),$$

where $Path_M^{i,j}(\mathbf{x})$ is the set of all possible paths of the $\mathcal{K}$-automaton M on $\mathbf{x}$ starting with state q_i and ending with state q_j.

Proof. We proceed by induction on the length of the input. The base case $|\mathbf{x}| = 0$ is verified easily. For the induction step, let $\mathbf{x} = \mathbf{w} \circ \sigma$, where $\mathbf{w} \in \Sigma^n$ and $\sigma \in \Sigma$. Note that, $\mu(\mathbf{x}) = \mu(\mathbf{w})\mu_\sigma$. Hence,

$$\begin{aligned}
[\mu(\mathbf{x})]_{i,j} &= [\mu(\mathbf{w})]_{i,1}\,[\mu_\sigma]_{1,j} + \cdots + [\mu(\mathbf{w})]_{i,\ell}\,[\mu_\sigma]_{\ell,j} \\
&= \left[\sum_{\mathcal{P}\in \mathrm{Path}_M^{i,1}(\mathbf{w})} \Delta(\mathcal{P})\right][\mu_\sigma]_{1,j} + \cdots + \left[\sum_{\mathcal{P}\in \mathrm{Path}_M^{i,\ell}(\mathbf{w})} \Delta(\mathcal{P})\right][\mu_\sigma]_{\ell,j} \\
&= \sum_{\mathcal{P}\in \mathrm{Path}_M^{i,j}(\mathbf{x})} \Delta(\mathcal{P}),
\end{aligned}$$

where the second equality follows from the induction hypothesis. □

Direction II. Let N be a $\mathcal{K}$-automaton of size ℓ as in the second definition. We now define the $\mathcal{K}$-automaton $M = (Q, \Sigma, \delta, \mathrm{S}, \mathrm{F})$ as in the first definition that will compute the same function as N. The set of states is $Q = \{q_1, \ldots, q_\ell\}$. For every $\sigma \in \Sigma$, every $1 \le i, j \le \ell$, let $\delta(q_i, \sigma, q_j) = [\mu_\sigma]_{i,j}$. Finally, for every $1 \le i \le \ell$, let $\mathrm{S}(q_i) = \lambda_i$ and $\mathrm{F}(q_i) = \gamma_i$. Then, using Lemma 4.1.2, we obtain

$$\begin{aligned}
f_N(\mathbf{x}) &= \lambda\mu(\mathbf{x})\gamma = \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i\,[\mu(\mathbf{x})]_{i,j}\,\gamma_j = \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i \left[\sum_{\mathcal{P}\in \mathrm{Path}_M^{i,j}(\mathbf{x})} \Delta(\mathcal{P})\right]\gamma_j \\
&= \sum_{\mathcal{P}\in \mathrm{Path}_M(\mathbf{x})} \mathrm{S}(\mathcal{P}_s)\Delta(\mathcal{P})\mathrm{F}(\mathcal{P}_f) = f_M(\mathbf{x}).
\end{aligned}$$

□

4.1.2 Hankel Matrix

The Hankel matrix F of a function $f : \Sigma^* \to \mathcal{K}$ is an infinite matrix, whose rows are indexed by $\mathbf{x} \in \Sigma^*$ and columns by $\mathbf{y} \in \Sigma^*$. Each entry $(\mathbf{x}, \mathbf{y})$ contains the value $f(\mathbf{x} \circ \mathbf{y})$. Recall that, $F_{\mathbf{x},*}$ denotes the $\mathbf{x}$th row of F, $F_{*,\mathbf{y}}$ denotes the $\mathbf{y}$th column of F, and $F_{\mathbf{x},\mathbf{y}}$ denotes the $(\mathbf{x}, \mathbf{y})$ entry of F. The following theorem of [10] relates the size of the smallest $\mathcal{K}$-automaton computing some function f to the rank of the corresponding Hankel matrix F over the field $\mathcal{K}$, and, it was used to construct the learning algorithm for the multiplicity automata. This theorem is also fundamental to our lower bound proofs, and therefore we include its proof as it appears in [10] with some minor modifications to make it suitable to our purposes.

Theorem 4.1.3. *Let $f : \Sigma^* \to \mathcal{K}$ such that $f \not\equiv 0$ and let F be the corresponding Hankel matrix. Then, the size ℓ of the smallest $\mathcal{K}$-automaton M such that $f_M \equiv f$ is equal to rank(F).*

Proof. In [10], it is assumed that $\lambda = \langle 1, 0, \ldots, 0\rangle$ but with some modifications the proof of the theorem holds for any $\lambda = \langle \lambda_1, \ldots, \lambda_\ell\rangle \in \mathcal{K}^{1\times\ell}$.

Direction I. Let M be a $\mathcal{K}$-automaton of size ℓ. We prove that $\text{rank}(F) \leq \ell$. Let A be a semi-infinite matrix whose rows are indexed by strings in Σ^* and columns are indexed by $1, \ldots, \ell$. Set the $(\mathbf{x}, i)$ entry of A to $\lambda\mu(\mathbf{x})_{*,i}$. Similarly, let B be a semi-infinite matrix whose columns are indexed by strings in Σ^* and rows are indexed by $1, \ldots, \ell$. Set the $(i, \mathbf{y})$ entry of B to $\mu(\mathbf{y})_{i,*}\gamma$. Then,

$$\begin{aligned} F_{\mathbf{x},\mathbf{y}} &= f(\mathbf{x}\circ\mathbf{y}) = f_M(\mathbf{x}\circ\mathbf{y}) = \lambda\mu(\mathbf{x}\circ\mathbf{y})\gamma = [\lambda\mu(\mathbf{x})]\,[\mu(\mathbf{y})\gamma] \\ &= \sum_{i=1}^{\ell} \lambda\mu(\mathbf{x})_{*,i}\mu(\mathbf{y})_{i,*}\gamma = \sum_{i=1}^{\ell} A_{\mathbf{x},i}B_{i,\mathbf{y}} = A_{\mathbf{x},*}B_{*,\mathbf{y}}. \end{aligned}$$

Thus, $F = AB$, and by linear algebra, $\text{rank}(F) \leq \min\{\text{rank}(A), \text{rank}(B)\}$. Therefore, $\text{rank}(F) \leq \ell$.

Direction II. Let f be a function such that the corresponding Hankel matrix F has rank $\ell > 0$. We construct a $\mathcal{K}$-automaton M that computes the function f. Choose ℓ independent rows $F_{\mathbf{w}_1,*}, \ldots, F_{\mathbf{w}_\ell,*}$ of F indexed by strings $\mathbf{w}_1 = \epsilon, \mathbf{w}_2, \ldots \mathbf{w}_\ell$. Since f is not constantly 0, at least one entry of $F_{\epsilon,*}$ is different from 0, and therefore, $F_{\epsilon,*}$ can always be included in the basis of the row space of F. Let λ be a row vector of length ℓ such that $\lambda = \langle 1, 0, \ldots, 0\rangle$ and let γ be a column vector of length ℓ such that $\gamma = \langle f(\mathbf{w}_1), \ldots, f(\mathbf{w}_\ell)\rangle$. To define the entries of μ_σ matrices, we use the fact that every row of F can be expressed as a linear combination of the rows that form the basis of its row space. Thus, for every σ, define the ith row of the matrix μ_σ as the coefficients of the row $F_{\mathbf{w}_i\circ\sigma,*}$ when expressed as a linear combination of $F_{\mathbf{w}_1,*}, \ldots, F_{\mathbf{w}_\ell,*}$, *i.e.*,

$$F_{\mathbf{w}_i\circ\sigma,*} = \sum_{j=1}^{\ell} [\mu_\sigma]_{i,j}\ F_{\mathbf{w}_j,*}. \tag{4.1.1}$$

We proceed by induction on the length of the input string $\mathbf{x}$ to show that $f(\mathbf{w}_i \circ \mathbf{x}) = [\mu(\mathbf{x})]_{i,*}\ \gamma$ for all i. This suffices to show that M computes f since $f(\mathbf{x}) = f(\mathbf{w}_1 \circ \mathbf{x}) = [\mu(\mathbf{x})]_{1,*}\ \gamma = \lambda\ \mu(\mathbf{x})\ \gamma = f_M(\mathbf{x})$. In the base case, $\mathbf{x} = \epsilon$, and so $f_M(\epsilon) = \lambda\ \mathrm{I}\ \gamma = \gamma_1 = f(\mathbf{w}_1) = f(\epsilon)$. For the induction step, using Eq. 4.1.1, we observe that,

$$f(\mathbf{w}_i \circ \sigma \circ \mathbf{x}) = F_{\mathbf{w}_i\circ\sigma,\mathbf{x}} = \sum_{j=1}^{\ell} [\mu_\sigma]_{i,j}\ F_{\mathbf{w}_j,\mathbf{x}} = \sum_{j=1}^{\ell} [\mu_\sigma]_{i,j}\ f(\mathbf{w}_j \circ \mathbf{x}).$$

From the induction hypothesis $f(\mathbf{w}_j \circ \mathbf{x}) = [\mu(\mathbf{x})]_{j,*}\ \gamma$, and thus,

$$\sum_{j=1}^{\ell} [\mu_\sigma]_{i,j}\ \ [\mu(\mathbf{x})]_{j,*}\ \gamma = [\mu(\sigma)\ \mu(\mathbf{x})]_{i,*}\ \ \gamma = [\mu(\sigma \circ \mathbf{x})]_{i,*}\ \ \gamma.$$

□

4.1.3 Functions $f : \Sigma^n \to \mathcal{K}$

In this thesis, we are interested in computing functions whose input has a fixed length n for some value n. However, Theorem 4.1.3 holds only for functions with domain Σ^*. Therefore, if a function f is defined on Σ^n, we view it as a function defined on Σ^*, where f is 0 for all strings whose length is not equal to n. Thus, if $\mathbf{x}$ is a string of length k, $0 \leq k \leq n$, then the entry $(\mathbf{x}, \mathbf{y})$ of the corresponding Hankel matrix F is 0 for all strings $\mathbf{y}$ whose length is not equal to $n-k$. If we denote by F^k the submatrix of F whose rows are indexed by strings of length k, and columns by strings of length $n-k$, then

$$\operatorname{rank}(F) = \sum_{k=0}^{n} \operatorname{rank}(F^k). \tag{4.1.2}$$

4.1.4 Some Simple Functions with Exponential Rank

We now show that the Hankel matrices of some simple Boolean functions have exponential ranks. It is immediate from Theorem 4.1.3 that the smallest $\mathcal{K}$-automata computing such function have exponential size as well. These are also the functions which we will use to show lower bounds for different circuit classes.

Theorem 4.1.4. *Let AndOr, OrAnd, ID, $\neg$ID : $\{0,1\}^{2n} \to \{0,1\}$ be defined as*

(a) $AndOr(\mathbf{x}) = AND\left(OR\left(x_1, x_{n+1}\right), \ldots, OR\left(x_n, x_{2n}\right)\right)$

(b) $OrAnd(\mathbf{x}) = OR\left(AND\left(x_1, x_{n+1}\right), \ldots, AND\left(x_n, x_{2n}\right)\right)$

(c) $ID(\mathbf{x}) = AND\left(MOD_2^{\{0\}}\left(x_1, x_{n+1}\right), \ldots, MOD_2^{\{0\}}\left(x_n, x_{2n}\right)\right)$

(d) $\neg ID(\mathbf{x}) = OR\left(MOD_2^{\{1\}}\left(x_1, x_{n+1}\right), \ldots, MOD_2^{\{1\}}\left(x_n, x_{2n}\right)\right)$

Then, for every field $\mathcal{K}$, the size of the smallest $\mathcal{K}$-automaton computing one of the functions AndOr, OrAnd, ID, $\neg$ID is at least $2^n - 1$.

Proof. From Theorem 4.1.3, the size of the smallest $\mathcal{K}$-automaton computing a function is equal to the rank of the corresponding Hankel matrix. Therefore, we show that the ranks of the Hankel matrices of the above functions are all exponential.

(a) Let F be the Hankel matrix of AndOr. Clearly, $\text{rank}(F) \geq \text{rank}(F^n)$, where F^n is the submatrix of F whose rows and columns are indexed by strings of length n. Let $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$, $\mathbf{b} = \langle b_1, \ldots, b_n \rangle \in \{0,1\}^n$ and consider the $(\mathbf{a}, \mathbf{b})$ entry of F^n. If $a_n = b_n = 0$, then the entry is zero, otherwise it is equal to the $(\mathbf{a}', \mathbf{b}')$ entry of F^{n-1}, where $\mathbf{a}' = \langle a_1, \ldots, a_{n-1} \rangle$ and $\mathbf{b}' = \langle b_1, \ldots, b_{n-1} \rangle$. Thus,

$$F^1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad F^n = \begin{pmatrix} 0 & F^{n-1} \\ F^{n-1} & F^{n-1} \end{pmatrix}.$$

Therefore, $\text{rank}(F^1) = 2$ and $\text{rank}(F^n) = 2\,\text{rank}(F^{n-1})$. By induction on n it follows that, $\text{rank}(F^n) = 2^n$.

(b) Consider the complement of OrAnd and follow the proof of part (a). A complete proof appears in [10].

(c) Note that the F^n submatrix is equal to the $2^n \times 2^n$ identity matrix.

(d) Note that the F^n submatrix is equal to the complement of the $2^n \times 2^n$ identity matrix. □

Theorem 4.1.5. *Let p be an odd prime. Define the functions $Mod_2And, Mod_2Or : \{0,1\}^{2n} \to \{0,1\}$ as*

(a) $Mod_2And(\mathbf{x}) = MOD_2^{\{0\}}\left(AND\left(x_1, x_{n+1}\right), \ldots, AND\left(x_n, x_{2n}\right)\right)$

(b) $Mod_2Or(\mathbf{x}) = MOD_2^{\{0\}}\left(OR\left(x_1, x_{n+1}\right), \ldots, OR\left(x_n, x_{2n}\right)\right)$

Then, the size of the smallest $\mathbb{Z}_p$-automaton computing one of the functions Mod_2And, Mod_2Or is at least 2^{n-1}.

Proof. In this proof, every mathematical operation is over $\mathbb{Z}_p$.

(a) Recall that F^n is the submatrix of F whose rows and columns are indexed by strings of length n. Since $\text{rank}(F) \geq \text{rank}(F^n)$, it suffices to show that $\text{rank}(F^n) = 2^n$. Noting that $\text{rank}(F^1) = 2$, the claim follows if we can show that

$$\text{rank}(F^n) = 2\,\text{rank}(F^{n-1})$$

for every $n > 1$. Consider an arbitrary entry $(\mathbf{x}, \mathbf{y})$ of the submatrix F^n, where $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$, $\mathbf{y} = \langle y_1, \ldots, y_n \rangle \in \{0,1\}^n$. Let $\mathbf{x}' = \langle x_1, \ldots, x_{n-1} \rangle$ and $\mathbf{y}' = \langle y_1, \ldots, y_{n-1} \rangle$. Clearly, if $x_n y_n = 0$, then $F^n(\mathbf{x}, \mathbf{y}) = F^{n-1}(\mathbf{x}', \mathbf{y}')$. Also, if $x_n y_n = 1$, then $F^n(\mathbf{x}, \mathbf{y}) = 1 - F^{n-1}(\mathbf{x}', \mathbf{y}')$. Therefore,

$$F^n = \begin{pmatrix} F^{n-1} & F^{n-1} \\ F^{n-1} & J^{n-1} - F^{n-1} \end{pmatrix},$$

where J^{n-1} is the all-1 matrix whose rows and columns are indexed by strings of length $n-1$.

Let $\ell = \text{rank}(F^{n-1})$ and let $\mathbf{w}_1, \ldots, \mathbf{w}_\ell$ be the row basis of F^{n-1}. Denote by $\mathbf{x}_i$ the binary vector of length $n-1$ that is the row index of $\mathbf{w}_i$. For $i = 1, \ldots, \ell$, let $\mathbf{v}_i$ denote the row of F^n whose index is the binary vector $0 \circ \mathbf{x}_i$ of length n, and let $\mathbf{v}_{\ell+i}$ denote the row of F^n whose index is the binary vector $1 \circ \mathbf{x}_i$ of length n. It is easy to see that,

$$\mathbf{v}_i = \begin{cases} \mathbf{w}_i \circ \mathbf{w}_i, & \text{if } 1 \leq i \leq \ell, \\ \mathbf{w}_{i-\ell} \circ (\mathbf{1}_\ell - \mathbf{w}_{i-\ell}), & \text{if } \ell < i \leq 2\ell, \end{cases}$$

where $\mathbf{1}_\ell$ is the all-1 binary vector of length ℓ.

Suppose

$$c_1\,\mathbf{v}_1 + \cdots + c_{2\ell}\,\mathbf{v}_{2\ell} = \mathbf{0}_{2\ell},$$

for some $c_1, \ldots, c_{2\ell} \in \{0, 1, \ldots, p-1\}$. Then,

$$c_1\,\mathbf{w}_1 + \cdots + c_\ell\,\mathbf{w}_\ell + c_{\ell+1}\,\mathbf{w}_1 + \cdots + c_{2\ell}\,\mathbf{w}_\ell = \mathbf{0}_\ell, \tag{4.1.3}$$

and

$$c_1\,\mathbf{w}_1 + \cdots + c_\ell\,\mathbf{w}_\ell - c_{\ell+1}\,\mathbf{w}_1 - \cdots - c_{2\ell}\,\mathbf{w}_\ell + (c_{\ell+1} + \cdots + c_{2\ell})\,\mathbf{1}_\ell = \mathbf{0}_\ell. \tag{4.1.4}$$

Since $\mathbf{w}_1, \ldots, \mathbf{w}_\ell$ are linearly independent and from Eq. 4.1.3, it follows that $c_i + c_{\ell+i} = 0$, $i = 1, \ldots, \ell$. Therefore, Eq. 4.1.4 is equivalent to

$$2c_1\,\mathbf{w}_1 + \cdots + 2c_\ell\,\mathbf{w}_\ell - (c_1 + \cdots + c_\ell)\,\mathbf{1}_\ell = \mathbf{0}_\ell. \tag{4.1.5}$$

The first entry of each $\mathbf{w}_i$ is equal to 1, since $F^{n-1}(\mathbf{a}, \mathbf{0}_\ell) = 1$ for every binary string $\mathbf{a}$ of length $n-1$. From the above observation and Eq. 4.1.5, we see that $2c_1 + \cdots + 2c_\ell - (c_1 + \cdots + c_\ell) = 0$, and thus,

$$c_1 + \cdots + c_\ell = 0.$$

Hence, Eq 4.1.5 is equivalent to

$$2c_1\,\mathbf{w}_1 + \cdots + 2c_\ell\,\mathbf{w}_\ell = \mathbf{0}_\ell. \tag{4.1.6}$$

Since $\mathbf{w}_1, \ldots, \mathbf{w}_\ell$ are linearly independent and noting that 2 has an inverse in $\mathbb{Z}_p$, it follows that $c_i = 0$, $i = 1, \ldots, \ell$. We showed that $c_i + c_{\ell+i} = 0$, $i = 1, \ldots, \ell$, therefore,

$$c_i = 0, \qquad i = 1, \ldots, 2\ell.$$

Thus, $\mathbf{v}_1, \ldots, \mathbf{v}_{2\ell}$ are linearly independent.

(b) Proceeding as in (a), we see that,

$$F^n = \begin{pmatrix} F^{n-1} & J^{n-1} - F^{n-1} \\ J^{n-1} - F^{n-1} & J^{n-1} - F^{n-1} \end{pmatrix},$$

and

$$\operatorname{rank}(F^n) = 2\operatorname{rank}(J^{n-1} - F^{n-1}).$$

Note that, $\operatorname{rank}(J^{n-1} - F^{n-1}) \geq \operatorname{rank}(F^{n-1}) - 1$ and $\operatorname{rank}(F^2) = 4$. Therefore,

$$\operatorname{rank}(F^n) = 2\left(\operatorname{rank}(F^{n-1}) - 1\right) \geq \cdots \geq 2^{n-2}\left(\operatorname{rank}(F^2) - 2\right) + 2 = 2^{n-1} + 2.$$

□

4.1.5 Closure Properties

In [11] it is shown that multiplicity automata are closed under the operations of multiplication, addition, and scalar multiplication. Since every polynomial over some field $\mathcal{K}$ can be constructed using these operations, it is not difficult to see how to use a $\mathcal{K}$-automaton to compute the value of a polynomial. Although simple, we will see in Section 4.2 that such properties are very important for the simulation of some circuit classes.

Theorem 4.1.6 (Bergadano *et al.* [11])**.** *Let $\mathcal{K}$ be any field. Let $M_1 = (Q_1, \Sigma, \delta_1, S_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, S_2, F_2)$ be $\mathcal{K}$-automata. Then, for each of the following, there exists a $\mathcal{K}$-automaton $M = (Q, \Sigma, \delta, S, F)$ satisfying*

(a) $f_M \equiv f_{M_1} f_{M_2}$ *and* $size(M) = size(M_1)size(M_2)$.

(b) $f_M \equiv f_{M_1} + f_{M_2}$ *and* $size(M) = size(M_1) + size(M_2)$.

(c) $f_M \equiv \alpha f_{M_1}$, *where* $\alpha \in \mathcal{K}$ *and* $size(M) = size(M_1)$.

Proof.

(a) Let $Q = Q_1 \times Q_2$. For every $(q_1, q_2) \in Q$, let $\mathrm{S}((q_1, q_2)) = \mathrm{S}_1(q_1)\mathrm{S}_2(q_2)$ and $\mathrm{F}((q_1, q_2)) = \mathrm{F}_1(q_1)\mathrm{F}_2(q_2)$. Finally, for every $\sigma \in \Sigma$, every $(q_1, q_2), (p_1, p_2) \in Q$, let $\delta((q_1, q_2), \sigma, (p_1, p_2)) = \delta_1(q_1, \sigma, p_1)\delta_2(q_2, \sigma, p_2)$.

(b) Let $Q = Q_1 \cup Q_2$. For every $q \in Q$, let $\mathrm{S}(q) = \mathrm{S}_1(q)\,[q \in Q_1] + \mathrm{S}_2(q)\,[q \in Q_2]$ and $\mathrm{F}(q) = \mathrm{F}_1(q)\,[q \in Q_1] + \mathrm{F}_2(q)\,[q \in Q_2]$. Finally, for every $\sigma \in \Sigma$, every $q, p \in Q$, let $\delta(q, \sigma, p) = \delta_1(q, \sigma, p)\,[q, p \in Q_1] + \delta_2(q, \sigma, p)\,[q, p \in Q_2]$.

(c) Let $Q = Q_1$. For every $q \in Q$, let $\mathrm{S}(q) = \lambda \mathrm{S}_1(q)$ and $\mathrm{F}(q) = \mathrm{F}_1(q)$. Finally, for every $\sigma \in \Sigma$, every $q, p \in Q$, let $\delta(q, \sigma, p) = \delta_1(q, \sigma, p)$.

□

It is a known fact that every Boolean function of Boolean variables can be expressed in disjunctive form, *i.e.,* as an OR of ANDs, or as a multilinear polynomial. This fact was used in [11] to compute arbitrary functions $f : \{0,1\}^n \to \mathcal{K}$ by $\mathcal{K}$-automata.

Theorem 4.1.7 (Bergadano *et al.* [11])**.** *Let $\mathcal{K}$ be any field. Let $M_1, M_2, \ldots, M_n$ be $\mathcal{K}$-automata each of size at most s. Then, for any function f over $\mathcal{K}$ on n inputs, $f(f_{M_1}, \ldots, f_{M_n})$ can be computed by a $\mathcal{K}$-automaton of size at most $2^n s^n$.*

Proof. Let $\mathbf{x} \in \Sigma^*$ be the input string to the $\mathcal{K}$-automata $M_1, \ldots, M_n$. For $1 \le i \le n$, let $\nu_i = f_{M_i}(\mathbf{x})$. Suppose the value of f on input $\nu_1, \ldots, \nu_n$ is ν. Then,

$$f(\nu_1, \ldots, \nu_n) = \nu\mu_1 \cdots \mu_n,$$

where for $1 \le i \le n$, $\mu_i = \nu_i$ if $\nu_i = 1$ and $\mu_i = 1 - \nu_i$ if $\nu_i = 0$. Therefore, $f(f_{M_1}, \ldots, f_{M_n})$ can be written as a multilinear polynomial, *i.e.,*

$$f(f_{M_1}, \ldots, f_{M_n}) = \sum_{H \in \mathcal{H}} c_H \prod_{i \in H} f_{M_i},$$

where $\mathcal{H} \subseteq \mathcal{P}(\{1, \ldots, n\})$ and $c_H \in \mathcal{K}$. Using the constructions described in Theorem 4.1.6, parts (a) and (c), we obtain a $\mathcal{K}$-automaton of size $s^{|H|} \le s^n$ which computes $c_H \prod_{i \in H} f_{M_i}$. Finally, summing all the terms of the polynomial gives a $\mathcal{K}$-automaton of size $|\mathcal{H}|\, s^n \le 2^n s^n$ which computes f. □

4.2 Simulating Circuits by Multiplicity Automata

In this section, we show how to simulate various gates and circuit classes by multiplicity automata. Our constructions are based on the closure properties of multiplicity automata (Subsection 4.1.5), the fact that multiplicity automata can behave as finite automata, and the properties of various gates (Section 2.2).

4.2.1 Simulating Symmetric Gates

We show how to simulate a symmetric gate when the inputs are Boolean variables.

Lemma 4.2.1. *Let $\mathcal{K}$ be any field. The function $SYM^A : \{0,1\}^n \to \{0,1\}$ is computable by a $\mathcal{K}$-automaton of size $(n+1)^2$.*

Proof. Denote the input string to the symmetric gate by $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$. However, it might be the case that only a subset of the input variables is actually connected to the gate — denote the indices of these input variables by $\mathcal{J}$, *i.e.,* $\mathcal{J} = \{j : x_j \text{ or } \neg x_j \text{ is connected to the gate}\}$. We construct a $\mathcal{K}$-automaton $M = (Q, \Sigma, \delta, \mathrm{S}, \mathrm{F})$ to simulate the SYM^A gate as follows: the input alphabet is $\Sigma = \{0,1\}$, the set of states is $Q = \{q_{j,\ell} : j \in [0,n], \ell \in [0,n]\}$. For each $j \in [0, n-1]$ and $\ell \in [0,n]$, the automaton M, on reading 0, goes from state $q_{j,\ell}$ to state $q_{j+1,\ell}$ and on reading 1, M goes from state $q_{j,\ell}$ to state $q_{j+1,\ell}$ if $j+1 \notin \mathcal{J}$ and it goes to state $q_{j+1,\ell+1}$ if $j+1 \in \mathcal{J}$. Each of the above edges is assigned a multiplicity of 1. Finally,

$$\mathrm{S}(q_{j,\ell}) = \begin{cases} 1, & \text{if } j = 0 \text{ and } \ell = 0, \\ 0, & \text{otherwise.} \end{cases} \quad \text{and} \quad \mathrm{F}(q_{j,\ell}) = \begin{cases} 1, & \text{if } j = n \text{ and } \ell \in A, \\ 0, & \text{otherwise.} \end{cases}$$

□

4.2.2 Simulating $\mathrm{MOD}_{p^k}^A$ Gates

In [18], it is shown how to construct $\mathbb{Z}_p$-automata to simulate $\mathrm{MOD}_{p^k}^{\{0\}}$ gates with Boolean functions computable by automata as inputs. In fact, we can simulate any $\mathrm{MOD}_{p^k}^A$ gate by expressing it as an OR of $\mathrm{MOD}_{p^k}^{\{0\}}$ gates (see Section 2.2).

Lemma 4.2.2. *Let p be a prime. Let $c_1, \ldots, c_\upsilon$ be Boolean functions that can be computed by $\mathbb{Z}_p$-automata of size at most s. Then, for any $k \in \mathbb{N}$ and $A \subseteq [0, p-1]$, the Boolean function $MOD^A_{p^k}(c_1, \ldots, c_\upsilon)$ can be computed by a $\mathbb{Z}_p$-automaton of size at most $\left(\left(\upsilon + p^k\right) s\right)^{p^{2k}}$.*

Proof. As in Lemma 2.2.1, for every $a \in A$, let $\mathbf{y}_a$ be a $(\upsilon + p^k)$-tuple such that $[y_a]_1 = c_1, \cdots, [y_a]_\upsilon = c_\upsilon; [y_a]_{\upsilon+1} = \cdots = [y_a]_{\upsilon+p^k-a} = 1$; and $[y_a]_{\upsilon+p^k-a+1} = \cdots = [y_a]_{\upsilon+p^k} = 0$. In Lemma 2.2.1, we showed that,

$$\mathrm{MOD}^A_{p^k}(c_1, \ldots, c_\upsilon) \;=\; 1 - \prod_{a \in A} \left(1 - \mathrm{MOD}^{\{0\}}_{p^k}([y_a]_1, \ldots, [y_a]_{\upsilon+p^k})\right),$$

and that, for every $a \in A$,

$$\mathrm{MOD}^{\{0\}}_{p^k}([y_a]_1, \ldots, [y_a]_{\upsilon+p^k}) = 1$$

$$\iff \prod_{i=0}^{k-1} \left(1 - \left(\sum_{\substack{S \subseteq [1, \upsilon+p^k] \\ |S| = p^i}} \prod_{j \in S} [y_a]_j\right)^{p-1}\right) \equiv 1 \pmod{p}.$$

From the closure properties of multiplicity automata (Theorem 4.1.6), and noting that each y_j is computed by a $\mathbb{Z}_p$-automata of size at most s, then there exists a $\mathbb{Z}_p$-automaton of size at most $s^{|S|}$ that computes $\prod_{j \in S}[y_a]_j$. Thus, the sum is computed by a $\mathbb{Z}_p$-automaton of size at most $\binom{\upsilon+p^k}{p^i} s^{p^i} \leq \left((\upsilon + p^k)s\right)^{p^i}$. The ith term in the product is computed by a $\mathbb{Z}_p$-automaton of size at most $\left((\upsilon + p^k)s\right)^{p^i(p-1)}$, and as a result, the final product is computable by a $\mathbb{Z}_p$-automatonof size at most $\prod_{i=0}^{k-1} \left((\upsilon + p^k)s\right)^{p^i(p-1)} \leq \left((\upsilon + p^k)s\right)^{p^k}$. Therefore, the function $\mathrm{MOD}^A_{p^k}(c_1, \ldots, c_\upsilon)$ is computed by a $\mathbb{Z}_p$-automaton of size $\left((\upsilon + p^k)s\right)^{|A|p^k} \leq \left((\upsilon + p^k)s\right)^{p^{2k}}$. □

4.2.3 Simulating $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ Circuits

We now show how to construct a multiplicity automata over $\mathbb{Z}_p$ to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits. This result also follows from Corollary 2.5.3 and the work of [11], where it is shown how to simulate $\mathrm{MOD}^{\{0\}}_p \circ \mathrm{TC}^0_1$ circuits by $\mathbb{Z}_p$-automata. As a corollary, the class of $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits is learnable under the exact learning model. Our construction is based on the simulation of individual gates from the previous two subsections and extends the proof of [18] where it is shown how to simulate constant-depth circuits with $\mathrm{MOD}^{\{0\}}_{p^k}$ gates by $\mathbb{Z}_p$-automata.

Theorem 4.2.3. *Let C be a $CC^0[p^k] \circ SYM$ circuit computing a Boolean function $f : \{0,1\}^n \to \{0,1\}$, where p is a fixed prime and k is a fixed integer. Then, there exists a $\mathbb{Z}_p$-automaton M, such that*

$$size(M) = (size(C))^{O(1)},$$

and for every $\mathbf{x} \in \{0,1\}^n$,

$$f_M(\mathbf{x}) = f(\mathbf{x}).$$

Proof. Since the field is explicit, we will refer to $\mathbb{Z}_p$-automata as simply automata. For each gate at the input level we construct an automaton as explained in Lemma 4.2.1 and let N be the size of the largest automaton. We claim inductively that if C has depth $d+1$, then it is computable by an automaton of size at most

$$N\big((\upsilon + p^k)N\big)^{\sum_{j=1}^{d+1} p^{2jk}},$$

where υ is the largest fan-in of any gate in C. The base case, $d = 1$, follows immediately from Lemma 4.2.1 since the circuit is a single symmetric gate. For the inductive step, let the output gate of C be a $\mathrm{MOD}^A_{p^k}(c_1, \ldots, c_\alpha)$ gate for some $A \subseteq [0, p-1]$. From the induction hypothesis, each c_i is computed by an automaton of size at most $S = N\big((\upsilon + p^k)N\big)^{\sum_{j=1}^{d} p^{2jk}}$. From Lemma 4.2.2, we see that C is computable by an automaton of size at most

$$\begin{aligned}
((\alpha + p^k)S)^{p^{2k}} &\leq ((\upsilon + p^k)S)^{p^{2k}} \\
&\leq (\upsilon + p^k)^{p^{2k}} N^{p^{2k}} \big((\upsilon + p^k)N\big)^{p^{2k} \sum_{j=1}^{d} p^{2jk}} \\
&\leq N\big((\upsilon + p^k)N\big)^{\sum_{j=1}^{d+1} p^{2jk}}.
\end{aligned}$$

Since p, k, d are constants and $N \in n^{O(1)}$, then C is computable by an automaton of size at most $(\mathrm{size}(C))^{O(1)}$. □

We remark that Theorem 4.2.3 can be proven by using polynomials, as in Corollary 2.2.2, to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits by $\mathrm{MOD}_p^{\{0\}} \circ \mathrm{AND}_{O(1)}$ circuits. We provided a longer proof because a precise bound on the size of the $\mathbb{Z}_p$-automaton is necessary for some of the lower bound proofs of the next section.

4.3 Lower Bounds

We are now ready to prove lower bounds for several circuit classes. In Subsection 4.1.4 we showed that $\mathcal{K}$-automata require exponential size to compute some simple Boolean functions.

In Section 4.2 we showed how to simulate $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits by $\mathbb{Z}_p$-automata where the size of the automata increases only by a polynomial factor. Immediately, this implies an exponential lower bound for $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits. The other lower bounds are obtained by relating the circuit classes to $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits.

4.3.1 $\mathbf{CC}^0[p^k] \circ \mathbf{SYM}$ Circuits

Theorem 4.3.1. *If $C \in CC^0[p^k] \circ SYM$ computes any of the functions ID, $\neg$ID, AndOr, OrAnd, Mod_2And, Mod_2Or (defined in Theorems 4.1.4, 4.1.5), then*

$$size(C) = 2^{\Omega(n)}.$$

Proof. We construct a $\mathbb{Z}_p$-automaton to simulate C as described in Theorem 4.2.3. Then, the lower bound follows since the size of the automaton simulating C is $(\mathrm{size}(C))^{O(1)}$ and any $\mathbb{Z}_p$-automaton computing any of the functions ID, $\neg$ID, AndOr, OrAnd, Mod_2And, Mod_2Or must have size $2^{\Omega(n)}$ (Theorems 4.1.4, 4.1.5). □

4.3.2 $\mathbf{AC}^0_1 \circ \mathbf{CC}^0[p^k] \circ \mathbf{SYM}$ Circuits

We prove the lower bound for $\mathrm{AC}^0_1 \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ circuits by showing that one of the subcircuits $\mathrm{CC}^0[p^k] \circ \mathrm{SYM}$ connected to the output gate computes a similar function to that computed by the entire circuit. There are two cases two consider depending on whether the output gate is an OR gate or an AND gate.

Theorem 4.3.2. *If $C \in OR \circ CC^0[p^k] \circ SYM$ computes the ID function, then*

$$size(C_{2n}) = 2^{\Omega(n)}.$$

Proof. Let $c_1, \ldots, c_t$ be the subcircuits connected to the OR gate. Denote by $C_{2n}(\mathbf{x})$ and $c_i(\mathbf{x})$ the values of the circuit C_{2n} and subcircuit c_i respectively when the input is $\mathbf{x} \in \{0,1\}^{2n}$. By the definition of the OR gate, there exist a set $S \subseteq \{0,1\}^{2n}$ of cardinality at least $\frac{2^n}{t}$ and an integer i, $1 \leq i \leq t$, such that,

$$\forall \mathbf{x} \in \{0,1\}^{2n} : C_{2n}(\mathbf{x}) = 0 \Rightarrow c_i(\mathbf{x}) = 0 \text{ and } \forall \mathbf{x} \in S : C_{2n}(\mathbf{x}) = 1 \Rightarrow c_i(\mathbf{x}) = 1. \quad (4.3.1)$$

Let F be the Hankel matrix of ID and let F^n be the submatrix of F whose rows and columns are indexed by strings of length n. It is easily seen that F^n equals the $2^n \times 2^n$ identity matrix

I_{2^n}. Let H be the Hankel matrix of the function computed by subcircuit c_i and let H^n be the submatrix of H whose rows and columns are indexed by strings of length n. It follows from Eq. 4.3.1 that the main diagonal of H^n contains at least $\frac{2^n}{t}$ entries equal to 1, and all other entries of H^n are equal to 0. From Eq. 4.1.2, we see that,

$$\text{rank}(H) \geq \text{rank}(H^n) \geq \frac{2^n}{t}. \tag{4.3.2}$$

We note that the rank is over some field $\mathcal{K}$ and that Eq. 4.3.2 holds for any field $\mathcal{K}$, and in particular for $\mathcal{K} = \mathbb{Z}_p$. Since c_i is a $\text{CC}^0[p^k] \circ \text{SYM}$ circuit, from Theorem 4.2.3, there exists a $\mathbb{Z}_p$-automaton M that computes the same function as c_i, and furthermore

$$\text{size}(M) = (\text{size}(c_i))^{O(1)}. \tag{4.3.3}$$

From Theorem 4.1.3, the size of the smallest such automaton is equal to rank(H). Hence, from Eqs. 4.3.2 and 4.3.3, it follows that for some constant $c > 0$,

$$\text{size}(c_i) \geq \left(\frac{2^n}{t}\right)^c.$$

□

Similarly to the case of $\text{OR} \circ \text{CC}^0[p^k] \circ \text{SYM}$ circuits, we show a lower bound for $\text{AND} \circ \text{CC}^0[p^k] \circ \text{SYM}$ circuits.

Theorem 4.3.3. *If $C \in AND \circ CC^0[p^k] \circ SYM$ computes the $\neg$ID function, then*

$$size(C_{2n}) = 2^{\Omega(n)}.$$

Proof. The proof is very similar to that of Theorem 4.3.2. Let t be the fan-in of the AND gate. We simply need to note that, by the definition of the AND gate, there exist a set $S \subseteq \{0,1\}^{2n}$ of cardinality at least $\frac{2^n}{t}$ and an integer i, $1 \leq i \leq t$, such that,

$$\forall \mathbf{x} \in \{0,1\}^{2n} :\ C_{2n}(\mathbf{x}) = 1 \Rightarrow c_i(\mathbf{x}) = 1 \qquad \text{and} \quad \forall \mathbf{x} \in S :\ C_{2n}(\mathbf{x}) = 0 \Rightarrow c_i(\mathbf{x}) = 0.$$

Also, F^n is the complement of the identity matrix I_{2^n}, and over any field, $\text{rank}(I_{2^n}) \geq 2^n - 1$.
□

4.3.3 AND $\circ$ {MOD$_m^{\{0\}}$, EXACT} $\circ$ SYM Circuits

We show an exponential lower bound for $\text{AND} \circ \text{MOD}_m^{\{0\}} \circ \text{SYM}$ circuits by expressing each modular gate with composite moduli as an AND of modular gates with prime power moduli. By allowing these primes to grow logarithmically, we can also express the exact gates as an AND of modular gates. As a result, we can show a $2^{\Omega(n/\log n)}$ lower bound for $\text{AND} \circ \{\text{MOD}_m^{\{0\}}, \text{EXACT}\} \circ \text{SYM}$ circuits.

Theorem 4.3.4. *If $C \in AND \circ MOD_m^{\{0\}} \circ SYM$ computes the $\neg$ID function, then*

$$size(C_{2n}) = 2^{\Omega(n)}.$$

Proof. Let $c_1, \ldots, c_t$ be the subcircuits connected to the AND gate and let F be the Hankel matrix of the function $\neg$ID. It is easily seen that F^n equals the complement of the identity matrix I_{2^n}. As in Theorem 4.3.3, there exists an integer i, $1 \leq i \leq t$, such that the main diagonal of the matrix H^n contains at least $\frac{2^n}{t}$ entries equal to 0, and all other entries of H^n are equal to 1, where H is the Hankel matrix of the function computed by subcircuit c_i.

The output gate of c_i is a modular gate, *i.e.,* $\mathrm{MOD}_m^{\{0\}}$ for some integer m. Let $m = p_1^{\alpha_1} \cdots p_\tau^{\alpha_\tau}$ be given by its prime factorization. Then, for any integer s,

$$s \equiv 0 \pmod{m} \iff \forall j\, (1 \leq j \leq \tau) : s \equiv 0 \pmod{p_j^{\alpha_j}}.$$

Thus, we can express a $\mathrm{MOD}_m^{\{0\}}$ gate as an AND of τ modular $\mathrm{MOD}_{p_j^{\alpha_j}}^{\{0\}}$ gates. (Since m is constant, then τ and each p_i, α_i are constants.) Therefore, c_i has an AND gate at the output to which subcircuits $d_1, \ldots, d_\tau$ are connected. Each d_j is a depth two circuit with SYM gates at the input level and a $\mathrm{MOD}_{p_j^{\alpha_j}}^{\{0\}}$ gate at the output. The lower bound now follows from Theorem 4.3.3. □

Theorem 4.3.5. *If $C \in AND \circ \{MOD_m^{\{0\}}, EXACT\} \circ SYM$ computes the $\neg$ID function, then*

$$size(C_{2n}) = \frac{2^{\Omega(n)/\log \upsilon}}{\log \upsilon},$$

where υ is the largest fan-in of the EXACT gates at the second level.

Proof. Proceeding as in the proof of Theorem 4.3.4, the output gate of c_i is an exact gate, *i.e.,* EXACT_m for some integer m, of fan-in at most υ. It is a known fact from Number Theory (see [20] or Subsection 3.4.2) that there exist primes $1 < p_1 < \cdots < p_\tau \leq \ln \upsilon$, such that $p_1 \cdots p_\tau > \upsilon$. Thus, the EXACT_m gate outputs 1 if and only if the $\mathrm{MOD}_{p_1 \cdots p_\tau}^{\{m\}}$ gate outputs 1. Proceeding as in the case when the output gate of c_i was a modular gate and since $\tau \leq \ln \upsilon$, the result follows. □

4.3.4 Lower Bounds for AND, OR Functions

In [8] it was conjectured that polynomial-size constant-depth circuits with modular gates and constant fan-in AND gates cannot compute the AND_n function. Grolmusz & Tardos [24] proved a special case of this conjecture where the circuit class is $\mathrm{CC}^0[p^k] \circ \mathrm{MOD}_m^A \circ \mathrm{AND}_{O(1)}$

and the fan-in of MOD_m^A gates is $o(n^2/\log n)$. Using multiplicity automata, we show exponential lower bounds for $\mathrm{OR} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ and $\mathrm{AND} \circ \mathrm{CC}_d^0[p^k] \circ \mathrm{SYM}_{\epsilon n}$ circuits computing the AND_n and OR_n functions, respectively, for any $\epsilon \leq p^{-4kd}$.

Theorem 4.3.6. *Let p be any prime constant and let k, d be any positive constants. Let $\alpha = \sum_{j=1}^{d+1} p^{2kjd}$ and let $\epsilon = \frac{1}{4\alpha}$. If $C \in OR \circ CC_d^0[p^k] \circ SYM_{\epsilon n}$ computes the AND_n function, then*

$$size(C) = 2^{\Omega(n)}.$$

Proof. Suppose the AND_n function is computed by the circuit C. Consider the $\mathrm{ID} : \{0,1\}^{2n} \to \{0,1\}$ function which is defined as

$$\mathrm{ID}(\mathbf{x}) = \mathrm{AND}\left(\mathrm{MOD}_2^{\{0\}}(x_1, x_{n+1}), \ldots, \mathrm{MOD}_2^{\{0\}}(x_n, x_{2n})\right).$$

It follows that the ID function can be computed by C if the inputs to the circuit are the Boolean variables $\mathrm{MOD}_2^{\{0\}}(x_1, x_{n+1})$, ..., $\mathrm{MOD}_2^{\{0\}}(x_n, x_{2n})$, their negations, and the constants 0, 1. Since each input to C can be computed by a symmetric gate of fan-in 2, then the ID function can be computed by circuits of the form

$$\mathrm{OR} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}_{\epsilon n} \circ \mathrm{SYM}_2,$$

where the subscript of the symmetric gates indicates their fan-in. From Lemma 2.5.2, each subcircuit $\mathrm{SYM}_{\epsilon n} \circ \mathrm{SYM}_2$ can be computed by a single symmetric gate of fan-in $4^{\epsilon n}$. Therefore, the ID function can be computed by some circuit D of the form

$$\mathrm{OR} \circ \mathrm{CC}^0[p^k] \circ \mathrm{SYM}_{4^{\epsilon n}}.$$

Note that D differs from C only by how the input variables are connected to the symmetric gates and by the fan-in of the symmetric gates. It follows that,

$$\mathrm{size}(D) \leq 4^{\epsilon n}\,\mathrm{size}(C). \tag{4.3.4}$$

Write D as $\mathrm{OR}(d_1, \ldots, d_t)$ where each subcircuit $d_i \in \mathrm{CC}^0[p^k] \circ \mathrm{SYM}_{4^{\epsilon n}}$. Note that for each i, $1 \leq i \leq t$,

$$\mathrm{size}(D) \geq \mathrm{size}(d_i). \tag{4.3.5}$$

From the proof of Theorem 4.2.3, each subcircuit d_i can be simulated by a $\mathbb{Z}_p$-automaton M_i of size

$$s_i \leq N\left[\left(\mathrm{size}(d_i) + p^k\right) N\right]^{\alpha} \leq N^{1+\alpha} p^{k\alpha} \left[\mathrm{size}(d_i)\right]^{\alpha}, \tag{4.3.6}$$

where $N = (n+1)4^{\epsilon n}$ is the largest size of the automata simulating the symmetric gates as described in Lemma 4.2.1. From Theorem 4.1.4, we know that the size of any $\mathbb{Z}_p$-automaton computing the ID function is 2^n and thus, as in Theorem 4.3.2, for some $1 \leq i \leq t$,

$$s_i \geq 2^{n-\log t}. \tag{4.3.7}$$

By combining Equations 4.3.4–4.3.7 we obtain

$$\text{size}(C) \geq 2^{\frac{1}{\alpha}n(1-\epsilon(3\alpha+1))-\left(\log t+\log(n+1)+\alpha\log p^k\right)}.$$

The claim follows since $\epsilon = \frac{1}{4\alpha}$. □

Theorem 4.3.7. *Let p be any prime constant and let k, d be any positive constants. Let $\alpha = \sum_{j=1}^{d+1} p^{2kjd}$ and let $\epsilon = \frac{1}{4\alpha}$. If $C \in AND \circ CC_d^0[p^k] \circ SYM_{\epsilon n}$ computes the OR_n function, then*

$$size(C) = 2^{\Omega(n)}.$$

Proof. Similar to the proof of Theorem 4.3.6 but using the ¬ID function instead of the ID function. □

Chapter 5

Computing Pseudorandom Function Generators

A function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is called a pseudorandom function generator (PRFG) if communicating with a randomly chosen function of F cannot be efficiently distinguished from communicating with a truly random function. We investigate the minimal circuit complexity of a PRFG. Naor & Reingold [37] constructed a PRFG at least as secure as factoring Blum integers and another one whose security is based on some other assumption. From the work of Krause & Lucks [28], who constructed circuits to compute these PRFGs, it follows that the circuit class TC^0_5 is the simplest class of circuits known to contain a PRFG secure against polynomial-time adversaries based on the assumption that factoring is hard and the circuit class TC^0_4 is the simplest class of circuits known to contain a PRFG secure against polynomial-time adversaries based on other assumptions. In [28], it is also shown that TC^0_2 circuits cannot contain PRFGs secure against subexponential-time adversaries. We modify the first PRFG of [37] and compute it by some simple depth 3 quasipolynomial-size circuits, namely by $\mathrm{q(SUM}\circ\mathrm{EXACT)}^+$ and $\mathrm{q(OR}\circ\mathrm{EXACT)}^+$ circuits[1]. Assuming that factoring Blum integers is 2^{n^δ}-hard, the modified PRFG is secure against polynomial-time adversaries. It follows that these circuit classes are not learnable in polynomial time even when membership queries are allowed.

5.1 A PRFG based on the GDH-Assumption

Naor & Reingold [37] construct a PRFG based on the Generalized Diffie-Hellman (GDH) Assumption modulo a Blum integer N. We modify their PRFG by reducing the length of N from n bits to polylog(n) bits. The security of the original PRFG is based on the security

[1]Recall that '+' denotes AND gates of polylog(n) fan-in.

of the GDH-Assumption against polynomial-time adversaries. The security of our modified construction is also based on the security of the GDH-Assumption against polynomial-time adversaries, but our assumption is stronger since the length of N is polylog(n) bits. It is known that if a polynomial-time algorithm breaks the GDH-Assumption of [37], then there exists a polynomial-time algorithm for factoring Blum integers. This implies that if our Strong GDH-Assumption can be broken, then for every constant $\alpha > 0$, 2ℓ-bit Blum integers can be factored in time 2^{ℓ^α}. This contradicts a widely believed assumption that factoring 2ℓ-bit Blum integers cannot be done in time 2^{ℓ^δ} for some absolute constant $\delta > 0$. Therefore, our modified PRFG is secure against polynomial-time adversaries if factoring 2ℓ-bit Blum integers is 2^{ℓ^δ}-hard. Furthermore, as we will show in Section 5.2, this modified PRFG is computable by $\mathrm{q}(\mathrm{SUM} \circ \mathrm{EXACT})^+$ and $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ circuits as opposed to TC^0_5 circuits which compute the PRFG of [37].

We now define what it means for a function generator to be pseudorandom and describe in more detail our construction.

Loosely speaking, a *function generator* F is an algorithm which outputs a function f_s for each key s from a set of specified keys. A function generator F is *pseudorandom* if any efficient procedure cannot distinguish between a randomly chosen function from F and a truly random function. A distinguishing procedure is a Turing machine $\mathcal{M}$ with access to the oracle $\mathcal{O}$ which we denote by $\mathcal{M}^{\mathcal{O}}$. Whenever $\mathcal{M}$ writes a string $\mathbf{x}$ on the oracle tape, the oracle $\mathcal{O}$ responds in one computational step by writing the value $\mathcal{O}(\mathbf{x})$. The machine $\mathcal{M}$ may query the value of the oracle at arguments of its choice and it is allowed to choose the next query depending on all previous queries. At the end, the machine $\mathcal{M}$ decides whether the answers to its queries were given by a function generated by F or by a truly random function. The function generator F is pseudorandom if no efficient machine $\mathcal{M}$ can decide with non-negligible probability whether its oracle is a function generated by F or a truly random function. The following definitions which can be found in [37] state precisely what it means for a function generator to be pseudorandom.

Definition 5.1.1. *Let $\{A_n, B_n\}_{n\in\mathbb{N}}$ be a sequence of domains. A* function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbb{N}}$ of random variables, where the random variable F_n assumes values in the set $\{f \mid f : A_n \to B_n\}$. The* uniform function ensemble*, denoted $R = \{R_n\}_{n\in\mathbb{N}}$, has the random variable R_n distributed uniformly over the set $\{f \mid f : A_n \to B_n\}$.*

Definition 5.1.2. *A function ensemble F is called an* efficiently computable pseudorandom function ensemble *if the following conditions hold:*

- *(pseudorandomness) for every probabilistic polynomial-time oracle machine $\mathcal{M}$, every*

polynomial $p(\cdot)$, and all sufficiently large n,

$$\left|Pr\left[\mathcal{M}^{F_n}(1^n)=1\right]-Pr\left[\mathcal{M}^{R_n}(1^n)=1\right]\right|<\frac{1}{p(n)}.$$

- *(efficient computation) there exist probabilistic polynomial-time algorithms, $\mathcal{I}$ and $\mathcal{V}$, and a mapping from strings to functions, Φ, such that $\Phi\left(\mathcal{I}\left(1^n\right)\right)$ and F_n are identically distributed and $\mathcal{V}(i,x)=\left(\Phi(i)\right)(x)$, where $\Phi(i)$ is the function assigned to the string i.*

The analysis of the security of the pseudorandom function generators are based on some widely believed assumptions. One such assumption is the GDH-Assumption. Informally, the GDH-Assumption says that no efficient procedure can compute $g^{\prod_{i\in[1,n]}a_i} \bmod N$ even if it is allowed to query the value of $g^{\prod_{i\in I}a_i} \bmod N$ for any proper subset $I\subset[1,n]$, where N is a Blum integer, g is a quadratic-residue in $\mathbb{Z}_N^*$ and $a_1,\ldots,a_n$ are elements of $[1,N]$. The security of the PRFG proposed by Naor & Reingold [37] and of our modified construction is based on this assumption; our version of the assumption is stronger, *i.e.,* N has polylog(n) bits while in [37], N has n bits. We now formalize our strong GDH-Assumption.

First, we define the factoring-instance generator which on input ℓ outputs a 2ℓ-bit Blum integer.

Definition 5.1.3 (Naor & Reingold [37])**.** *The factoring-instance generator, FIG, is a probabilistic polynomial-time algorithm such that on input 1^ℓ its output, N, is distributed over 2ℓ-bit integers, where $N=PQ$ for two ℓ-bit primes, P and Q, such that $P\neq Q$ and $P\equiv Q\equiv 3 \pmod 4$.*

Second, we need to allow any procedure to query the value of $g^{\prod_{i\in I}a_i} \bmod N$ for any proper subset $I\subset[1,n]$ of its choice.

Definition 5.1.4 (Naor & Reingold [37])**.** *Let N be any positive integer, let g be any quadratic residue in $\mathbb{Z}_N^*$ and let $\mathbf{a}=\langle a_1,a_2,\ldots,a_n\rangle$ be any sequence of n elements of $[1,N]$. Define the function $h_{N,g,\mathbf{a}}$ with domain $\{0,1\}^n$ such that for any n-bit input, $\mathbf{x}=\langle x_1,\ldots,x_n\rangle$,*

$$h_{N,g,\mathbf{a}}(\mathbf{x})=g^{\prod_{x_i=1}a_i} \bmod N.$$

Define $h^r_{N,g,\mathbf{a}}$ to be the restriction of $h_{N,g,\mathbf{a}}$ to inputs $\{0,1\}^n-\{1\}^n$.

Assumption 5.1.1 (Strong GDH)**.** *Let $\ell : \mathbb{N} \to \mathbb{N}$. There exists an absolute constant $\delta > 0$, such that if $\ell(n) = \left\lceil \log^{1/\delta} n \right\rceil$, then for every polynomial-time probabilistic oracle machine $\mathcal{A}$, for every constant $\alpha > 0$, and all sufficiently large n,*

$$Pr\left[\mathcal{A}^{h^r_{N,g,\mathbf{a}}}(N, g, 1^n) = h_{N,g,\mathbf{a}}(1^n)\right] < \frac{1}{n^\alpha}$$

where the probabilities are taken over the random bits of $\mathcal{A}$, the choice of N according to the distribution $FIG\left(1^{\ell(n)}\right)$, the choice of g uniformly at random in the set of quadratic residues in $\mathbb{Z}^*_N$ and the choice of each of the values in $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$ uniformly at random in $[1, N]$.

Recall that in [37], N is an n-bit Blum integer. We now modify the PRFG construction of [37] by reducing the length of N to only polylog(n) bits.

Construction 5.1.1. *We define the function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$. Let $\ell : \mathbb{N} \to \mathbb{N}$ be defined by $\ell(n) = \left\lceil \log^{1/\delta} n \right\rceil$, where δ is the constant of Assumption 5.1.1. For every n, a key of a function in F_n is a tuple $\langle N, g, \mathbf{a}, \mathbf{r} \rangle$, where N is a $2\ell(n)$-bit Blum-integer, g is a quadratic-residue in $\mathbb{Z}^*_N$, $\mathbf{a} = \langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1} \rangle$ is a sequence of $2n$ values in $[1, N]$, and $\mathbf{r}$ is a $2\ell(n)$-bit binary string. For any n-bit input, $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$, the Boolean function $f_{N,g,\mathbf{a},r}$ is defined as*

$$f_{N,g,\mathbf{a},r}(\mathbf{x}) = \left(g^{\prod_{i=1}^{n} a_{i,x_i}} \bmod N\right) \odot \mathbf{r}.$$

The distribution of functions in F_n is induced by the following distributions on their keys: g, $\mathbf{a}$ and $\mathbf{r}$ are uniform in their range and the distribution of N is $FIG\left(1^{\ell(n)}\right)$.

Proceeding as in [37], the above function can be shown to be pseudorandom; we simply state the theorem without including a proof since the analysis mimics the proof of Theorem 5.4 in [37].

Theorem 5.1.1. *Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 5.1.1. If the Strong GDH-Assumption (Assumption 5.1.1) holds, then for every probabilistic polynomial time oracle*

machine $\mathcal{M}$, every constant $\alpha > 0$, and all sufficiently large n,

$$\left|Pr\left[\mathcal{M}^{f_{N,g,\mathbf{a},\mathbf{r}}}(N,g,1^n)=1\right] - Pr\left[\mathcal{M}^{R_n}(N,g,1^n)=1\right]\right| < \frac{1}{n^\alpha}$$

where in the first probability $f_{N,g,\mathbf{a},\mathbf{r}}$ is distributed according to F_n and in the second probability, the distribution of $\langle N,g\rangle$ is $FIG\left(1^{\ell(n)}\right)$ and R_n is uniformly chosen in the set of functions with domain $\{0,1\}^n$ and range $\{0,1\}$.

As it is seen from the above theorem, the modified PRFG remains secure even when the oracle $\mathcal{M}$ in addition to the length of the input knows the values of N and g.

In [31] it has been shown that if a polynomial-time algorithm breaks the GDH-Assumption used in [37], then there exists a polynomial-time algorithm that factors $2n$-bit Blum integers (see Theorem 5.5 of [37]). Thus, the security of the PRFG of [37] is based on the assumption that factoring Blum integers requires superpolynomial time. Our modified PRFG is also based on the assumption that factoring Blum integers is hard since the same reduction applies; however, since N is only $\ell = \text{polylog}(n)$ bits long we require that factoring cannot be done in time 2^{ℓ^δ} for some absolute constant $\delta > 0$.

Assumption 5.1.2 (Hardness of Factoring)**.** *There exists an absolute constant $\delta > 0$, such that there is no probabilistic algorithm $\mathcal{A}$ with running time 2^{ℓ^δ} and*

$$Pr\left[\mathcal{A}\left(N\right) = \langle P,Q\rangle\right] > 2^{-\ell^\delta},$$

where $N = PQ$ is a 2ℓ-bit Blum integer and P, Q are selected uniformly.

We say that factoring is 2^{ℓ^δ}-hard.

Theorem 5.1.2. *If the Strong GDH-Assumption (Assumption 5.1.1) does not hold, then Assumption 5.1.2 does not hold.*

Proof. Assume to the contrary that for every constant $\delta > 0$, there exists an algorithm $\mathcal{A}$ with running time $t(n) = n^\alpha$ for some constant α that breaks the GDH-Assumption and that there exists an absolute constant $\lambda > 0$ such that factoring 2ℓ-bit Blum integers is 2^{ℓ^λ}-hard, where $\ell = \ell(n) = \left\lceil \log^{1/\delta} n \right\rceil$. Recall that, N is a 2ℓ-bit integer. Thus, in terms of ℓ the running time of $\mathcal{A}$ is $T(\ell) = 2^{\alpha\ell^\delta}$. As in Theorem 5.5 of [37], this implies the existence of a factoring algorithm for 2ℓ-bit Blum integers with running time $n^\beta T(\ell) = 2^{\alpha\beta\ell^\delta}$ for some

constant β. We provide a contradiction by choosing a constant $\delta > 0$ such that $2^{\alpha\beta\ell^\delta} < 2^{\ell^\lambda}$. Our $\delta > 0$ satisfies

$$0 < \delta < \lambda - \log_\ell \alpha - \log_\ell \beta.$$

We can always find such δ since for sufficiently large n, for all constants δ, α and β, it holds that $\log_\ell \alpha < \lambda/3$ and $\log_\ell \beta < \lambda/3$. □

5.2 Computing the PRFG in q(SUM ∘ EXACT)$^+$ and q(OR ∘ EXACT)$^+$

In this section, we show how to compute the modified pseudorandom function of [37] by q(SUM ∘ EXACT)$^+$ and q(OR ∘ EXACT)$^+$ circuits. Our constructions are based on efficient circuits for iterated multiplication in $\mathbb{Z}_p$, where p is a prime number.

Definition 5.2.1. *Let m and $a_1, \ldots, a_n$ be integers. The function $\Pi_m : \mathbb{Z}^n \to \mathbb{Z}_m$ is defined as*

$$\Pi_m(a_1, \ldots, a_n) = \prod_{i=1}^{n} a_i \bmod m.$$

Definition 5.2.2. *Let $\ell \in$ polylog(n). Let $N = PQ$ be a 2ℓ-bit Blum-integer, $g \in \mathbb{Z}_N^*$, $\mathbf{a} = \langle a_{1,0}, a_{1,1}, \ldots, a_{n,0}, a_{n,1} \rangle \in \{1, \ldots, N\}^{2n}$, and $\mathbf{r} \in \{0,1\}^{2\ell}$. The function $f_{GDH} : \{0,1\}^n \to \{0,1\}$ is defined as*

$$f_{GDH}(\mathbf{x}) = \left(g^{\prod_{i=1}^{n} a_{i,x_i}} \bmod N \right) \odot \mathbf{r},$$

where $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ is the n-bit input string.

Theorem 5.2.1. *The function f_{GDH} (Definition 5.2.2) is in*

$$q(SUM \circ EXACT)^+.$$

The proof of this theorem is based on the following lemma of Maciel [32]. For completeness, we include the proof of the lemma.

Lemma 5.2.2 (Maciel [32]). *Let p^υ be a prime power of p and let $a_1, \ldots, a_n$ be ℓ-bit integers. Then, $\Pi_{p^\upsilon}(a_1, \ldots, a_n)$ is computed by SYM$\circ$AND$_\ell$ circuits, where the fan-in of the symmetric gate is $2^\ell(3np^\upsilon)^3$.*

Proof. Let $\mathbb{Z}^*_{p^\upsilon}$ be the set of integers in $\mathbb{Z}_{p^\upsilon}$ that are not divisible by p. Then, it can be shown that $\mathbb{Z}^*_{p^\upsilon}$ is a group of order $p^\upsilon - p^{\upsilon-1}$ with respect to multiplication modulo p^υ. Moreover, $\mathbb{Z}^*_{p^\upsilon}$ is cyclic, unless $p = 2$ and $\upsilon > 2$; in that case $\mathbb{Z}^*_{p^\upsilon}$ is generated by $2^\upsilon - 1$ and 5.

For every $1 \leq i \leq n$, let p^{t_i} be the largest power of p dividing $a_i \bmod p^\upsilon$ and let $r_i = (a_i \bmod p^\upsilon)/p^{t_i}$. Also, let $t = \sum_{i=1}^n t_i$ threshold υ, which means that $t = \sum_{i=1}^n t_i$ if $\sum_{i=1}^n t_i < \upsilon$, and $t = \upsilon$ otherwise. Thus, $r_i \in \mathbb{Z}^*_{p^\upsilon}$ and

$$\Pi_{p^\upsilon}(a_1, \ldots, a_n) \equiv \prod_{i=1}^n a_i \equiv p^t \prod_{i=1}^n r_i \pmod{p^\upsilon}.$$

First, suppose $\mathbb{Z}^*_{p^\upsilon}$ is a cyclic group and let g be a multiplicative generator. Clearly, for every r_i, there exists a unique α_i, $0 \leq \alpha_i < p^\upsilon - p^{\upsilon-1}$, such that

$$r_i = g^{\alpha_i} \bmod p^\upsilon.$$

Let $\alpha = \sum_{i=1}^n \alpha_i \bmod (p^\upsilon - p^{\upsilon-1})$. Then, $\prod_{i=1}^n r_i \equiv g^\alpha \pmod{p^\upsilon}$, since $g^{p^\upsilon - p^{\upsilon-1}} \equiv 1 \pmod{p^\upsilon}$. Therefore, Π_{p^υ} can be computed as follows.

1. For $i = 1, \ldots, n$, compute α_i and t_i.
2. Compute α and t.
3. Compute $(p^t g^\alpha) \bmod p^\upsilon$.

Note that, $\alpha_i = \sum_{j=1}^{(p^\upsilon - p^{\upsilon-1})-1} [\alpha_i \geq j]$ and $t_i = \sum_{j=1}^{\upsilon-1} [t_i \geq j]$. The value of each α_i and t_i is determined by the value of the corresponding a_i. Therefore, each $[\alpha_i \geq j]$ and $[t_i \geq j]$ is a function of at most ℓ input bits. Furthermore,

$$\alpha = \left(\sum_{i=1}^n \sum_{j=1}^{(p^\upsilon - p^{\upsilon-1})-1} [\alpha_i \geq j] \right) \bmod (p^\upsilon - p^{\upsilon-1})$$

and

$$t = \left(\sum_{i=1}^n \sum_{j=1}^{\upsilon-1} [t_i \geq j] \right) \text{ threshold } \upsilon.$$

Thus, $(p^t g^\alpha) \bmod (p^\upsilon)$ is a function of two sums of size at most np^υ. Therefore, Lemma 2.5.2 implies that every bit of $(p^t g^\alpha) \bmod (p^\upsilon)$ can be computed by a symmetric gate of fan-in $(2np^\upsilon)^2$ whose inputs are the $[\alpha_i \geq j]$ and $[t_i \geq j]$.

In the non-cyclic case, Π_{2^υ} can be computed as follows.

1. For $i = 1, \ldots, n$,in parallel
   (a) compute α_i.
   (b) for $j = 1, \ldots, 2^{\upsilon-2} - 1$, determine if $b_i \geq j$.
   (c) for $j = 1, \ldots, \upsilon - 1$, determine if $t_i \geq j$.
2. Compute $(2^t(2^\upsilon - 1)^\alpha 5^b) \bmod 2^\upsilon$.

The analysis for this case is similar and we get for both cases that the computation of Π_{p^υ} can be carried out by computing:

1. Functions of ℓ input bits.
2. Symmetric gates of fan-in $(3np^\upsilon)^3$.

Any function of ℓ bits can be computed by $\mathrm{SUM}_{2^\ell} \circ \mathrm{AND}_\ell$ circuits using disjunctive normal form. Therefore, since $\mathrm{SYM} \circ \mathrm{SUM} = \mathrm{SYM}$, each bit of Π_{p^υ} is computable by $\mathrm{SYM} \circ \mathrm{AND}_\ell$ circuits where the fan-in of the symmetric gate is $2^\ell(3np^\upsilon)^3$. □

Proof of Theorem 5.2.1. Let $\phi(N) = \prod_{i=1}^m q_i$ be the prime decomposition of $\phi(N)$, where $\phi(\cdot)$ is the Euler function. From the Prime Number Theorem (Eq. 3.4.2), it is clear that $m, \log q_i \in O(\ell)$. Let $y = \prod_{i=1}^n a_{i,x_i}$ and $y_i = y \bmod q_i$. Then,

$$f_{\mathrm{GDH}}(\mathbf{x}) = (g^{y \bmod \phi(N)} \bmod N) \odot \mathbf{r}.$$

Note that $y = \prod_{i=1}^n b_{i,0}\, b_{i,1}$, where for every i, $1 \leq i \leq n$, we define

$$b_{i,0} = \begin{cases} a_{i,0}, & \text{if } x_i = 0, \\ 1, & \text{otherwise.} \end{cases} \quad \text{and} \quad b_{i,1} = \begin{cases} a_{i,1}, & \text{if } x_i = 1, \\ 1, & \text{otherwise.} \end{cases}$$

Let $\left\langle \alpha^\ell_{i,j}, \ldots, \alpha^1_{i,j} \right\rangle$ and $\left\langle \beta^\ell_{i,j}, \ldots, \beta^1_{i,j} \right\rangle$ be the binary representations of $a_{i,j}$ and $b_{i,j}$ respectively, $i = 1, \ldots, n$, $j = 0, 1$. A simple analysis shows that,

$$\beta^1_{i,0} = \begin{cases} x_i, & \text{if } \alpha^1_{i,0} = 0, \\ 1, & \text{if } \alpha^1_{i,0} = 1, \end{cases} \qquad \beta^1_{i,1} = \begin{cases} \neg x_i, & \text{if } \alpha^1_{i,1} = 0, \\ 1, & \text{if } \alpha^1_{i,1} = 1, \end{cases}$$

and, for $2 \leq k \leq \ell$,

$$\beta^k_{i,0} = \begin{cases} 0, & \text{if } \alpha^k_{i,0} = 0, \\ \neg x_i, & \text{if } \alpha^k_{i,0} = 1, \end{cases} \qquad \beta^k_{i,1} = \begin{cases} 0, & \text{if } \alpha^k_{i,1} = 0, \\ x_i, & \text{if } \alpha^k_{i,1} = 1. \end{cases}$$

Therefore, since the value of each $a_{i,j}$ is known, the computation of each $b_{i,j}$ is carried out at no expense — we simply connect the input wires as indicated above. Once each of the values $b_{i,j}$ is known, from Lemma 5.2.2, y_i can be computed by $\mathrm{SYM} \circ \mathrm{AND}_\ell$ circuits, where the fan-in of the symmetric gate is $2^\ell(3nq_i)^3 \in n^{O(\ell)}$. From the Chinese Remainder Theorem, we know that there exist $u_1, \ldots, u_m$ such that $y = (\sum_{i=1}^m u_i y_i) \bmod \phi(N)$. Therefore,

$$f_{\mathrm{GDH}}(\mathbf{x}) = \left(g^{(\sum_{i=1}^m u_i y_i) \bmod \phi(N)} \bmod N\right) \odot \mathbf{r}.$$

Note that, the values of g, N, $\mathbf{r}$ are known. Since the values of $u_1, \ldots, u_m$ depend only on N, they can be precomputed. Therefore, f_{GDH} is a function of $y_1, \ldots, y_m$. Since each y_i is at most a 2ℓ-bit integer and $m \in O(\ell)$, then f_{GDH} is a function of $O(\ell^2)$ bits. Therefore, f_{GDH} is in $\mathrm{SUM}_{2^{\ell^{O(1)}}} \circ \mathrm{AND}_{O(\ell^2)} \circ \mathrm{SYM}_{n^{O(\ell)}} \circ \mathrm{AND}_\ell$. From Lemma 2.5.2, each subcircuit $\mathrm{AND}_{O(\ell^2)} \circ \mathrm{SYM}_{n^{O(\ell)}}$ can be replaced by a single symmetric gate of fan-in $n^{\ell^{O(1)}}$. Each symmetric gate can be replaced by SUM $\circ$ EXACT circuits. Also, note that SUM $\circ$ SUM $=$ SUM. Therefore, f_{GDH} is computed by circuits of the form

$$\mathrm{SUM}_{n^{\ell^{O(1)}}} \circ \mathrm{EXACT} \circ \mathrm{AND}_\ell,$$

where the fan-in of the EXACT gates is $n^{\ell^{O(1)}}$. Since $\ell \in \log^{O(1)} n$, then the fan-in of the SUM and EXACT gates is $n^{\log^{O(1)} n}$. □

Theorem 5.2.3. *The function f_{GDH} (Definition 5.2.2) is in*

$$q(OR \circ EXACT)^+.$$

Proof. Notice that in the proof of Theorem 5.2.1 we could have used OR gates instead of SUM gates. □

5.3 Hardness of Learning

It is a known result that a circuit class containing a PRFG secure against polynomial-time adversaries cannot be learned in polynomial-time (see [49]). In Section 5.2, based on the assumption that factoring 2ℓ-bit Blum integers is 2^{ℓ^δ}-hard, we showed that the circuit classes $\mathrm{q}(\mathrm{SUM} \circ \mathrm{EXACT})^+$ and $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ contain such a PRFG. Therefore, these circuit classes are not weakly learnable in polynomial time even when membership queries are allowed. We now define what it means for a circuit class to be weakly learnable.

In Section 4.1 we described the exact learning model. Another learning model is the Probably Approximately Correct (PAC) model introduced by Valiant [49]. In this model,

the learning algorithm is not allowed to ask membership or equivalence queries, and therefore, the task is more difficult. Let $\mathcal{C}$ be a complexity class and let $f \in \mathcal{C}$ be an unknown member of $\mathcal{C}$. The random example oracle, denoted $\mathrm{EX}(f, \mathcal{D})$, when invoked randomly chooses $\mathbf{x}$ according to the distribution $\mathcal{D}$ and outputs a labeled example $\langle \mathbf{x}, f(\mathbf{x}) \rangle$. Loosely speaking, a learning algorithm, after observing many labeled pairs $\langle \mathbf{x}, f(\mathbf{x}) \rangle$ generated by $\mathrm{EX}(f, \mathcal{D})$, should be able to output a hypothesis h that agrees with f on many inputs. We say that an algorithm $\mathcal{A}$ PAC-learns $\mathcal{C}$ under the distribution $\mathcal{D}$ with accuracy ϵ and confidence δ if $\mathcal{A}$ satisfies the following condition: for every $0 < \epsilon,\ \delta < 1$, if $\mathcal{A}$ is given access to $\mathrm{EX}(f, \mathcal{D})$, then, with probability at least $1 - \delta$, algorithm $\mathcal{A}$ outputs an hypothesis h such that $\Pr[h(\mathbf{x}) = f(\mathbf{x})] \geq 1 - \epsilon$. If $\epsilon = \frac{1}{2} - \gamma$ for some $0 < \gamma < \frac{1}{2}$, then the learning algorithm is called *weak*.

As we mentioned, if a circuit class contains a pseudorandom function generator, then it is not learnable under the PAC model with membership queries. Furthermore, the PAC learning algorithm can simulate equivalence queries and if it is allowed to ask membership queries, it can simulate any exact learning algorithm. Consequently, if a circuit class contains a pseudorandom function generator, then it is also not learnable under the exact model.

Theorem 5.3.1. *Let $\mathcal{C}$ be a circuit class which computes the pseudorandom function generator family $\{F_n\}_{n \in \mathbb{N}}$ of Construction 5.1.1. Suppose that there is a constant $\alpha > 0$, and a membership query algorithm $\mathcal{A}$ running in time n^α that* weakly *learns the class $\mathcal{C}$ with accuracy $\frac{1}{2} + \frac{1}{n^\alpha}$, that is, for every function $f : \{0,1\}^n \to \{0,1\}$ computable by $\mathcal{C}$ circuits the algorithm $\mathcal{A}$ outputs an hypothesis $h : \{0,1\}^n \to \{0,1\}$, such that*

$$Pr[h(\mathbf{x}) = f(\mathbf{x})] \geq \frac{1}{2} + \frac{1}{n^\alpha},$$

where $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ is chosen uniformly from $\{0,1\}^n$. Then, Assumption 5.1.2 does not hold.

Proof. Assume that there exists an algorithm $\mathcal{A}$ satisfying the conditions of the theorem. We use the learning algorithm $\mathcal{A}$ to construct an oracle machine $\mathcal{M}$ that breaks the pseudorandom function generator of Construction 5.1.1, that is, there exists a constant $\beta > 0$, such that $\mathcal{M}$ runs in time n^β and for all sufficiently large n's

$$\left| \Pr\left[\mathcal{M}^{f_{N,g,\mathbf{a},\mathbf{r}}}(N, g, 1^n) = 1\right] - \Pr\left[\mathcal{M}^{R_n}(N, g, 1^n) = 1\right] \right| > \frac{1}{n^\beta},$$

where the distributions are as in Theorem 5.1.1. The claim, then, follows from the contrapositive of Theorem 5.1.1 and from Theorem 5.1.2. We now proceed with the construction

of the oracle machine $\mathcal{M}$. On input $\langle N, g\rangle$, the oracle machine $\mathcal{M}$ simulates the learning algorithm $\mathcal{A}$. The machine $\mathcal{M}$ invokes its oracle to answer the membership queries of $\mathcal{A}$. Let h be the hypotheses produced by $\mathcal{A}$. Clearly, if $f_{N,g,\mathbf{a},\mathbf{r}}$ was the oracle of $\mathcal{M}$, then $\Pr[h(\mathbf{x}) = f_{N,g,\mathbf{a},\mathbf{r}}(\mathbf{x})] \geq \frac{1}{2} + \frac{1}{n^{\alpha}}$; and if R_n was the oracle of $\mathcal{M}$, then $\Pr[h(\mathbf{x}) = R_n(\mathbf{x})] = \frac{1}{2}$. Thus, $\mathcal{M}$ can test whether its oracle is $f_{N,g,\mathbf{a},\mathbf{r}}$ or R_n by quering its oracle and comparing the answers to those given by the hypothesis h. □

5.4 Remarks on Natural Proofs

Razborov & Rudich [41] argued that all lower bound proofs for non-monotone non-uniform Boolean circuits are based on the following strategy:

— Formulate a combinatorial property of the Boolean functions which in some mathematical sense measures the "variation" of the function values.

— Show that circuits from a certain class $\mathcal{C}$ can only compute functions of "low" variation.

— Find a function f with "high" variation, and conclude that $f \notin \mathcal{C}$.

They introduced the notion of *natural proofs* to formalize the above structure[2]. They concluded that this strategy is weak, in the sense that, under some widely believed cryptographic assumption, current methods cannot be used to separate P from NP.

To define a natural proof, Razborov & Rudich developed the notion of *natural combinatorial properties*.

Definition 5.4.1 (Razborov & Rudich [41])**.** *A* combinatorial property of boolean functions *is a sequence* $\{C_n\}_{n\in\mathbb{N}}$ *of subsets, where for every* $n \in \mathbb{N}$, C_n *is a subset of the set of all Boolean functions on* n *variables,* i.e.,

$$\{C_n \subseteq \mathcal{B}_n : n \in \mathbb{N}\}$$

Let Γ *and* Λ *be two complexity classes and let* δ_n *be a real-valued function. The combinatorial property* C_n *is called* Γ-natural against Λ with density δ_n *if it satisfies the following conditions:*

[2]Since the time of the latest publication, 1997, new lower bounds have been proven. To our knowledge, all the proofs have been natural or could be represented as natural.

constructivity: *The predicate* $f_n \in^? C_n$ *is computable in* Γ.

largeness: $|C_n| \geq \delta_n \mathcal{B}_n$.

usefulness: *For any sequence of functions* $\{f_n\}_{n\in\mathbb{N}}$, *where* $f_n \in C_n$ *for infinitely many* n*'s,* $f_n \notin \Lambda$.

It is not difficult to see that the lower bound proofs presented in this thesis are natural. To illustrate, consider the lower bound proof for the circuit class $\Lambda = \mathrm{CC}^0[p^k] \circ \mathrm{SYM}$. We define the natural combinatorial property C_n as the set of all functions $f_n \in \mathcal{B}_n$ for which the rank of the corresponding Hankel matrix is large. In Section 4.1, we showed that the rank of the Hankel matrix F of a function on n variables is expressed as $\mathrm{rank}(F) = \sum_{\ell=0}^{n} \mathrm{rank}(F^\ell)$, where F^ℓ is the submatrix of F whose rows are indexed by strings of length ℓ and columns by strings of length $n-\ell$. Therefore, since the rank of a matrix is computable by NC^2 circuits, the combinatorial property C_n is NC^2-natural. The density δ_n of the combinatorial property C_n is at least $1/2$ (see [41]). We also showed how to simulate Λ circuits by multiplicity automata with only polynomial blow-up in size. Since the rank of the Hankel matrix of a function f_n is equal to the size of the smallest multiplicity automaton computing f_n, it is clear that C_n is useful against Λ. Thus, any function with the property C_n is not in Λ.

Razborov & Rudich [41] observed that a Γ-natural lower bound proof that some function f is not in the circuit class Λ can be used to construct an algorithm in Γ that would distinguish f from a pseudorandom function in Λ. Therefore, if a circuit class Λ contains a pseudorandom function secure against Γ-attacks, then there are no Γ-natural lower bound proofs against Λ.

Definition 5.4.2 (Γ-Attacks with Density δ_n)**.** *Let* $F = \{F_n\}_{n\in\mathbb{N}}$ *be a sequence of pseudorandom function generators. Let* $\delta : \mathbb{N} \to \mathbb{R}$ *and let* $\delta_n = \delta(n)$. *Let* Γ *and* Λ *be two circuit classes. Then,* F *is a pseudorandom function generator family* δ*-secure against* Γ*-statistical tests, if for all sufficiently large* n, *for every statistical test* $C_n \in \Gamma$, *and for every random seed, it holds that*

$$|Pr[\textit{random function in } \mathcal{B}_n \in C_n] - Pr[f_{\textit{random seed}} \in C_n]| < \delta_n.$$

Theorem 5.4.1 (Natural Proof Theorem [41])**.** *If a complexity class* Λ *contains a pseudorandom function generator family* $F = \{F_n\}_{n\in\mathbb{N}}$ *which is* δ*-secure against* Γ*-statistical tests, then there is no* Γ*-natural combinatorial property useful against* Λ *with density* δ.

It is very important to notice the difference between the statistical tests (Definition 5.4.2) and pseudorandomness (Definition 5.1.2). In a statistical test, the input to the algorithm is the function represented by its truth table which has exponential length in the number of variables of the function. In Definition 5.1.2, the number of times the algorithm can query the function is bounded by a polynomial in the number of variables of the function. Thus, the security of a PRFG family against statistical tests, in general, does not follow from the pseudorandomness of the family as in Definition 5.1.2.

Therefore, despite the fact that a PRFG can be computed by $\mathrm{q(SUM \circ EXACT)}^+$ and $\mathrm{q(OR \circ EXACT)}^+$ circuits, we cannot claim that there are no natural lower bound proofs for the above circuit classes. If we could show that the modified PRFG (Chapter 5) is secure against statistical tests, then there would be no natural proofs against $\mathrm{q(OR \circ EXACT)}^+$ and $\mathrm{q(SUM \circ EXACT)}^+$ circuits. Consequently, we would have an extremely fine line of separation between the circuit classes for which we know lower bounds, *i.e.,* $\mathrm{AND \circ EXACT \circ SYM}$, and circuit classes for which lower bound proofs would have to be unnatural, *i.e.,* $\mathrm{q(OR \circ EXACT)}^+$ and $\mathrm{q(SUM \circ EXACT)}^+$.

Chapter 6

Conclusions and Further Research

We presented a new method based on multiplicity automata to prove lower bounds for some circuit classes. Our method simplifies the lower bound proof for $\mathrm{MOD}_m^{\{0\}} \circ \mathrm{SYM}$ circuits and extends the result in some ways.

We used ideas from computational learning theory to develop a lower bound method. An interesting line of research is to transform other learning algorithms into lower bound proofs, and ideally, to show that learning a circuit class implies a lower bound for it and vice-versa. In Chapter 5 we showed that the circuit class $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ is not learnable in polynomial time under the usual learning models. In Chapter 4, we showed a strong lower bound for a closely related circuit class, namely $\mathrm{AND} \circ \mathrm{EXACT} \circ \mathrm{SYM}$ circuits. We would obtain a fine line of separation between learnability and hardness of learnability if we could use the lower bound proof to show that $\mathrm{q}(\mathrm{AND} \circ \mathrm{EXACT})^+$ circuits can be learned.

An important question to consider is whether the modified PRFG (Chapter 5) is secure against statistical tests. A positive answer would imply that there are no natural proofs against $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ and $\mathrm{q}(\mathrm{SUM} \circ \mathrm{EXACT})^+$ circuits. Consequently, we would have an extremely fine line of separation between the circuit classes for which we know lower bounds, *i.e.,* $\mathrm{AND} \circ \mathrm{EXACT} \circ \mathrm{SYM}$, and circuit classes for which lower bound proofs would have to be unnatural, *i.e.,* $\mathrm{q}(\mathrm{OR} \circ \mathrm{EXACT})^+$ and $\mathrm{q}(\mathrm{SUM} \circ \mathrm{EXACT})^+$.

In Chapter 4 we showed lower bounds for several circuit classes which could not be simulated by multiplicity automata. Our proofs exploited the fact that one of the subcircuits of a circuit with an AND or an OR gate at the top computes a similar function to that computed by the entire circuit. In a similar manner, we would like to exploit the properties of other gates and use our method to prove lower bounds for circuits of the form $G \circ \mathcal{C}$, where G is a threshold or a symmetric gate, and $\mathcal{C}$ is a circuit class that can be simulated by multiplicity automata.

Bibliography

[1] Eric Allender, Michael E. Saks, and Igor E. Shparlinski. A lower bound for primality. *Journal of Computer and System Sciences*, 62(2):356–366, 2001.

[2] Noga Alon and Richard Beigel. Lower bounds for approximations by low degree polynomials over Z_m. *Proc. 16th Annual IEEE Conference on Computational Complexity*, 2001.

[3] Noga Alon and Ravi B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7:1–2, 1987.

[4] Alexander E. Andreev. On a method for obtaining lower bounds for the complexity of individual monotone functions. *Doklady Akademii Nauk USSR*, 282(5):1033–1037, 1985.

[5] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[6] James Aspnes, Richard Beigel, Merrick Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):1–14, 1994.

[7] David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4(1):367–382, 1994.

[8] David A. Mix Barrington, Howard Straubing, and Denis Thérien. Nonuniform automata over groups. *Information and Computation*, 89(2):109–132, 1990.

[9] Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, 4(4):350–366, 1994.

[10] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM*, 47(3):506–530, 2000.

[11] Francesco Bergadano, Nader H. Bshouty, Christino Tamon, and Stefano Varricchio. On learning branching programs and small depth circuits. *Lecture Notes in Computer Science*, 1208:150–161, 1997. Third European Conference on Computational Learning Theory.

[12] Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalent queries. *SIAM Journal on Computing*, 25(6):1268–1280, 1996.

[13] Anna Bernasconi, Carsten Damm, and Igor E. Shparlinski. The average sensitivity of square-freeness. *Computational Complexity*, 9(1):39–51, 2000.

[14] Anna Bernasconi, Carsten Damm, and Igor E. Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Information and Computation*, 168(2):113–124, 2001.

[15] Anna Bernasconi and Igor E. Shparlinski. Circuit complexity of testing square-free numbers. *Proc. 16th Annual International Symposium on Theoretical Aspects of Computer Science, Trier, Germany.* Lecture Notes in Computer Science, 1563:47–56, 1999.

[16] Jehoshua Bruck. Harmonic analysis of polynomial threshold functions. *SIAM Journal on Discrete Mathematics*, 3:168–177, 1990.

[17] Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions, and spectral norms. *SIAM Journal on Computing*, 21:33–42, 1992.

[18] Nader H. Bshouty, Christino Tamon, and David K. Wilson. Learning matrix functions over rings. *Algorithmica*, 22(1-2):91–111, 1998.

[19] Don Coppersmith and Igor E. Shparlinski. On polynomial approximation of the discrete logarithm and the Diffie-Hellman mapping. *Journal of Cryptology*, 13(3):339–360, 2000.

[20] Harold Davenport. *Multiplicative number theory*, volume 74 of *Graduate Texts in Mathematics*. Springer-Verlag, 3rd edition, 2000.

[21] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits and the polynomial time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[22] Frederic Green. A complex-number Fourier technique for lower bounds on the MOD-m degree. *Computational Complexity*, 9(1):16–38, 2000.

[23] Vince Grolmusz. On the weak MOD-m representation of Boolean functions. *Chicago Journal of Theoretical Computer Science*, 1(2):1–8, 1995.

[24] Vince Grolmusz and Gábor Tardos. Lower bounds for (MOD p, MOD m) circuits. *SIAM Journal on Computing*, 29(4):1209–1222, 2000.

[25] András Hajnal, Wolfgang Maass, Pavel Pudlák, and György Turán. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46(2):129–154, 1993.

[26] Johan Håstad. *Computational limitations for small depth circuits*. MIT Press, Cambridge, MA, 1987.

[27] Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113–129, 1991.

[28] Matthias Krause and Stefan Lucks. On the minimal hardware complexity of pseudorandom function generators. *Proc. 18th Annual International Symposium on Theoretical Aspects of Computer Science*, pages 419–430, 2001.

[29] Matthias Krause and Pavel Pudlák. On computing boolean functions by sparse real polynomials. *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 682–691, 1995.

[30] Matthias Krause and Stephan Waack. Variation ranks of communication matrices and lower bounds for depth two circuits having symmetric gates with unbounded fan-in. *Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 777–782, 1991.

[31] Michael Luby. *Pseudorandomness and cryptographic applications*. Princeton University Press, 1996.

[32] Alexis Maciel. *Threshold Circuits of Small Majority-Depth*. PhD thesis, School of Computer Science, McGill University, February 1995.

[33] Alexis Maciel and Erion Plaku. Multiplicity automata and lower bounds for small depth circuits. *Manuscript*, 2002.

[34] Alexis Maciel and Erion Plaku. Pseudorandom function generators in depth 3 circuits. *Manuscript*, 2002.

[35] Alexis Maciel and Denis Thérien. Threshold circuits of small majority-depth. *Information and Computation*, 146(1):55–83, 1998.

[36] Marvin L. Minsky and Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1988. Expanded version of the original 1968 edition.

[37] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Proc. 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 458–467, 1997.

[38] Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994.

[39] Erion Plaku and Igor E. Shparlinski. On polynomial representations of boolean functions related to some number theoretic problems. *Proc. 21st Conference on Foundations of Software Technology and Theoretical Computer Science, Bangalore, India.* Lecture Notes in Computer Science, 2245:305–316, 2001.

[40] Alexander A. Razborov. Lower bounds on the complexity of some boolean functions. *Doklady Akademii Nauk USSR*, 31:354–357, 1985.

[41] Alexander A. Razborov and Steven Rudich. Natural proofs. *Proc. 26th ACM Symposium on Theory of Computing*, pages 204–213, 1994.

[42] Alexander A. Razborov and Avi Wigderson. $n^{\Omega(\log n)}$ lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom. *Information Processing Letters*, 45(6):303–307, 1993.

[43] Jean-Jacques Risler. Additive complexity and zeros of real polynomials. *SIAM Journal on Computing*, 14(1):178–183, 1985.

[44] Igor E. Shparlinski. *Number theoretic methods in cryptography: Complexity lower bounds.* Birkhäuser, 1999.

[45] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. *Proc. 19th ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[46] Gábor Tardos and David A. Mix Barrington. A lower bound on the MOD 6 degree of the OR function. *Computational Complexity*, 7(2):99–108, 1998.

[47] Shi-Chun Tsai. Lower bounds on representing boolean functions as polynomials in Z_m. *Proc. 25th ACM Symposium on Theory of Computing*, pages 96–101, 1993.

[48] Shi-Chun Tsai. A depth 3 circuit lower bound for the parity function. *Journal of Information Science and Engineering*, 17(5):857–860, 2001.

[49] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[50] Andrew C. Yao. Separating the polynomial-time hierarchy. *Proc. 26th International Journal of Foundations of Computer Science*, pages 1–10, 1985.