Growth Processes on Formulas and Reversible Circuits

by

Alexander Brodsky

B.Math. University of Waterloo, 1997
M.Sc. University of British Columbia, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

June 2003

In presenting this thesis/essay in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying for this thesis for scholarly purposes may be granted by the Head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Aug 1, 2003
Date

Department of Computer Science
The University of British Columbia
2366 Main mall
Vancouver, BC
Canada V6T 1Z4

Abstract

Among their many uses, growth processes have been used for constructing reliable networks from unreliable components (Moore and Shannon, 1956) and deriving complexity bounds of various families of functions (Valiant, 1984). Hence, analyzing such processes is an important and challenging problem. In this thesis we parameterize a growth process by its initial conditions and characterize it by the existence and shape of a limiting probability distribution that describes the likelihood of it realizing a particular Boolean function. We identify the limiting distributions of several classes of growth processes on formulas, and derive conditions under which results such as Valiant's hold. We consider growth processes that use linear, self-dual, and monotone connectives, completely characterizing those processes that use linear or monotone connectives. In the latter case, we derive a novel technique that combines spectral analysis (Savický, 1990) with probabilistic arguments; the technique is also applicable to growth processes that use other connectives. Our characterizations also yields bounds on the convergence of these growth processes.

Unfortunately, a comparable definition and characterization of growth processes on general circuits is impractical due to the dependencies between the probabilities associated with various circuit components. However, reversible circuits (Toffoli, 1980) are inherently more structured than general circuits. To study growth processes beyond the formula setting, we propose and characterize growth processes on reversible circuits. Intriguingly, aspects of the characterizations that proved difficult in the former setting—such as proving that the limiting distribution is uniform—turn out to be relatively easy in the latter. In fact, the limiting distribution of a growth process on reversible circuits is characterized completely by its support—the set of functions that the process can generate. Consequently, we also provide bounds on the convergence of these growth processes.

Finally, the regular structure of reversible circuits provides ample motivation for considering the reversible circuit complexity of finite Boolean functions—an important issue, since the precursor to such applications as quantum computation is reversible computation. We derive relationships between reversible circuits and other models of computation such as permutation branching programs (Barrington, 1985), based on a new measure that we call bandwidth. By leveraging these relationships, we exhibit a natural gap between two families of reversible circuits that correspond to width-4 and width-5 permutation branching programs. Based on the same measure, we define a hierarchy of families of reversible circuits that corresponds to the **SC** class hierarchy—a natural circuit-based definition of the class **SC**. Lastly, we provide constructions for several common Boolean functions and derive sufficient conditions under which a Boolean function has a polynomial-size realization.

Contents

List of Figures

Acknowledgements

"Research does not occur in a vacuum" is an oft-cited axiom, meaning that research does not occur in isolation. An immediate corollary is that this work would not have been possible without the contributions of a great many people. Primus inter pares, I thank my supervisor and mentor Nick Pippenger. Without his patience, insight, and encouragement, I doubt that this work would have come to fruition; via example and advice, he was my first guide through world of theoretical computer science. With equal import, I thank my friend and mentor Bettina Speckmann, whose generosity, steadfastness, ideals, and uncompromising demand for excellence in all aspects of life greatly tempered me. I have no doubt that without her friendship and example, I would be a different person; thank you for the ideas, arguments, schemes, and the multifarious late-night discussions!

My home for the past five years has been the Distributed Systems Group, who accepted me—a theory person—and provided me with a lively and stimulating work environment—though I must admit that I do not have a performance section in my dissertation. Many thanks to my twin brother Dima, my great friend Matt Pedersen, as well as Yvonne, Chamath, Joon, and the rest of the DSG. You made me look forward to spending each day in the lab!

Thank you to the Theory Group for the collaboration, interaction, and feedback that I have received over the years. Special thanks to Mike McAllister for taking me under his wing when I arrived here six years ago, Stephane Durocher and Ellen Gethner for their collaboration on the rectilinear crossing number problem, and Anne Condon, Joel Friedman, and David Kirkpatrick for being on my committee and providing very helpful feedback.

The Department of Computer Science at UBC is a unique and an amazing place to work in; many of its members have made my tenure here an experience that I would not trade for anything. Many thanks to my fellow grad students and friends, Andrea, Paul, Martin, Lisa, and Brian, for your friendship, and to the faculty, Alan, Norm, and Gail for sharing your experiences and your advice.

Most importantly, I thank my family for their everlasting support. In my weekly conversations I was always asked three questions: Are you healthy? Have you found someone nice? Are you done yet? I am pleased to say that I can answer affirmative to the first and the last questions; I am still working on number 2. Thanks to my parents, Larisa and Boris, my grandparents, Bella and Benjamin, my favourite brother Dima, and everyone else for the love, expectations, and care packages!

ALEXANDER BRODSKY

The University of British Columbia
June 2003

Chapter 1

Introduction

Bounding the circuit complexity of Boolean functions is an outstanding and ongoing challenge. Determining the circuit complexity of a Boolean function yields insight not only into the practical aspects of complexity, i.e., how many transistors does it take to realize a given function, but also into the structure and relationships of complexity classes. The techniques for deriving the bounds have themselves become objects of study; generalizing techniques—making them applicable to a wider class of problems—is an important problem in its own right. Thus, it goes without saying that exploration of the applicable techniques furthers the study of complexity.

One well known technique that has come to prominence over the past two decades is the probabilistic method. The probabilistic method, commonly termed *probabilistic amplification*, has been used to characterize complexity classes [Adl78, BG81, ABO84] and to derive nonconstructive upper bounds on the circuit complexity of specific Boolean functions [Val84, Raz88]. The latter method iteratively constructs, or grows, a circuit via a random process. After some number of iterations the resulting circuit may realize the required function with a nonzero probability. We call the latter form of the probabilistic method a *growth process*.

To make the analysis of a growth process tractable, the constructions are usually restricted to a subclass of circuits. Otherwise, the dependencies between the probabilities associated with various components of a circuit render the analysis intractable. For example, both Valiant [Val84] and Razborov [Raz88], restricted their attention to fan-out 1 circuits: formulas. Alternatively, if the circuits are reversible—comprising fan-out 1 bijective gates—the growth processes are also amenable to analysis.

Reversible circuits [Lan61, Tof80, FT82] and reversible computation [Ben73, Ben89] constrain every gate and every step of the computation to be completely reversible, so that no information may be lost at any step of the computation. Reversible circuits, which realize a bijection from n inputs to n outputs, comprise n wires that are manipulated by bijective gates. Since quantum computation has come to the forefront of theoretical and applied research, investigations into reversibility and reversible circuits have assumed a more prevalent role [Ben88a, Ben88b] because reversibility is a precondition of quantum computation. Fortuitously, the restrictive nature of reversibility induces a structure on the circuits that is much more amenable to analysis; both within the framework of growth processes as well as for circuit complexity in general.

In this thesis we are concerned with growth processes on formulas, growth processes on reversible circuits, and reversible circuit complexity. In the first two parts of the thesis we investigate and characterize growth processes based on their initial parameters; we formulate general theorems that predict the resulting distributions. The goal is to provide a methodology for constructing a growth process, given some specification. This not only provides a design strategy, but also delineates the natural limitations of growth processes.

In the latter part of the thesis we investigate reversible circuit complexity. We investigate if there exists a reversible circuit complexity hierarchy analogous to the one comprising the low-lying complexity classes, an important goal for fitting reversible computation within the current complexity hierarchy. We also investigate the relationships between reversible circuits and other models of computation, as well as concrete constructions, which allow us to derive a set of useful rules and heuristics for general construction of reversible circuits.

1.1 Growth Processes on Formulas

The *growth process*, elegantly exhibited by Valiant [Val84] and first studied by Moore and Shannon [MS56], is used to nonconstructively prove the realization of a given n-adic function. The growth process on Boolean formulas is a random process defined on the space of n-adic Boolean functions, whose parameters are the initial distribution on the variables, negations, and constants, and a fixed k-adic Boolean function, called a *connective*. To grow a formula of depth i, the random process first grows k random formulas of depth $i-1$ and then composes them using the connective; formulas of depth 0 are chosen randomly according to the initial distribution. If the probability of growing a random formula that realizes a particular function can be shown to be greater than zero, then the function may be realized in k^i gates—the size of the random formula.

Alternatively, a random Boolean formula generated by i iterations of the growth process comprises a complete k-ary tree of depth i, where each internal node corresponds to the connective in the natural way, and the leaf nodes are randomly assigned according to the initial distribution. Consequently, the realized function only depends on the connective and the assignment of the leaf nodes.

For each iteration of the growth process there is a corresponding distribution on the formulas of depth i that induces a distribution on the space of n-adic Boolean functions. If the induced distribution approaches some fixed distribution as i approaches infinity, then the growth process has a *limiting distribution*, which is approached at some rate as i approaches infinity. The limiting distribution and the rate of approach characterize the growth process. In many applications [Val84, Bop85, GM91, Sav95a, Sav98] the rate of approach is the main issue of interest. As we will show, in almost all cases that we considered, the rate of approach is exponential in n.

In this thesis we are concerned with three questions: the existence of the limiting distribution, the shape of the limiting distribution, and the rate at which the limiting distribution is approached. In most cases, we assume that the initial distribution is uniform over its support and that the support is a subset of the variables, their negations, and constants. For a given support and connective, the goal is to answer the three preceding questions: the existence, shape and convergence to a

limiting distribution. For example, Savický [Sav90, Sav95a] formulated broad conditions under which the limiting distribution is uniform over the entire n-adic function space and is approached at an exponential rate.

1.2 Reversible Circuits and Growth Processes on Reversible Circuits

Reversible circuit, first proposed by Landauer [Lan61] and Toffoli [Tof80], must satisfy one criterion: no information may be lost during any step of the computation. For example, an AND-gate loses information, because given an output of 0, it is impossible determine if the inputs were $(0,0)$, $(0,1)$, or $(1,0)$. Consequently, all gates must be bijective and have fan-out 1. Using the Toffoli's terminology, a reversible circuit on n inputs comprises n or more wires, called *lines* that are manipulated by gates that take some number of lines as inputs, and place output on the same set of lines; see Figure 1.1. Additional lines, called *garbage* (or *ancillary*) lines, may be used for temporary storage of values.

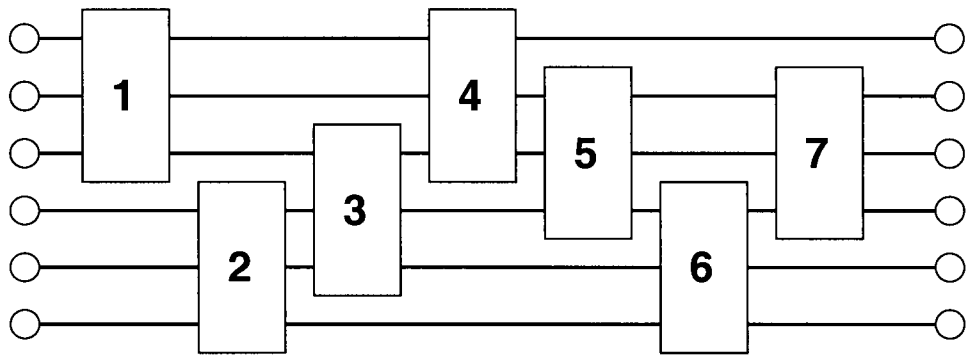


Figure 1.1: A block diagram of a reversible circuit.

A reversible circuit is simply a sequence of gates that are placed on the lines and manipulate their values as the lines pass through the gates. The *length* or *size* of the circuit is equal to the length of the gate sequence. The standard gates—defined in Section 2.3.1—such as the NOT gate, the controlled-NOT gate, and the Toffoli gate, only modify one of the lines and use the remaining lines as read-only control lines. This creates a natural distinction between *read-only* lines, which are not modified by any gate in the circuit, and *read-write* lines, which may be modified by at least one gate in the circuit. The *width* of a circuit is the number of lines and the *bandwidth* of a circuit is the number of read-write lines.

A growth process on a reversible circuit is a random process defined on the space of permutation functions and whose parameters are the number of lines in the circuit, and the distribution on the set of gates that may comprise the circuit. The growth process starts with an empty circuit, which realizes the identity permutation. On each iteration the process selects a gate from the distribution and suffixes it to the circuit from the preceding iteration, growing the circuit by one gate. For each iteration of the process there is a corresponding distribution on the set of reversible functions, which

induces a distribution on the set of permutations.

Naturally, we consider the same three issues as in the preceding section: the existence of a limiting distribution, the support of the the limiting distribution, and the convergence rate to the limiting distribution. Not surprisingly, the characteristics depend exclusively on the width of the circuit and the distribution on the set of gates. As we shall see, under a very broad set of constraints, the limiting distribution exists and is uniform over the set of even permutation functions.

1.3 Reversible Circuit Complexity

The complexity of a reversible circuit is usually parameterized by a combination of circuit depth, width, and bandwidth, which are usually stated as functions of the number of variables. The bandwidth of a circuit corresponds to the amount of state that a circuit must track during the computation and is analogous to the space requirements of a computation. Analogously, circuit length corresponds to the time requirements of a computation.

Reversible circuits, and reversible computation in general, present an interesting space-time trade-off. As Landauer [Lan61] observed, information cannot be thrown away during computation; it must either be stored, or reversibly erased. Bennett [Ben73, Ben89] demonstrated that the latter is accomplished by running the computation that generated the information backwards, leveraging the fact that the computation is reversible. The trade-off between storing data, reversible erasure, and regeneration is analogous to the trade-off between space and time, i.e., a computation may either use additional space to store the information that was generated during the course of the computation, or use additional time to reversibly erase unnecessary information. Within the framework of reversible Turing machines, Bennett [Ben73, Ben89] and Lange et al. [LMT97, LMT00] derived optimal strategies for both time parsimonious and space parsimonious computation. Additionally, Li and Vitányi [LV96a, LV96b] and, Buhrman et al. [BTV01] derived general space-time tradeoffs using versions of Bennett's [Ben89] pebbling games that simulate reversible computation.

Toffoli [Tof80] noted the analogous requirement for reversible circuits. Reversible circuits use some number of garbage lines to store temporary values used during the computation. Before being reused, a garbage line must be erased; this is usually accomplished by reversing the computation that generated the value. Consequently, there is a trade-off between the length of the circuit and the number of garbage lines required.

Various models of computation, such as branching programs [Lee59, Weg87, Weg79, Weg82] and straight line programs [Ost54, Cle90], have been used to investigate various problems in circuit complexity. Similarly, models such as permutation branching programs [CG75, Bar85, Bar89, BS95], programs over groups [BT88, BST90], and reversible straight-line programs [Ost54, Cle90], are used to investigate problems within reversible circuit complexity. Results within the frameworks of these computation models are directly applicable—with a little work—within the context of reversible circuit complexity, and vice versa.

In this thesis, we use the framework of permutation branching programs and programs over groups to investigate reversible circuits with low or or constant bandwidth. There is a natural relationship between permutation branching programs and a class of reversible circuits whose variable

lines are read-only and whose read-write lines represent the state of the branching program during the computation. On the other hand, reversible straight-line programs are closely related to reversible circuits that comprise mostly read-write lines.

In recent years, reversible circuits have generated greater interest within both the quantum computation and the hardware communities. Reversible computation is a necessary requirement for quantum computing and uses significantly less power [Ben82, BGL+93, Ben88a, Ben88b], an important benefit to today's hardware architects. Consequently we consider concrete realizations of several commonly utilized Boolean functions.

1.4 Results in this Thesis

Our goal, as stated previously, is to improve our understanding of two frameworks: growth processes and reversible circuits. Our approach for the former provides a set of general theorems that characterize a growth process based on its initial parameters. We develop theorems for both growth processes on formulas and growth processes on reversible circuits. Our approach for the latter yields a multifarious set of results: relationships between reversible circuits and various models of computation, new bounds on constant bandwidth reversible circuits, new concrete reversible realizations of commonly used Boolean functions, and new heuristics for constructing reversible circuits.

The third chapter of this thesis derives a set of theorems for characterizing growth processes on formulas. After presenting the notation (Section 3.1), we characterize growth processes that use linear connectives (Section 3.2), self-dual connectives (Section 3.3), and monotone connectives (Section 3.4). We use a novel combination of amplification arguments [Val84, Bop85] and spectral analysis [Raz88, Sav90, Sav95a] to derive a technique for characterizing growth processes using other connectives as well (Sections 3.4.2 and 3.5).

Additionally, we derive general convergence bounds for the majority of the growth processes under investigation. We note that in some cases even very minute changes in the connective can result in drastically different convergence rates.

The fourth chapter derives a characterization of growth processes on reversible circuits. We relate growth processes on reversible circuits to random walks on groups [Ald89, KLNS89]. Our characterization of growth processes on reversible circuits is accomplished via the techniques used to analyze the random walks. We define growth processes on reversible circuits (Section 4.1) and derive a broad set of constraints that ensure that the limiting distribution exists and is uniform over the set of even permutation functions (Section 4.2–4.3). We also show that the convergence rate of such growth processes is exponentially slower than in the case of growth process on formulas (Section 4.4).

The fifth chapter focuses on reversible circuit complexity. In Section 5.1 we review permutation branching programs [Bar85], three different notions of acceptance, and define the notion of a program transformation. Using these notions, we derive relationships between reversible circuits and permutation branching programs, making our results applicable to both reversible circuits and permutation programs (Section 5.2). Using two different complexity measures on reversible circuits we define a complexity hierarchy and relate it to the low-lying complexity classes (Section 5.3).

Additionally, in Section 5.4, we describe several concrete reversible circuit constructions, including: an incrementor, threshold function, consensus function and adder. We show that all of these have concise constructions and derive a set of heuristics for realizing other Boolean functions. Particularly, in Subsection 5.4.2, we show that if a permutation has a polynomial size cycle representation, then the permutation can be realized by a polynomial size reversible circuit!

Chapter 2

Background

2.1 Boolean Functions, Formulas, and Circuits

A Boolean cube of order n is the set $B_n = \{0,1\}^n$ of all n-bit vectors and an n-adic Boolean function is a map $f : B_n \rightarrow B_1$, i.e., an n-adic Boolean function take n Boolean arguments and yields a truth value, 0 (false) and 1 (true). A function is monotone if for any assignment of variables, changing any variable from 0 to 1 does not cause the function to change from 1 to 0.

A Boolean circuit is represented by a directed acyclic graph comprising nodes of positive outdegree and a single node of outdegree 0. Nodes of indegree 0 are labeled by a Boolean variable x_i, or its negation $\bar{x}_i$, and nodes of positive indegree are labeled by a Boolean function, which is typically a conjunction ($\wedge$), disjunction ($\vee$), or negation ($\neg$); the node of outdegree 0 is called the output node. The circuit computes a truth value in the natural way; the value of a node is either the value of the variable with which it is labeled, or the value of the function with which it is labeled—the function's arguments are the values of the node's predecessors. The truth value of the computation is the value of the output node. A Boolean formula is a Boolean circuit where each node has outdegree (fan-out) of at most 1. A monotone circuit (formula) is a Boolean circuit (formula) such that none of the nodes are labeled by negations or nonmonotone functions.

2.2 Random Formulas

Random Boolean formulas have been studied throughout the last half century. Five years after Shannon's [Sha38] seminal paper relating Boolean algebra and switching circuits, Riordan and Shannon [RS42] proved that "almost all" Boolean functions require an exponential number of circuit elements to be realized. Although Shannon [Sha49] proved an upper bound on the complexity of realizing any Boolean function on n variables, it wasn't until 1959 that Lupanov [Lup61b] obtained an asymptotic upper bound corresponding to the lower bound in [RS42].

There was a quick succession of results on bounding the size of circuits and formulas that realize any Boolean function on n variables [Lup58, Lup61a, Lup61b, Lup65, Kri61, Rez62, Kor65, Kor66, Pip76] in the early 60s. However, apart from the quadratic lower bounds on formulas of Nečiporuk [Neč66] and Khrapchenko [Khr71, Khr72], little was accomplished in deriving bounds

on the complexity of specific functions.

In the early 80's new techniques for deriving lower bounds were introduced. The main features were restrictions on the class of allowable circuits—bounded circuit depth or monotone circuits—and the use of probabilistic techniques. Furst et al. [FSS81] showed that parity could not be computed by a polynomial size bounded depth circuit, by a probabilistic technique involving random partial assignments, called *random projections*. Similar methods were used by Yao [Yao83, Yao85] and Hastad [Has86, Has89]. Razborov [Raz85] proved a super-polynomial lower bound on monotone circuits that decide the clique problem. This was extended to an exponential lower bound by Andreev [And85] and, Alon and Boppana [AB87] and used by Tardos [Tar88] to prove an exponential gap between monotone and nonmonotone circuit models.

Not surprisingly, probabilistic techniques are extremely useful in proving upper bounds as well. In 1984 Valiant [Val84], using random formulas, proved that the formula complexity of the majority function was not only polynomial—an implication of a result of Ajtai et al. [AKS83] on sorting networks—but that the size was bounded by $O(n^{5.3})$. Four years later, Razborov [Raz88] proved that there exist polynomial size monotone formulas that represent some Ramsey graphs. In the former case, probabilistic amplification was used to prove the existence of the small formula that realizes the majority function. In the latter case, spectral analysis was used to prove that the probability of creating the appropriate small formula was not zero.

2.2.1 Probabilistic Amplification

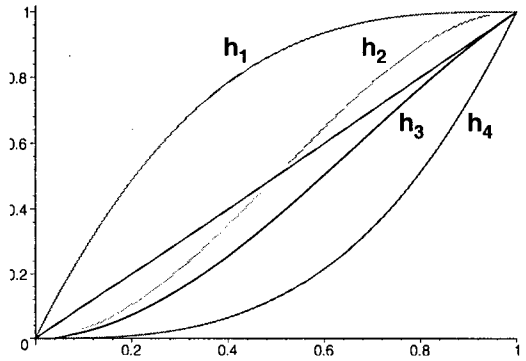
Following the work of von Neumann [vN56], Moore and Shannon [MS56] demonstrated how to construct "Reliable Circuits from Less Reliable Relays". This two part work laid the foundations for the probabilistic technique that would later be called *probabilistic amplification*. Moore and Shannon [MS56] asked the following question: can a reliable two-terminal circuit—one that behaved as if constructed from reliable relays—be constructed using unreliable "crummy" relays that work with probability p; all relays are controlled by a single source. Their approach was to construct a more reliable relay from a number of crummy relays, iterating the process until the required level of reliability was achieved.

Let p be the probability that a crummy relay *works* (closes when its coil is activated) and let $1-p$ be the probability that the crummy relay does not work (remains open when its coil is activated). Given an network of k relays that connect two terminals, the probability that the network works like a relay—the two terminals will become connected when the coils are activated—is equal to the probability that some number of relays will work, creating a path from one terminal to the other. This probability is denoted by the corresponding *characteristic polynomial*

$$h(p) = \sum_{i=0}^{k} B_i p^i (1-p)^{k-i},$$

where B_i is the number of ways of selecting i of k relays that if closed (work), create a closed circuit. Figure 2.1 illustrates behaviour of some characteristic polynomials.

Since each relay within the network is also a two-terminal network, one possible way to improve reliability is to replace each relay within the original k-relay network with a copy of the network



$$h_1(p) = 3p(1-p)^2 + 3p^2(1-p) + p^3$$

$$h_2(p) = 3p^2(1-p) + p^3$$

$$h_3(p) = 2p^2(1-p) + p^3$$

$$h_4(p) = p^3$$

Figure 2.1: Some typical characteristic polynomials for networks of three crummy relays.

itself; this is called an *iterative composition.* Analogously, the probability that the iterative composition will work is equal to the composition of the original k-relay network's characteristic polynomial with itself, i.e., $h(h(p))$, and the probability that a network that comprises i iterative compositions will work is $h^i(p)$, the ith composition of $h(p)$ with itself. Moore and Shannon [MS56] showed that iteratively composing the k-relay network with itself monotonically increases (or decreases) the reliability of the overall network, and that the rate of change is determined by the characteristic polynomial of the k-relay network.

For example, let N_1, N_2, N_3, and N_4 be the four different 3-relay networks that correspond to the characteristic polynomials in Figure 2.1. In the first case, since $h_1(p) > p$ on the interval $(0,1)$ and hence $h(h^{i-1}(p)) > h^i(p)$, an iterative composition of N_1 realizes a reliable relay that works with probability approaching 1—as the number of iterations increases—regardless of how crummy the original relays were, as long as they're not completely broken. Similarly, since $h_3(p) < p$ and $h_4(p) < p$ on $(0,1)$, iterative compositions of N_3 or N_4 realize very crummy relays that work with probability approaching 0—unless the crummy relays are not crummy at all. Finally, since $h_2(p)$ has a fixed-point, in this case $h(\frac{1}{2}) = \frac{1}{2}$, an iterative composition of N_2 realizes a reliable relay if the relays are not half bad[1], and realizes a crummy relay if the relays are more than half bad.

Moore and Shannon [MS56] considered relays that not only did not close properly, but also that did not open properly, causing a closed circuit even when the relay coils were inactive. Thus, their focus was on networks whose characteristic polynomial had a fixed-point. In [MS56] several important properties about $h(p)$ are proved including that $h(p)$ has at most one fixed-point on the interval $(0,1)$ and that the slope at the internal fixed-point is strictly greater than 1 unless $h(p) = p$. The latter implies that for all p less than the fixed-point, say s, $h(p) < p$ on $(0,s)$, and $h(p) > p$ on $(s,1)$.

Following Moore and Shannon [MS56], the technique proved useful in showing that certain functions had a small circuit or formula complexity. Following Adleman's [Adl78] characterization of the complexity class **RP**, Bennett and Gill [BG81] proved a similar result for the class **BPP**, namely, that the languages in **BPP** could be decided by polynomial size circuits. Additionally,

[1] $p > \frac{1}{2}$

Ajtai and Ben-Or [ABO84] showed that probabilistic constant depth circuits can be simulated by their deterministic analogs. However, perhaps the most elegant exhibition of the technique was by Valiant [Val84].

Since closing a relay in a circuit cannot open a closed circuit and since opening a relay cannot close an open circuit, two-terminal circuits are equivalent to monotone Boolean formulas. For every k element network, there is a corresponding k-adic formula, called a *connective*, whose inputs correspond to the relays of the network. The formula evaluates to true on an assignment if and only if closing the corresponding relays closes the k element network. Analogously, an iterative composition of the formula, replaces each instance of a variable with a copy of the formula—the variables are renamed in the copy. If we assume that the variables are independent and are true with probability p and false with probability $1-p$, then the probability of the formula evaluating to true is equal to $h^i(p)$, where $h^i(p)$ is ith composition of $h(p)$, the characteristic polynomial of the k element network. We call this iterative composition process a *growth process*.

This framework was used by Valiant [Val84] to prove that the majority function can be realized by a monotone formula of size $O(n^{5.3})$. Valiant showed that iteratively composing the connective $(G_1 \vee G_2) \wedge (G_3 \vee G_4)$ with itself about $2.65\log(n)$ times and then randomly assigning the n variables and 0 to the inputs of the resulting formula realizes the majority function with high probability. The distribution used for the variable assignment was $p(x_i) = \phi$ and $p(0) = 1 - n\phi$, where $\phi = s/(n-1)$ and s is the fixed-point of the connective.

The key to the proof is the probabilistic amplification phenomenon. Since the probability of a variable being true corresponds to the weight of the assignment, the probability of an input being 1 is greater than s if and only if the majority of the variables are 1. Consequently, the iterative composition creates divergence away from the internal fixed-point followed by convergence toward the bounding fixed-points. Consequently, with high probability the resulting formula evaluates to true if the majority of the inputs are 1, and false otherwise.

Boppana [Bop85, Bop89] further studied probabilistic amplification, deriving upper and lower bounds on the monotone formula complexity of Boolean threshold functions. He showed that Valiant's [Val84] result was indeed optimal, and that the monotone formula complexity of the mth threshold function is bounded by $O(m^{4.3}n\log(n))$, where $0 \leq m \leq n$. This was accomplished by analyzing the slope of the characteristic polynomial near its fixed-points, in order to obtain bounds on both the divergence away from the internal fixed-point and convergence towards the bounding fixed-points.

Additionally, Gu and Maruoka [GM91] investigated amplification within the context of formulas comprising alternating levels of AND and OR gates, and Radhakrishnan and Subrahmanyam [RS94] investigated threshold functions within the context of directed contact networks. Dubiner and Zwick [DZ92, DZ97] extended Boppana's [Bop89] lower bounds to nonmonotone formulas. All of the preceding work analyzed growth processes whose limiting distribution became concentrated on a single function, i.e., a threshold function; however many of the growth processes have a limiting distribution that is spread over a significant portion of the space of Boolean functions, requiring a different approach.

2.2.2 Spectral Analysis

If the limiting distribution of a growth process is not concentrated on a single function, then proving that a limiting distribution exists or even that there is a nonzero probability of growing a specific function requires a different approach. One such approach is spectral analysis, which was used by Razborov [Raz88] and Savický [Sav88, Sav90, Sav94, Sav95a, Sav95b]. In the former case, Razborov [Raz88] used spectral analysis to prove that some large graphs with Ramsey properties had formula representations that are exponentially smaller; this was later quantitatively improved by Savický [Sav95b], who demonstrated that these Ramsey properties are far from random.

In 1988, Savický [Sav88, Sav90] proved that under a broad set of conditions, the limiting distribution of a growth process is uniform over the entire space of n-adic functions. Five years later he further generalized the conditions and determined the asymptotic convergence rate towards the limiting distribution. Under a general set of conditions Savický [Sav94, Sav95a] showed that the rate of approach of the probability of realizing a particular function f to the limiting probability 2^{-2^n} yields information about f: the rate is fastest for the linear functions, and slowest for the *bent* functions [Rot76]—bent functions are furthest, in Hamming distance, from the linear functions.

In [Sav88, Sav90] Savický leverages the fact that the limiting distribution is uniform over the space of n-adic Boolean functions and that the Fourier transform (defined below) of the uniform distribution is concentrated on a single Fourier coefficient. He proves that all but one of the Fourier coefficients of the ith distribution approach zero as i approaches infinity, implying that the ith distribution approaches the uniform distribution as i approaches infinity. In [Sav95a] Savický derives an exponentially decaying upper bound on the magnitude of the Fourier coefficient, proving that asymptotically, the convergence is exponentially fast.

The Fourier transform Δ_i of a probability distribution π_i is defined by

$$\Delta_i(f) = \sum_{g \in \mathcal{F}_n} (-1)^{\langle f,g \rangle} \pi_i(g) \tag{2.1}$$

where $\mathcal{F}_n$ is the set of all n-adic Boolean functions and $\pi_i(g)$ is the probability of selecting function g from the ith distribution. The vectors f and g are Boolean vectors of size 2^{2^n} that represent functions by their truth table. Naturally, the inverse Fourier transform is defined by

$$\pi_i(g) = \frac{1}{2^{2^n}} \sum_{f \in \mathcal{F}_n} (-1)^{\langle f,g \rangle} \Delta_i(f). \tag{2.2}$$

Since the Fourier coefficient of the zero function, $\mathbf{0}$, is always equal to one, i.e., $\Delta_i(\mathbf{0}) = 1$ for all i, Savický [Sav90] shows that all other coefficients tend to zero as i tends to infinity, proving his result.

The Fourier transform plays a role in many of our results, but it needs to be adapted in various ways to suit different cases. For example, when dealing with linear functions, we represent the functions f and g in definition 2.1 not as Savický does, by their truth-tables, but rather by their coefficients as multivariate polynomials over $GF(2)$. In other cases, when establishing a limiting distribution that is uniform over a proper subset of all n-adic Boolean functions, we use what we call "restriction lemmas", which assert relationships that hold among the coefficients of the Fourier transform.

2.2.3 Related Work

Not all growth processes have a limiting distribution. In some cases the processes may have *alternating limiting distributions*, an example of this is worked out in Section 3.2. In this case, as i approaches infinity, the sequence of distributions π_{2i} converges to one distribution, while the sequence of distributions π_{2i+1} converges to another distribution. This is due to the fact that some functions may not be realizable by formulas of odd (or even) depth. For example, a growth process whose initial distribution is uniform on the projections and whose connective is $\overline{(G_1 \vee G_2) \wedge (G_3 \vee G_4)}$, has an alternating distribution; even depth formulas realize only the monotone functions, while odd depth formulas realize only the negated monotone functions.

In general, if the projection functions cannot be realized by formulas of synchronized depth greater than some constant depth, no limiting distribution can exist [HN77, HN79]. This inquiry was begun by Kurdryavcev [Kud60b, Kud60a], who investigated the completeness of automata with no feedback—essentially combinational circuits. Some of this work was rediscovered [HR98] by Loomis [Loo65] in 1965. In 1977, Hikata and Nozaki [HN77, HN79] proved that projections are necessary and sufficient to ensure completeness. However, this is not necessarily a sufficient condition for the existence of a limiting distribution.

Using another model of random formulas—where a formula is approximated by an infinite tree—Lefmann and Savický [LS97] obtained a relationship between the formula complexity of a function and the probability of its occurrence within the infinite tree model, namely the negative logarithm of the probability differs by a polynomial factor from the formula complexity of the respective function. For a fixed connective, $G_1 \wedge G_2 \oplus G_3 \oplus G_4$, Savický [Sav98] derived a bound between the formula complexity of a function and the probability that a random formula, grown using the connective, realizes the function. This was accomplished by deriving the behaviour of the Fourier coefficients from one iteration of the growth process to the next.

2.3 Reversible Computation

In 1948 Shannon and Weaver [Sha48, SW49] formalized the equivalence between information and entropy, a result dating back to Maxwell's Demon [Max71] and the work Szilard [Szi29]. In 1961 Landauer [Lan61] showed that if a computation is performed reversibly—none of the operations throw away information—then the amount of power required to drive the computation is dependent only on its rate. Landauer [Lan61] also proposed a reversible gate that was capable of simulating traditional irreversible Boolean operations, such as conjunction.

In the early 70's Bennett [Ben73] continued the work on reversible computation within the framework of reversible Turing machines. He proved that any Turing machine could be simulated reversibly by using extra space proportional to the length of the computation. Using the same framework, the trade-off between space and time was further investigated by Bennett [Ben89], Levine and Sherman [LS90], and Lange et al. [LMT00]. This trade-off was also investigated within the context of pebbling games and information theory [Zur89, BGL+93, LV96a, LV96b, LTV98, ABOIN96, BTV01]. However, not until 1980 was a realizable model of reversible computation proposed.

2.3.1 Reversible Circuits

Reversible circuits were first defined by Toffoli and Fredkin in [Tof80, FT82]. The circuits comprise fan-out 1 gates that implement bijective functions and operate on fixed subsets of lines. The three standard gates are: the unary NOT gate, which negates the value on the wire; the binary controlled-NOT gate, which negates the second line if the value of the first line is 1, and the ternary Toffoli gate (controlled-controlled-NOT), which negates the third line if the first two lines both have a value of 1; see Figure 2.2. A feature of these gates is that each one is its own inverse.

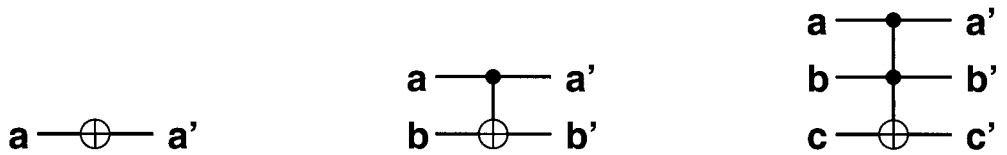


Figure 2.2: A NOT ($a' = 1 \oplus a$), a controlled-NOT ($b' = b \oplus a$), and a Toffoli ($c' = a \wedge b \oplus c$).

Since reversible circuits are bijective, not all n-adic Boolean functions may be realized by an n line reversible circuit. For example, the disjunction on n variables cannot be implemented by an n line circuit where the output is, say, the first line in the circuit. This follows from the fact that n line reversible circuits are bijections on the Boolean cube, hence, for exactly half of the outputs the first line must have a value of 0. Thus, in most of the cases an additional line, which is initialized to 0 and called a *garbage* line, is required. Using the following elegant argument, Toffoli [Tof80] showed that one additional line is both necessary and sufficient.

Let $F : B_m \rightarrow B_n$ be an arbitrary Boolean function. Construct a reversible function $R_F : B_{m+n} \rightarrow B_{m+n}$ such that F is embedded within it. Begin with a truth table for $F = \{(x,y) \in B_m \times B_n \mid F(x) = y\}$. Let the truth table for R_F be

$$\{(x',y') \in B_{m+n} \times B_{m+n} \mid x' = zx,\ y' = (y \oplus z)x,\ z \in B_m,\ F(x) = y\}.$$

This truth table defines a bijective function, within which F is embedded. In fact, with a little twiddling, its easy to ensure that R_F is an even permutation function. This is vital because even if an n-adic Boolean function is a permutation function, it may not be realizable by an n line circuit.

Universality

Given a universal set of gates, e.g., $\{\wedge, \neg\}$, any Boolean function can be realized using only gates from the set. In the case of reversible gates and circuits, regardless of what finite set of gates is used, there exist reversible functions that are not realizable without the use of an additional line. This was shown by Coppersmith and Grossman [CG75].

Let $k < n$. The operation of a k-ary invertible gate within an n line circuit realizes an even permutation—an element of the alternating group A_{2^n}. Assuming that universal set of gates is finite, the set contains a gate with a maximal number of inputs and outputs, say k. Hence, all n line circuits, where $k < n$ are composed of gates that only realize even permutations, implying that such

circuits can only realize even permutations themselves. Thus, reversible functions that compute odd permutations of the symmetric group S_{2^n} cannot be realized. By using an additional line, any odd permutation on 2^n points can be embedded into an even permutation on 2^{n+1} points and hence, can be realized by an $n+1$ line reversible circuit.

We say that a set of gates is *universal*, with respect to reversible circuits, if any permutation in the alternating group A_{2^n} may be realized by an n line reversible circuit comprising these gates. Coppersmith and Grossman [CG75] proved that the Toffoli gate and the NOT gate, which they call 2-functions, form a universal set.

Related Results

Fredkin and Toffoli [FT82] derived similar results for conservative circuits. Conservative circuits are reversible circuits that preserve the weight of the wires—the number of ones and zeros—across every operation. In this case, the number of additional lines required to realized any n-adic Boolean function is equal to the maximum difference between the weight of an input and the weight of its corresponding output.

Both the results of Toffoli [Tof80] and, Fredkin and Toffoli [FT82] imply no bounds on the size of a reversible circuit that realizes a particular function. However, taking a cue from the framework of reversible Turing machines [Ben73, Ben89, LMT00], there is a trade-off between the number of garbage lines and circuit size. Since creating garbage, without cleaning it up, creates unnecessary entropy; a simple technique, first exhibited by Bennett [Ben73], can be used to clean up the garbage lines, i.e., reversibly reseting them to zero, at a cost of some additional lines.

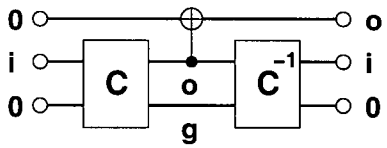


Figure 2.3: A method for resetting garbage lines.

Let C be a reversible circuit comprising a number of input/output lines, and additional garbage lines. The construction to reversibly reset the garbage lines to zero is shown in Figure 2.3; here i denotes the input, g denotes the garbage, o denotes the output of circuit C, and C^{-1} denotes circuit C's inverse. The gate sequence of C^{-1} is simply the reverse gate sequence of circuit C. The concept of trading lines (space) for another resource reappears frequently in the literature on the complexity of reversible computation; see section 2.5.

2.4 Random Reversible Circuits

Unlike growth processes on formulas, the multiple fan-out of gates within a general circuit makes analysis of growth processes on circuits intractable. Although reversible circuits retain the interdependence of general circuits, growth processes on reversible circuits are much more amenable to

analysis.

Since all gates are reversible and each gate within a n line reversible circuit realizes a permutation, each gate realizes an element of the symmetric group and the empty circuit realizes the identity. A growth process on reversible circuits starts with the empty circuit and grows a circuit by iteratively suffixing gates to the circuit; the gates are chosen according to a fixed distribution on a set of allowable gates. Thus, a growth process on reversible circuits corresponds to a random walk on the symmetric group. To determine whether a growth process has a limiting distribution, the shape of the distribution, and to bound the convergence rate, we look to the theory of random walks on groups and graphs.

A Markov chain is a discrete stochastic process that is defined by its set of states, and a stochastic matrix that defines the probability of transitioning from one state to another [MR95]. An ergodic Markov process is one where every state is reachable from every other state in the process; a random walk on a group is an ergodic Markov process. The periodicity of a process is the greatest common denominator of the lengths of all cycles of the corresponding graph induced by the states and transitions of the process. A process is aperiodic if its periodicity is equal to 1. In our case the induced graph is the Cayley graph and assuming that all gates are their own inverses, the periodicity is either 2 or 1.

The underlying analysis of such processes can be traced to the theorems of Frobenius [Fro12] and Perron [Per07], which are used to characterize the eigenvalues of the respective transition matrices. Assuming the respective process is ergodic, the eigenvalues of the corresponding transition matrix have magnitude at most 1; one of the eigenvalues is equal to 1, and the remaining eigenvalues are of magnitude strictly less than 1 if and only if the process is aperiodic [MR95]. If the process has a period of 2, then one other eigenvalue is equal to -1 and the remaining eigenvalues are of magnitude strictly less than 1. A stationary or limiting distribution exists if and only if the process is aperiodic. In Chapter 4, using these criteria, we derive the conditions under which a growth process on reversible circuits has a limiting distribution and show that the distribution is uniform over the entire set of realizable functions.

2.5 Complexity of Reversible Computation

The modern notion of computational complexity dates back to Shannon [Sha49], who formalized the circuit complexity of a function, and to the work of Hartmanis and Stearns [HS65], who formalized the complexity of a function within the framework of Turing machines. The first notions of complexity within the framework of reversible computation came from Lecerf [Lec63], who proposed the notion of a reversible Turing machine, and Landauer [Lan61] who showed that the energy consumption of a reversible computation could be arbitrarily reduced by reducing the speed of the computation itself. However, it wasn't until Bennett's [Ben73] investigation of reversible Turing machines and Toffoli's [Tof80] investigation of reversible circuits that the basic framework of reversible computational complexity was established.

Within the framework of reversible Turing machines the two measures generally considered are space—the amount of tape that a Turing machine uses—and time—the number of steps that a

Turing machine performs. Although these measures are identical to the resources considered in the general complexity framework, their behaviour is governed by an additional set constraint, namely, reversibility.

Within the framework of reversible circuits, the main complexity measures include: circuit *size*, circuit *width*, and circuit *bandwidth*. The size or length of the circuit is the number of steps a reversible circuit performs during the computation. In almost all circuits, each gate depends on the preceding gate in the sequence. Consequently, almost all the gates execute sequentially, and the depth of the circuit—the longest path through the circuit—is comparable to its size; at most the size is a factor of one circuit width greater than the depth. The *width* of a circuit is the total number of lines used by the circuit, and is analogous to the total space used by a Turing machine with a single tape that is used for input, computation, and output. Since this definition is unsatisfactory, for the study of low-lying complexity classes, the *bandwidth* of the circuit is defined as the number of read-write lines in the circuit. This is analogous to the space used by a Turing machine that has a separate tape for the input. Additionally, we generally assume a nonuniform circuit model, i.e., the circuits correspond to Turing machines with some amount of advice, which is a function of the input size.

Unlike in general circuit complexity, where circuit depth and circuit size are often orthogonal measures, in the case of reversible circuits the two measures are for the most part synonymous—the number of gates is comparable to the depth of the circuit. Thus, reversible circuits are most closely related to narrow circuits, such as the ones inherent in the definition of the class $\mathbf{SC}^i$: the set of problems computable by bounded fan-in polynomial size circuits of width $O(\log(n)^i)$ [Coo79]. Reversible circuits can be simulated by general circuits of comparable size—a Toffoli gate can be realized by an AND gate and an XOR gate. Hence, many of the results applicable to general circuits are also applicable to reversible circuits.

Reversible computational complexity attempts to address the same questions as general computational complexity, for example: determining the trade-offs, e.g., space-time, relating the computational power of the model with respect to other computation models, and, determining the complexity of a particular function or class of functions. We quickly review the related work done in these endeavours.

2.5.1 Space-Time and Other Trade-offs

Space-time trade-offs have been a major theme within the framework of reversible Turing machine complexity [Ben73, Ben89, LS90, LMT00]. These trade-offs were also further investigated within the framework of pebbling games [LV96a, LV96b, LV97, LTV98] and information theory [Ben82, Zur89, BGL+93].

Given an irreversible computation using time T and space S, Bennett [Ben73] showed how to reversibly simulate it using time $O(T)$ and space $O(S+T)$ by having the Turing machine keep a history of the computation, which was reversibly erased after the output was computed. One of the major criticisms of this simulation is that the amount of space used is proportional to the length of the computation. Unfortunately, since T could be exponentially larger than S, the space overhead is

considerable. This overhead is mitigated by trading time for space. Instead of recording the history of the entire computation, the computation is divided into segments and the history is kept only for the current segment. At the end of each segment a checkpoint is saved. The history between the last two checkpoints is reversibly erased and used for the history of the next segment. Depending on the size of the segments, a computation may be performed reversibly using $O(S \log T)$ space and $O(T^2)$ time.

Bennett [Ben89] tightened these trade-offs to $O(T^{1+\varepsilon})$ time and $O(S \log T)$ space, by using a hierarchy of segments and checkpoints to perform the simulation. The computation is divided into segments of size m, where $m \approx S$ and the number of segments is c^n, where $c > 1$. The c^n irreversible segments of computation are simulated by $(2c-1)^n$ reversible segments via a recursive computation; a partial simulation for $c = 2$ is illustrated in Figure 2.4. The figure depicts the schedule of creation and deletion of the computation segments. At each stage a computation segment is either reversibly created (listed in the ✔ row) or deleted (listed in the ✘ row); the time during which the ith computation segment exists is represented by the bars in the ith row. Levine and Sherman [LS90] noted that the constants in the time and space bounds depend exponentially on ε, which is proportional to the inverse logarithm of c.

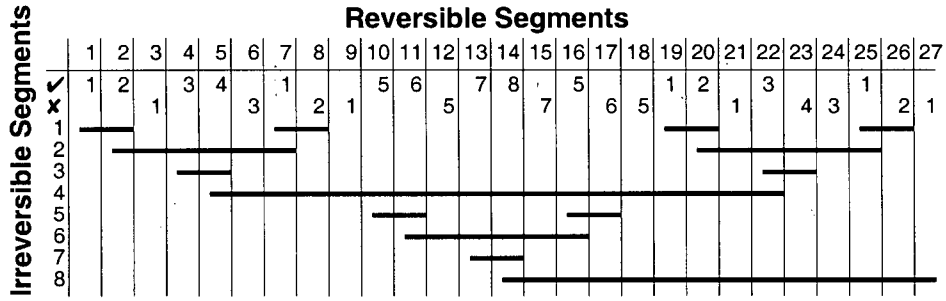


Figure 2.4: A partial representation of the improved simulation for $c = 2$.

Lange, McKenzie, and Tapp [LMT00] demonstrated that a space parsimonious reversible simulation of a deterministic Turing machine M is also possible. Using a variation of the technique used by Sipser [Sip80], and by Cook and McKenzie [CM87], an Euler tour is performed on the configuration graph of M, where the edges are induced by the transitions between configurations. Since M is space bounded, the induced graph is finite and the space used by the simulation is only an additive constant more than the space used by M. The key point is that the tour can be performed reversibly; once a halting configuration is found, the tour stops. Since the number of configurations is potentially exponential, the time of the computation may also become exponential.

The results of Bennett [Ben73] and Lange et al. [LMT00], correspond directly to the observations made about reversible circuits [Tof80, FT82]. Given a general k-ary circuit of depth d and comprising n gates, we can construct an equivalent reversible circuit either by adding $O(n)$ lines (space), thus preserving the depth, or by adding one line and exponentially increasing the depth. A comparable result to the one in [Ben89] is also possible.

2.5.2 Relationships with Other Models of Computation

In our investigation we commonly restrict the bandwidth of the reversible circuits to be either constant, or polylogarithmic in the number of inputs. General circuits that are restricted to polylogarithmic width or constant width are conjectured to only be able to compute problems in the low-lying complexity classes such as **NC** [Pip79] and **SC** [Coo79]. We say "conjectured" because we can't even prove that $\mathbf{NC}^1 \neq \mathbf{NP}$ [Joh94, Page 136].

A related model of computation is that of branching programs [Lee59], in particular permutation branching programs [Bar85]. These programs are very similar to reversible circuits with read-only input lines and read-write additional lines on which all the computation is performed.

Complexity Classes NC, AC, and SC

The class $\mathbf{NC}^i$ (Nick's Class) is the class of problems computable by bounded fan-in circuits of depth $O(\log(n)^i)$ and polynomial size; the classes were named by Cook [Coo79] in honour of Nicholas Pippenger [Pip79]. If we allow gates of unbounded fan-in, i.e., $\{\neg, \wedge_n, \vee_n\}$ where $\wedge_n = \wedge_{i=1}^n x_i$ and $\vee_n = \vee_{i=1}^n x_i$ for all positive n, the class $\mathbf{AC}^i$, also defined in [Ruz79, Coo85], is the class of functions computable by unbounded fan-in circuits of depth $O(\log(n)^i)$ and polynomial size. A direct implication of the definitions is that $\mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$. The class **NC** (**AC**) is defined as the union of $\mathbf{NC}^i$ ($\mathbf{AC}^i$) for all i [Joh94].

The class $\mathbf{SC}^i$ (Steve's Class) is the class of problems computable by bounded fan-in circuits of width $O(\log(n)^i)$ and polynomial size; the classes were named after Stephen Cook who first studied them in [Coo79]. The class **SC** is the union of $\mathbf{SC}^i$ over all i. The class $\mathbf{L}_{/\text{poly}}$, nonuniform logarithmic space, is the class of problems solvable by a logarithmic space bounded automata with a polynomial amount of advice; note that $\mathbf{L}_{/\text{poly}} = \mathbf{SC}^1$.

Branching Programs

A *branching program* (BP), introduced by Lee [Lee59], is a rooted directed acyclic graph comprising interior nodes (out-degree 2) and two leaf nodes (out-degree 0). Each interior node is labeled by a variable, x_i, and its two outgoing arcs are labeled 0 and 1. The two leaf nodes are also labeled 0 and 1. The computation begins at the root and traverses the graph along the arcs; at each interior node the computation follows the arc whose label is the value of the corresponding variable and terminates at leaf node, outputting the leaf's label.

The depth of a node is the length of the longest path from the root to the node and the depth of a branching program is the maximum depth of the two leaf nodes. A branching program is *synchronized* (leveled) if the difference in depth between any two adjacent nodes is exactly one, and its width is the maximum number of nodes at the any depth. A width-w *permutation branching program* (w-PBP) is a synchronized width-w branching program such that the labels of all arcs incident on the same interior node differ. The class $\mathbf{P}_{BP}$ is the class of problems that are solvable by polynomial-size branching programs and the class $\mathbf{P}_{BP}^w$ is the class of problems solvable by polynomial-size, width-w branching programs.

There have been significant attempts to derive lower bounds for branching programs, particularly bounded-width branching programs [CFL83, BDFP83, Pud84, BS95] and polylogarithmic-width branching programs [ABH+86]. Perhaps the best known result is that of Barrington [Bar89] who showed that $\mathbf{NC}^1$ is equal to the class of functions computed by polynomial-size bounded-width permutation branching programs.

A w-PBP B can be formulated as a sequence of instructions where the ith instruction is a tuple (j_i, f_i, g_i), where $j_i \in \mathbb{Z}_n$ identifies one of the n inputs $\{x_1, x_2, ..., x_n\}$, and f and g are permutations on $\mathbb{Z}_w$. If x_{j_i} is true, the ith instruction yields permutation f_i, otherwise it yields g_i. The output, $B(x)$, of program B on input x, is the product of permutations yielded by the instructions of B. Program B accepts language L if there is a permutation $\sigma \neq \varepsilon$ such that

$$B(x) = \begin{cases} \sigma & x \in L \\ \varepsilon & x \notin L \end{cases}.$$

Using permutations from the alternating group A_5, Barrington's [Bar89] construction simulates an AND operation using a commutator and inductively composes a polynomial size 5-PBP that realizes any function in $\mathbf{NC}^1$. The other direction follows from the fact that the computing the product of polynomial number of elements from a fixed finite group is in $\mathbf{NC}^1$.

Interestingly, 3-PBPs are not as powerful as 5-PBPs; computing the conjunction requires an exponentially long program [Bar85]. Not surprisingly, there has been significant work done to investigate the reasons for the disparity between 3-PBPs and 5-PBPs, and to generalize the lower bounds for bounded-width branching programs and programs over groups [BT88, BST90, BS95]. Perhaps the most fascinating problem is determining if there exists a polynomial size 4-PBP that can compute the conjunction over n variables. Barrington et al. [BST90] have proven that the answer is negative if the instructions of the program are restricted to the alternating group A_4, i.e., the program is over the group A_4. Intriguingly, the problem for general 4-PBPs remains open!

2.5.3 Related Work

For completeness, we mention several related frameworks in which reversible computation has been studied. These include: reversible straight line programs [Cle90], space-time-reversibility tradeoffs using a pebbling model [LV96b, LTV98, BTV01], and information theory based analysis of reversible computation [Ben82, Zur89, BGL+93, LV96a].

Reversible Programs

In his doctoral thesis, Cleve [Cle90] investigated reversible computation using a straight line program model. Following the same formalism, a statement of the a program is denoted by

$$R_i \leftarrow f(R_{j_1}, R_{j_2}, ..., R_{j_k}),$$

where f is a function over some ring $\mathcal{R}$ that operates on k inputs from registers, and stores the result in another register, not necessarily distinct from one of the arguments; a register may hold

any element of $\mathcal{R}$. Each statement induces a map from $\mathcal{R}^n$ to $\mathcal{R}^n$ and is reversible if the map is a bijection. The statements are usually drawn from a basis, such as

$$\begin{aligned} R_i &\leftarrow c - R_i,\ c \in \mathcal{R} \\ R_i &\leftarrow (R_j \cdot R_k) - R_i,\ j,k \neq i, \end{aligned}$$

and is reversible if all its statements are reversible. Another concrete example is the basis

$$\begin{aligned} R_i &\leftarrow 1 \oplus R_i \\ R_i &\leftarrow (R_j \wedge R_k) \oplus R_i,\ j,k \neq i, \end{aligned}$$

over the field $\mathbb{Z}_2$, which computes the negation and the Toffoli gates respectively.

Reversible Pebbling

A technique for proving space-time trade-offs is pebbling [Pip80]. The pebble game consists of a connected digraph G and a fixed number of pebbles. Play consists of placing pebbles on and removing pebbles from the vertices of G using the following rules: a pebble may be removed from a vertex at any time, and a pebble may be placed on a vertex only if all of its predecessors have a pebble on them or if the vertex is a start vertex. The game is won only if every vertex is visited. The pebbles represent registers (space) and the vertices represent steps in the computation. The total number of moves—removing or placing a pebble (erasing or storing a register)—corresponds to the length of the computation. The number of pebbles versus the number of moves corresponds to the space-time trade-off.

Using the pebble game, Cook [Coo74], Ladner [Lad75], Hopcroft et al. [HPV75], and others proved space-time trade-offs and completeness results. Additionally, the first space-time trade-off, which was due to Paterson and Hewitt [PH70], could be cast in the framework of a pebbling game [Pip80]. In 1996, Li and Vitányi [LV96b], and Li et al. [LTV98] used pebbling to show that Bennett's [Ben89] reversible simulation of an irreversible computation is, given some assumptions, optimal.

The reversible pebble game uses a modified removal rule: a pebble cannot be removed from a vertex unless all of the predecessors of the vertex have pebbles on them. In essence, information cannot be erased, it may only be cancelled out. The simulation graph is a singly linked chain of T vertices, with each vertex representing a step in the irreversible computation. The main result is that if there are n pebbles and $T > 2^n$, then there is no winning strategy; the computation cannot be simulated. Additionally, if E erasures are allowed, then the computation can be performed with $n - \log(E+1)$ pebbles where E is odd; for each missing pebble exponentially more erasures must be performed. Buhrman et al. [BTV01] extended the model to accommodate the result of Lange et al. [LMT00].

Reversibility and Information Theory

Bennett et al. [BGL+93] and, Li and Vitányi [LV96a], make the observation that the amount of erasure performed during a computation is a measure of its irreversibility. The irreversibility cost

$E_3(x,y)$ of reversibly computing x from y, defined by Zurek [Zur89] and discussed by Bennett et al. [BGL+93], is the number of source and garbage bits required to reversibly perform the computation. They show that $E_3(x,y) = K(x|y) + K(y|x)$ up to an additive logarithmic term, where $K(x|y)$ is Kolmogorov complexity of x given y [LV97]. This measure highlights a striking relationship between the information content of strings and the resources required to compute them reversibly.

Chapter 3

Growth Processes on Formulas

In this chapter we investigate growth processes on formulas. We classify growth processes primarily by the type of connective used by the process. For example, one class is the class of growth processes that use linear connectives. The remaining parameters, such as the initial distribution, the number of variables, and the specific nature of the connective, are considered simply as parameters of the class of growth processes. We characterize several classes of growth processes, including growth processes that use linear connectives, self-dual connectives, and monotone connectives. We derive criteria that determine if a limiting distribution exists, the shape of the limiting distribution, and the rate at which the limiting distribution is approached. We first cover the basic definitions before proceeding to our results.

3.1 Definitions

Let B_n denote the Boolean cube of size n and let the **weight** of a Boolean vector $x \in B_n$, denoted $|x|$, be the number of 1 bits in x. For any two Boolean vectors $x, y \in B_n$, we say that $x \leq y$ if $x_i \leq y_i$, for all $i = 1 \ldots n$; we say that $x < y$ if $x \leq y$ and $x_i < y_i$ for at least one $i \in [1, n]$.

A Boolean function is said to be **linear** if it can be represented as the sum of binary variables and constants modulo 2. A Boolean function is said to be **self-dual** if it satisfies the equation $f(x_1, \ldots x_n) = \overline{f(\bar{x}_1, \ldots, \bar{x}_n)}$. A Boolean function f is said to be **monotone** if for all assignments x and y, $x \leq y$ implies that $f(x) \leq f(y)$. A Boolean function is **balanced** if exactly half of the entries in its truth table are ones, and the rest are zeros. Let $\mathcal{F}_n$ denote the family of n-adic Boolean functions, let $\mathcal{M}_n$ denote the family of n-adic monotone Boolean functions, and let $\mathcal{L}_n$ denote the family of n-adic linear functions.

Let k be a positive integer and α be a k-adic Boolean function, which we call the **connective**. A **growth process** is denoted by a pair (μ, α), where μ is a distribution on $\mathcal{F}_n$ and α is a connective. Distribution μ is called the **initial distribution** and is uniformly distributed on a subset of the projections, their negations, and constants. This subset contains the n projections $\{x_1, \ldots, x_n\}$; it may contain their negations $\{\bar{x}_1, \ldots, \bar{x}_n\}$, and it may contain neither, one, or both of the constants $\{0, 1\}$.

A growth process induces a sequence of probability distributions π_i on $\mathcal{F}_n$ for each $i \geq 0$ in the following way. We take $\pi_0 = \mu$ and for $i \geq 1$, $\pi_i(f)$ is the probability that $\alpha \circ (\tilde{g}_1, \ldots, \tilde{g}_k) =$

f, where $\tilde{g}_1,\ldots,\tilde{g}_k$ are independent random variables with distribution π_{i-1} and $\alpha\circ(g_1,\ldots,g_k)$ is the composition of α with the Boolean functions $g_1,\ldots,g_k$, which for conciseness is denoted $\alpha(g_1,\ldots,g_k)$. The ith iteration of the growth process induces the probability distribution π_i on $\mathcal{F}_n$.

The **support** of a probability distribution π, denoted $supp(\pi)$, is the set $\{f : \pi(f) > 0\}$ and the **support** of a growth process is the set of all functions $f \in \mathcal{F}_n$ for which $\pi_i(f) > 0$ for some $i > 0$: $\cup_i supp(\pi_i)$.

If π_i tends to π, as i approaches infinity, distribution π is called the **limiting distribution** of the growth process. We say that a growth process **converges** to a limiting distribution if a limiting distribution π exists such that π_i tends to π as i approaches infinity. A process **converges rapidly** if for $i > C_\alpha \log(n)$, $\max_f |\pi(f) - \pi_i(f)| < 2^{-n}$, where C_α is a constant that depends only on α. (Unless otherwise stated, the base of the logarithm is assumed to be 2.)

There are also cases—which we shall see later—in which π_{2i} and π_{2i+1} tend to distinct **alternating limiting distributions**. When a limiting distribution exists, we can have $\pi(f) > 0$ only for f in the support of the growth process. As Valiant's [Val84] result indicates, however, there may be functions in the support for which $\pi_i(f)$ tends to zero as i approaches infinity, so that $\pi(f) = 0$. The **asymptotic support** of a growth process with a limiting distribution π is the set $supp(\pi)$. Informally, the "result of a growth process" refers to the existence, the identity, and the convergence to, the limiting or alternating distributions.

If α is a k-adic connective, the **characteristic polynomial** of α is

$$A_\alpha(p) = \sum_{i=0}^{n} \beta_i \binom{n}{i} p^i (1-p)^{n-i}$$

where β_i is the fraction of assignments of weight i for which α is true:

$$\beta_i = \frac{|\{x \in B_k : |x| = i,\ \alpha(x) = 1\}|}{\binom{n}{i}}.$$

If the arguments of α are assigned independent random binary variables, $\tilde{x} = \tilde{x}_1,\ldots,\tilde{x}_k$, that are 1 with probability p and 0 with probability $1-p$, then $\Pr[\alpha(\tilde{x}) = 1] = A_\alpha(p)$. The ***i*th composition** of $A_\alpha(p)$ with itself is denoted by $A^i_\alpha(p) = A_\alpha(A_\alpha(\ldots A_\alpha(p)\ldots))$.

The Fourier transform Δ_i of a probability distribution π_i is defined as

$$\Delta_i(w) = \sum_{f\in\mathcal{F}_n} (-1)^{\langle w,f\rangle} \pi_i(f). \tag{3.1}$$

For convenience, the inner product $\langle w,f\rangle = \sum_i w_i f_i$ is defined to be over the integers, rather than over $\mathbb{Z}_2$, and unless otherwise noted, Boolean n-adic functions are represented by their truth tables, namely, as Boolean vectors from B_{2^n}. Since the probabilities must sum to 1, $|\Delta_i(w)| \le 1$, for all $w \in \mathcal{F}_n$, $\Delta_i(\mathbf{0}) = 1$, and the inverse Fourier transform is

$$\pi_i(f) = \frac{1}{2^{2^n}} \sum_{w\in\mathcal{F}_n} (-1)^{\langle f,w\rangle} \Delta_i(w). \tag{3.2}$$

The Fourier transform plays a role in many of our results, but needs to be adapted in various ways to suit different cases. When dealing with linear functions, for example, we represent the

functions not by their truth-tables, but rather by their coefficients as multivariate polynomials over $\mathbb{Z}_2$. However, unless otherwise noted, the standard representation is assumed.

3.2 Growth Processes that Use Linear Connectives

A function f is linear if it is of the form $f(x_1,\ldots,x_n) = c_0 \oplus c_1x_1 \oplus \cdots \oplus c_nx_n$ for some constants $c_0, c_1, \ldots, c_n \in \mathbb{Z}_2$. We may assume without loss of generality that α depends on all its arguments, so that $\alpha(y_1,\ldots,y_k) = c \oplus y_1 \oplus \cdots \oplus y_k$, where $k \geq 2$. The result of the growth process depends on the support of of the initial distribution μ, the parity of k, and the constant term c. This claim is proven in Theorem 3.2, which considers supports of μ that do not contain negations of projections, and in Theorem 3.3, which considers supports of μ that do contain negations of projections. For all cases where a limiting distribution exists, Theorem 3.4 bounds the convergence rate of π_i to the limiting distribution. Thus, we completely characterize growth processes that use linear connectives.

To begin, we derive a recurrence for the Fourier coefficients of the respective probability distributions π_i, from which we derive the limiting distribution. Since compositions of linear functions are themselves linear, we represent the linear functions by the vector $(c_0, c_1, \ldots, c_n)$ of their coefficients, and the class of linear functions $\mathcal{L}_n$ is represented by the Boolean cube B_{n+1}. Finally, let w_1 denote the constant function 1 ($w_1 = 100\ldots0$), whereas $\mathbf{1} = 11\ldots1$.

Proposition 3.1 *Let (μ, α) be a growth process, where α is a k-adic linear connective and let $w \in \mathcal{L}_n$. The Fourier coefficients of the probability distribution π_i of the growth process are described by the recurrence relation*

$$\Delta_{i+1}(w) = (-1)^{c\langle w_1, w\rangle}\Delta_i(w)^k,$$

where c is the constant term of the connective.

Proof:

$$\begin{aligned}
\Delta_{i+1}(w) &= \sum_{f\in\mathcal{L}_n} \pi_{i+1}(f)(-1)^{\langle f,w\rangle} = \sum_{f\in\mathcal{L}_n} \sum_{\substack{\mathbf{g}\in\mathcal{L}_n^k \\ \alpha(\mathbf{g})=f}} \prod_{j=1}^{k} \pi_i(\mathbf{g}_j)(-1)^{\langle f,w\rangle} \\
&= \sum_{\mathbf{g}\in\mathcal{L}_n^k} \prod_{j=1}^{k} \pi_i(\mathbf{g}_j)(-1)^{\langle \alpha(\mathbf{g}),w\rangle} = \sum_{\mathbf{g}\in\mathcal{L}_n^k} \prod_{j=1}^{k} \pi_i(\mathbf{g}_j)(-1)^{\langle cw_1 \oplus \bigoplus_{j=1}^{k} \mathbf{g}_j, w\rangle} \\
&= \sum_{\mathbf{g}\in\mathcal{L}_n^k} \prod_{j=1}^{k} \pi_i(\mathbf{g}_j)(-1)^{\langle cw_1,w\rangle \oplus \bigoplus_{j=1}^{k} \langle \mathbf{g}_j,w\rangle} = (-1)^{\langle cw_1,w\rangle} \sum_{\mathbf{g}\in\mathcal{L}_n^k} \prod_{j=1}^{k} \pi_i(\mathbf{g}_j)(-1)^{\langle \mathbf{g}_j,w\rangle} \\
&= (-1)^{\langle cw_1,w\rangle}\Delta_i(w)^k.
\end{aligned}$$

In the last step of the derivation, the sum is expanded into a telescopic sum comprising k nested sums that collapse into the final result. ■

Using Proposition 3.1, the following theorems classify the growth processes on linear connectives.

Theorem 3.2 *Let (μ, α) be a growth process, where $\alpha(y) = c \oplus y_1 \oplus \cdots \oplus y_k$, $k > 1$, and the support of μ does not contain negations of the projections.*

1. *If $\{0,1\} \cap supp(\mu) \neq \{0,1\}$, k is odd and $c = 1$, then the growth process has alternating limiting distributions, each of which is uniform over one half of the support of the growth process (which consists of all linear functions for which $\bigoplus_{j=1}^{n} c_j = 1$).*

2. *In all other cases, the limiting distribution is uniform over the support of the growth process (which depends on k, c, and the presence of constants in the support).*

Proof: Two facts are key to this theorem: first, that $|\Delta_i(w)| \leq 1$, and second, that if $|\Delta_i(w)| < 1$, then $\lim_{i \to \infty} \Delta_i(w) = 0$. Only the nonzero (magnitude 1) coefficients contribute to limiting distribution (equation 3.2); fortunately, these are determined solely by the support of the initial distribution. Depending on which constants are part of the support, there are either one, two, or four coefficients of magnitude 1:

$$\begin{aligned}
\{0,1\} \cap supp(\mu) = \{0,1\} &\Rightarrow \Delta_0(\mathbf{0}) = 1, \\
\{0,1\} \cap supp(\mu) = \{0\} &\Rightarrow \Delta_0(\mathbf{0}) = \Delta_0(w_1) = 1, \\
\{0,1\} \cap supp(\mu) = \{1\} &\Rightarrow \Delta_0(\mathbf{0}) = 1,\ \Delta_0(\mathbf{1}) = -1, \\
\{0,1\} \cap supp(\mu) = \emptyset &\Rightarrow \Delta_0(\mathbf{0}) = \Delta_0(w_1) = 1,\ \Delta_0(\mathbf{1}) = \Delta_0(w_1 \oplus \mathbf{1}) = -1.
\end{aligned}$$

If k is odd and $c = 1$, the recurrence from Proposition 3.1 implies that $\Delta_{i+1}(w_1) = -\Delta_i(w_1)$ and $\Delta_{i+1}(\mathbf{1}) = -\Delta_i(\mathbf{1})$. Hence, if $supp(\mu) \cap \{0,1\} \neq \{0,1\}$, the resulting distribution is alternating. In the case where one of the constants is missing from the support, only two coefficients have magnitude 1, and thus, the alternating distributions are each uniform over half of $\mathcal{L}_n$. Otherwise, if both constants are missing, the alternating distributions are each uniform over one quarter of $\mathcal{L}_n$.

If $c = 0$, k is even, or $\{0,1\} \cap supp(\mu) = \{0,1\}$, the limiting distribution exists because the sign of the magnitude 1 coefficients does not alternate. We can read off the limiting distribution from the Fourier coefficients. If both constants are in the support, then the limiting distribution is uniform over $\mathcal{L}_n$. If only one of the constants is present, then the distribution is uniform over half of $\mathcal{L}_n$, and if neither is present, then the distribution will be uniform over a quarter of $\mathcal{L}_n$. ■

If the support of μ contains negations of projections, then using the same proof technique yields the following theorem.

Theorem 3.3 *Let (μ, α) be a growth process, where $\alpha(y) = c \oplus y_1 \oplus \cdots \oplus y_k$, $k > 1$, and the support of μ contains negations of the projections.*

1. *If $\{0,1\} \cap supp(\mu) = \emptyset$ and k is odd then the limiting distribution is uniform over all linear functions of odd number of variables.*

2. *If $\{0,1\} \cap supp(\mu) = \emptyset$ and k is even then the limiting distribution is uniform over all linear functions of even number of variables.*

3. *Otherwise, the limiting distribution is uniform over all of $\mathcal{L}_n$.*

Proof: If $\{0,1\} \cap supp(\mu) \neq \emptyset$, then there is only one coefficient of magnitude 1, $\Delta_0(\mathbf{0}) = 1$, implying the last case.

Otherwise, there is one other magnitude 1 coefficient, $\Delta_0(\mathbf{1} \oplus w_1) = -1$. If k is odd, then $\Delta_{i+1}(\mathbf{1} \oplus w_1) = \Delta_i(\mathbf{1} \oplus w_1)^k = -1$, implying the first case of the theorem. If k is even, then $\Delta_{i+1}(\mathbf{1} \oplus w_1) = \Delta_i(\mathbf{1} \oplus w_1)^k = 1$, implying the second case. ■

Note, that if projection negations are present, no alternating distribution can occur. To show that the convergence of π_i to π is rapid we use the inverse Fourier transform.

Theorem 3.4 *Let (μ, α) be a growth process, where α is a k-adic linear connective, $k > 1$, and the support of μ contains the n projections. If the growth process has a limiting distribution π and $i > \frac{2\log(n)}{\log(k)}$, then for any linear function f, $|\pi(f) - \pi_i(f)| < 2^{-n}$.*

Proof: Let $D = \{w : |\Delta_0(w)| < 1\}$. Then $\pi_i(f)$ may be written as:

$$\begin{aligned}
\pi_i(f) &= 2^{-n-1} \sum_{w \in B_{n+1}} (-1)^{\langle w,f \rangle} \Delta_i(w) \\
&= 2^{-n-1} \sum_{w \notin D} (-1)^{\langle w,f \rangle} \Delta_i(w) + 2^{-n-1} \sum_{w \in D} (-1)^{\langle w,f \rangle} \Delta_i(w) \\
&= \pi(f) + 2^{-n-1} \sum_{w \in D} (-1)^{\langle w,f \rangle} \Delta_i(w).
\end{aligned}$$

Thus, for any linear function f,

$$|\pi(f) - \pi_i(f)| = |2^{-n-1} \sum_{w \in D} (-1)^{\langle w,f \rangle} \Delta_i(f)| \leq \max_{w \in D} |\Delta_0(w)|^{k^i} \leq (1 - n^{-1})^{k^i}.$$

Solving inequality $(1 - n^{-1})^{k^i} < 2^{-n}$, in terms of i, yields: $i > \frac{2\log(n)}{\log(k)}$. ■

3.3 Growth Processes that Use Self-Dual Connectives

Savický [Sav90] showed that if the connective is balanced (that is, if it assumes the value 1 for just one-half of the combinations of argument values) and nonlinear, and the distribution μ is uniform over all the projections, their negations, and constants, then the limiting distribution will be uniform over all of $\mathcal{F}_n$. If we remove the constants from the support of μ and assume the connective α is self-dual, then the support of the growth process is the set of all self-dual functions. In this case the limiting distribution of the growth process is uniform over this support. Unfortunately, bounding the rate of convergence of these growth processes appears to be too difficult.[1]

Theorem 3.5 *If the connective is nonlinear and self-dual, and the support of μ comprises the projections and their negations, then the limiting distribution will be uniform over the family of self-dual n-adic functions.*

[1] Savický [Sav95a] did derive bounds for the growth process discussed in [Sav90] (see Theorem 3.27), these bounds are not applicable because the Fourier coefficients of weight 2 functions are not guaranteed have magnitude strictly less than 1.

Proof: Observe that there is a bijection between the set of all functions on n variables and the set of self-dual functions on $n+1$ variables, for example, the map

$$f(x_1,x_2,\ldots,x_n) \mapsto f(x_1,x_2,\ldots,x_n)x_{n+1} \vee \overline{f(\bar{x}_0,\bar{x}_1,\ldots,\bar{x}_n)}\bar{x}_{n+1}.$$

The result follows. ■

3.4 Growth Processes that use Monotone Connectives

We now focus on growth processes that use monotone connectives. For the rest of this section we assume that α is monotone and the support of μ contains only the projections and possibly constants. We first investigate growth processes that use unbalanced connectives and show that the result of such growth processes is a limiting distribution that is concentrated on a threshold function (Theorem 3.13). We show that except in one case (Theorem 3.17), the convergence to the limiting distribution is rapid (Theorem 3.16 and Theorem 3.19). We then investigate growth processes on balanced connectives and show that if the number of arguments, n, is even, then the distribution is not concentrated on a threshold function (Theorem 3.26); in Theorem 3.34, we also bound the rate of convergence.

3.4.1 Growth Processes that Use Unbalanced Monotone Connectives

Growth processes that use unbalanced monotone connectives concentrate probability on a threshold function; the type of threshold function depends on the connective and the support. A threshold function $T_k(x_1,\ldots,x_n)$ assumes the value 1 if and only if at least k of its n arguments assume the value 1. We consider constant functions $T_{n+1} = 0$ and $T_0 = 1$ to be special cases of threshold functions. The following sequence of propositions and lemmas culminate in a formal statement of this claim, Theorem 3.13. There are two cases to consider: first, when the characteristic polynomial of α, $A_\alpha(p)$, has no fixed-point on the open interval $(0,1)$, and second, when $A_\alpha(p)$ has a fixed-point on $(0,1)$.

Proposition 3.6 *Let (μ,α) be a growth process, where α is a monotone connective whose characteristic polynomial, $A(p)$, has no fixed-point on the interval $(0,1)$. Then the limiting distribution will be concentrated on a threshold function.*

Proof: Since $A(p)$ has no fixed-point on $(0,1)$, either $A_\alpha(p) < p$ throughout $(0,1)$, or $A_\alpha(p) > p$ throughout $(0,1)$. If $A_\alpha(p) > p$ throughout $(0,1)$, then by the standard amplification argument—see Section 2.2.1 and Theorem 3.16—the limiting distribution is concentrated on T_1 (disjunction of all variables), or T_0 if 1 is in the support of μ. Similarly, if $A_\alpha(p) < p$ throughout $(0,1)$, then the limiting distribution is concentrated on T_n (conjunction of all variables), or T_{n+1} if 0 is in the support of μ. ■

Furthermore, we can easily describe connectives whose characteristic polynomials have no fixed-points on $(0,1)$.

Lemma 3.7 *If $A_\alpha(p) \neq p$ on the interval $(0,1)$, then $\alpha(x) = x_i \vee \alpha'(x)$ (when $A_\alpha(p) > p$), or $\alpha(x) = x_i \wedge \alpha'(x)$ (when $A_\alpha(p) < p$).*

Proof: If $\alpha(x) \neq x_i \vee \alpha'(x)$, by Lemma 3.9, $A_\alpha(p) = O(p^2)$ which implies that there exists a positive constant ε_0 such that for all $0 < \varepsilon < \varepsilon_0$, $A_\alpha(\varepsilon) < \varepsilon$. Similarly, if $\alpha(x) \neq x_i \wedge \alpha'(x)$, then by duality, $1 - A_\alpha(1-p) = O(p^2)$, which means that $A_\alpha(1-\varepsilon) > 1-\varepsilon$ for all $0 < \varepsilon < \varepsilon_1$ for some $\varepsilon_1 > 0$. Since $A_\alpha(p)$ is continuous, there must be a fixed-point in $(0,1)$, which is a contradiction. ■

In the second case, where $A_\alpha(p)$ has a fixed-point in $(0,1)$, Moore and Shannon [MS56] have shown that this fixed-point is unique. Not surprisingly, the limiting distribution depends on the fixed-point. Thus, we first derive two facts about the fixed-point of the characteristic polynomial.

Lemma 3.8 *The characteristic polynomial $A_\alpha(p)$ has a fixed-point of $\frac{1}{2}$ if and only if the connective α is balanced.*

Proof: By definition $\sum_{i=0}^{n} \beta_i \binom{n}{i}$ is the number of assignments for which α is true. If $A_\alpha(\frac{1}{2}) = \frac{1}{2}$, then $A_\alpha(\frac{1}{2}) = \sum_{i=0}^{n} \beta_i \binom{n}{i} (\frac{1}{2})^i (\frac{1}{2})^{n-i} = \frac{1}{2^n} \sum_{i=0}^{n} \beta_i \binom{n}{i} = \frac{1}{2}$. Hence, $\sum_{i=0}^{n} \beta_i \binom{n}{i} = 2^{n-1}$ which means that α is balanced. Conversely, if α is balanced, then $A_\alpha(\frac{1}{2}) = \frac{1}{2}$. ■

Lemma 3.9 *If α is a monotone connective that is not of the form $\alpha(x) = x_i \vee \alpha'(x)$, then on the interval $(0,1)$, $A_\alpha(p) < (\binom{k}{2} + 1)p^2$.*

Proof: Since α is not of the form $\alpha(x) = x_i \vee \alpha'(x)$, the first two coefficients of $A_\alpha(p)$ are $\beta_0 = 0$, $\beta_1 = 0$. Thus, on the interval $(0,1)$,

$$A_\alpha(p) = \beta_2 \binom{k}{2} p^2 + B(p)p^3 \leq \binom{k}{2} p^2 + B(p)p^2 < (\binom{k}{2} + 1)p^2$$

since $B(p) < 1$ on the interval $(0,1)$. ■

Fact 3.10 *If α is a nonconstant monotone connective, then:*

1. *$A_\alpha(p) \geq p$ on the interval $(0,1)$ if and only if for some i, $\alpha(x) = x_i \vee \alpha'(x)$ if and only if $\beta_1 > 0$.*
2. *$\beta_0 = 0$, $\beta_k = 1$ and $\beta_{k-1} = \frac{a}{k}$, for some $a \in [0,k]$.*
3. *$A_\alpha(s) = s$, $s \in (0,1)$, $A_\alpha(p) < p$, $p \in (0,s)$ and $A_\alpha(p) > p$, $p \in (s,1)$ if and only if $A'_\alpha(0) = A'_\alpha(1) = 0$ if and only if $\beta_1 = 0$ and $\beta_{k-1} = 1$.*

Proof: First, the coefficient β_1 is the fraction of singleton assignments on which α is 1. Since α is also monotone, $\beta_1 > 0$ if and only if $\alpha(x) = x_i \vee \alpha'(x)$ for at some x_i; the first part of the equivalence follows from Lemma 3.7.

Second, since α is nonconstant, the fractions of weight 0 and weight k assignments for which it is 1, is exactly 0 and 1, respectively; there are also exactly k assignments of weight $k-1$.

Third, if A_α has a fixed-point s on the interval $(0,1)$, then by part one, $\beta_1 = 0$. Also, $\beta_{k-1} = 1$, otherwise α is 0 on at least one assignment of weight $k-1$ and can be written as $\alpha(x) = x_i \wedge \alpha'(x)$. Consequently, by Lemma 3.7, A_α does not have a fixed-point on the interval $(0,1)$, which is a

contradiction. If $\beta_1 = 0$ and $\beta_{k-1} = 1$, then α takes neither of the two forms of Lemma 3.7 and hence, must have a fixed-point on the interval $(0,1)$. Finally, since the derivative of A_α is

$$A'_\alpha(p) = \beta_1 k(1-p)^{k-1} + kp^{k-1}(1-\beta_{k-1}) + p(1-p)B(p),$$

where $B(p)$ is some polynomial of p. Therefore, $A'_\alpha(0) = A'_\alpha(1) = 0$ if and only if $\beta_1 = 0$ and $\beta_{k-1} = 1$. ■

Lemma 3.11 *If $A_\alpha(p)$ is a characteristic polynomial of monotone connective α, such that $A(p)$ is not identically p, then any fixed-point of $A_\alpha(p)$ on $(0,1)$ is either irrational or $\frac{1}{2}$.*

Proof: By contradiction; without loss of generality assume that the fixed-point $p_0 = \frac{r}{s} < \frac{1}{2}$ and $\gcd(r,s) = 1$; if $p_0 > \frac{1}{2}$ we consider $A_\alpha(1-p)$, whose fixed-point is $1 - p_0 < \frac{1}{2}$. Hence,

$$A_\alpha\left(\frac{r}{s}\right) = \sum_{j=0}^{k} \beta_j \binom{k}{j} \left(\frac{r}{s}\right)^j \left(\frac{s-r}{s}\right)^{k-j} = \frac{r}{s}.$$

Multiplying both sides by s^k, noting from Fact 3.10 that $\beta_k = \beta_{k-1} = 1$, and evaluating the result modulo $(s-r)^2$ yields

$$\begin{aligned} rs^{k-1} &\equiv \sum_{j=0}^{k} \beta_j \binom{k}{j} r^j (s-r)^{k-j} \equiv r^k + kr^{k-1}(s-r) + (s-r)^2 \sum_{j=0}^{k-2} \beta_j \binom{k}{j} r^j (s-r)^{k-j-2} \\ &\equiv r^k + kr^{k-1}(s-r) \bmod (s-r)^2. \end{aligned}$$

Evaluating the left side modulo $(s-r)^2$ yields

$$\begin{aligned} rs^{k-1} &\equiv r(r+(s-r))^{k-1} \equiv r \sum_{i=0}^{k-1} \binom{k-1}{i} r^i (s-r)^{k-1-i} \\ &\equiv rr^{k-1} + r(k-1)r^{k-2}(s-r) + r(s-r)^2 \sum_{j=0}^{k-3} \binom{k-1}{j} r^j (s-r)^{k-3-j} \\ &\equiv r^k + (k-1)r^{k-1}(s-r) \bmod (s-r)^2. \end{aligned}$$

Therefore,

$$r^{k-1}(s-r) \equiv 0 \bmod (s-r)^2.$$

Since $\gcd(r,s) = \gcd(r,(s-r)^2) = 1$, $r^{k-1} \not\equiv 0 \bmod (s-r)^2$; this is a contradiction. ■

Theorem 3.12 *Let (μ,α) be a growth process, where α is a monotone unbalanced connective whose characteristic polynomial has a fixed-point $t \in (0,1)$, and the support of μ contains only the projections. The limiting distribution of the growth process is concentrated on the threshold function $T_{\lceil tn \rceil}$.*

Proof: Since α is unbalanced and has a fixed-point on $(0,1)$, by Lemma 3.8, the fixed-point is not $\frac{1}{2}$. Hence, by Lemma 3.11, the fixed-point is irrational. Since the fraction of variables set to true in any assignment is by definition rational, the fraction will always be strictly greater or strictly less than the fixed-point t. Hence, by the standard amplification argument, the limiting distribution will be concentrated on the threshold function $T_{\lceil tn \rceil}$. ■

Theorem 3.12 can easily be modified to cover the cases in which one or both constants are in the support of μ. Combining proposition 3.6 and theorem 3.12 proves the initial claim:

Theorem 3.13 *If (μ, α) is a growth process, where α is a monotone unbalanced connective and the support of μ does not contain the negations of projections, then the limiting distribution will be concentrated on a threshold function.*

Convergence Bounds

Except in one case, all these growth processes converge rapidly to their limiting distribution. In the exceptional case the convergence requires $C_\alpha n^k$ iterations where k is the arity of the connective α; we provide specific criteria that determine whether a process will converge rapidly or not. There are two main cases: either $A_\alpha(p)$ has a fixed-point, or not. We first derive bounds for the latter case, and then for the former. Unless explicitly stated, we assume that constants are not in $supp(\mu)$, however, the following analysis changes little if constants are in $supp(\mu)$.

When $A_\alpha(p)$ has no Fixed-point on $(0,1)$

In this case, either $A_\alpha(p) > p$ for $p \in (0,1)$, or $A_\alpha(p) < p$, for $p \in (0,1)$. Since the two cases are symmetric, the same bounds apply to both. Hence, without loss of generality, assume that $A_\alpha(p) < p$ on the interval $(0,1)$.

Lemma 3.14 *If α is a monotone connective such that $A_\alpha(p) < p$ on the interval $(0,1)$ and, $A_\alpha(p)$ has degree $k > 2$ and $\beta_{k-1} \leq \frac{k-2}{k}$, then $A'_\alpha(1-\varepsilon) > \frac{35}{24}$ for all positive $\varepsilon < \varepsilon_k = \frac{1}{k2^{k+1}}$.*

Proof: Begin by differentiating $A_\alpha(p)$:

$$\begin{aligned} A'_\alpha(p) &= \frac{d}{dp}\left(\sum_{i=0}^{k} \beta_i \binom{k}{i} p^i (1-p)^{k-i}\right) \\ &= \frac{d}{dp}\left(p^k + \beta_{k-1} k p^{k-1}(1-p) + \sum_{i=0}^{k-2} \beta_i \binom{k}{i} p^i (1-p)^{k-i}\right) \\ &= kp^{k-1} + \beta_{k-1} k p^{k-2}(k-kp-1) - \sum_{i=0}^{k-2} \beta_i \binom{k}{i} p^{i-1}(1-p)^{k-i-1}(pk-i), \end{aligned}$$

and evaluate at $1-\varepsilon$:

$$\begin{aligned}
A'_\alpha(1-\varepsilon) &= k(1-\varepsilon)^{k-2}(1-\varepsilon+\beta_{k-1}k\varepsilon-\beta_{k-1}) - \sum_{i=0}^{k-2}\beta_i\binom{k}{i}(1-\varepsilon)^{i-1}(\varepsilon)^{k-i-1}(k-i-k\varepsilon) \\
&> k(1-\varepsilon)^{k-2}(1-\varepsilon+\beta_{k-1}k\varepsilon-\beta_{k-1}) - \sum_{i=0}^{k-2}\beta_i\binom{k}{i}(1-\varepsilon)^{i-1}(\varepsilon)^{k-i-1}k \\
&> k(1-\varepsilon)^{k-2}(1-\varepsilon+\beta_{k-1}k\varepsilon-\beta_{k-1}) - \sum_{i=0}^{k-2}\beta_i\binom{k}{i}(\varepsilon)^{k-i-1}k \\
&= k(1-\varepsilon)^{k-2}(1-\varepsilon+\beta_{k-1}k\varepsilon-\beta_{k-1}) - k\varepsilon\sum_{i=0}^{k-2}\beta_i\binom{k}{i}(\varepsilon)^{k-i-2} \\
&> k(1-\varepsilon)^{k-2}(1-\varepsilon+\beta_{k-1}k\varepsilon-\beta_{k-1}) - k\varepsilon\sum_{i=0}^{k}\binom{k}{i} \\
&> k(1-\varepsilon)^{k-2}(1-\varepsilon+\frac{k-2}{k}(k\varepsilon-1)) - k\varepsilon 2^k \\
&= (1-\varepsilon)^{k-2}(2+k\varepsilon(k-3)) - k\varepsilon 2^k.
\end{aligned}$$

For $k>2$ and $\varepsilon<\varepsilon_k$,

$$\begin{aligned}
A'_\alpha(1-\varepsilon) &= (1-\varepsilon)^{k-2}(2+k\varepsilon(k-3)) - k\varepsilon 2^k \\
&> (1-\varepsilon)^{k-2}(2+k\varepsilon(k-3)) - \frac{1}{2} \\
&> (1-\frac{1}{48})2 - \frac{1}{2} \\
&= \frac{35}{24}.
\end{aligned}$$

■

Lemma 3.15 *If α is a monotone connective such that $A_\alpha(p)<p$ on the interval $(0,1)$ and, $A_\alpha(p)$ has degree $k>2$ and $\beta_{k-1}=\frac{k-1}{k}$, then, for all positive $\varepsilon<k^{-1}$,*

$$1+\varepsilon^k < A'_\alpha(1-\varepsilon) \le (1-\varepsilon)^{k-2}(k(k-2)\varepsilon+1).$$

Proof: For the lower bound, observe that since

$$A'_\alpha(p) = kp^{k-1} - \sum_{i=0}^{k-1}\beta_i\binom{k}{i}p^{i-1}(1-p)^{k-i-1}(pk-i),$$

for all $p>1-k^{-1}$, minimizing $A'_\alpha(p)$ maximizes the coefficients β_i, for all $i=2\ldots k-2$. Since the connective is of the form $\alpha(x)=x_j\wedge\alpha'(x)$, $\beta_i\le\binom{k-1}{i}\binom{k}{i}^{-1}=\frac{k-i}{k}$. Hence,

$$A'_\alpha(p) \ge kp^{k-1} - \sum_{i=0}^{k-1}\binom{k-1}{i}p^{i-1}(1-p)^{k-i-1}(pk-i) = 1+(1-p)^{k-2}(kp-1).$$

Thus, for all $\varepsilon < k^{-1}$,

$$A'_\alpha(1-\varepsilon) \geq 1+\varepsilon^{k-2}(k-1-k\varepsilon) > 1+\varepsilon^{k-2} > 1+\varepsilon^k.$$

For the upper bound we minimize all β_i for $i = 2 \ldots k-2$, i.e., $\beta_i = 0$. Thus, for all $p > 1-k^{-1}$,

$$A'_\alpha(p) \leq kp^{k-1} - (k-1)p^{k-2}(pk-k+1) = p^{k-2}(k(k-2)(1-p)+1),$$

implying that for all $\varepsilon < k^{-1}$,

$$A'_\alpha(1-\varepsilon) \leq (1-\varepsilon)^{k-2}(k(k-2)\varepsilon+1).$$

■

Theorem 3.16 *Let (μ,α) be a growth process where α is a k-adic monotone connective such that $A_\alpha(p) < p$ on the interval $(0,1)$, $k > 2$ and $\beta_{k-1} \leq \frac{k-2}{k}$. There exists a constant c_α, such that for all $n > 0$, if $i \geq 3\log(n) + c_\alpha$, then for all f, $|\pi_i(f) - \pi(f)| < 2^{-n}$.*

Proof: Let $\tilde{f}_i$ be a random variable with the distribution π_i. Using an argument that is similar to Valiant's [Val84], we claim that if $i \geq 3\log(n) + c_\alpha$, then for $|x| = n$, $P[\tilde{f}_i(x) = 0] = 0$, and for all x such that $|x| < n$, $P[\tilde{f}_i(x) = 1] < 2^{-2n}$. The former follows from the monotonicity of α; regardless of the number of iterations, a false negative will never occur.

In the latter case, assuming that all variables are independent, if $|x| < n$, $P[\tilde{f}_0(x) = 1] = |x|/n \leq 1-n^{-1}$. For $i > 0$, $P[\tilde{f}_i(x) = 1] = A^i_\alpha(p)$, where A^i_α denotes the ith composition of $A_\alpha(p)$ with itself. Expanding $A_\alpha(p)$ around 1,

$$A_\alpha(p) = A_\alpha(1) + A'_\alpha(1)(p-1) + O((p-1)^2),$$

yields:

$$A_\alpha(1-\varepsilon) = 1 - \varepsilon A'_\alpha(1) + O(\varepsilon^2).$$

From Lemma 3.14, let $\gamma = 35/24$ and let $\varepsilon_k = \frac{1}{k2^{k+1}}$. There exists an $\varepsilon_0 < \varepsilon_k$ such that, for all $\varepsilon < \varepsilon_0$,

$$A_\alpha(1-\varepsilon) < 1 - \varepsilon\gamma.$$

Since $P[\tilde{f}_0(x) = 1] \leq 1-n^{-1}$, for $i \geq 2\log(n) + 2\log(\varepsilon_0) > (\log(n) + \log(\varepsilon_0))/\log(35/24)$,

$$A^i_\alpha(1-\varepsilon) < 1 - \varepsilon\gamma^i < 1 - \varepsilon_0.$$

An additional constant number of iterations, say d_α, yields

$$A^{d_\alpha}_\alpha(1-\varepsilon_0) < c.$$

By Lemma 3.9, $A_\alpha(p) < k^2p^2$, thus we fix $c < \frac{1}{2k^2}$ and let $j = \log(n) + 1$. Hence,

$$A^j_\alpha(c) < (k^2c)^{2^j} < 2^{-2^j} = 2^{-2n}.$$

Therefore, for $i \geq 3\log(n) + 2\log(\varepsilon_0) + d_\alpha + 1$ and all x such that $|x| < n$, $P[\tilde{f}_i(x) = 1] < 2^{-2n}$, implying that $|\pi_i(f) - \pi(f)| < 2^{-n}$. ■

Unfortunately, if $\beta_{k-1} = \frac{k-1}{k}$, convergence takes time polynomial in n, e.g., if $\alpha(x) = \vee_{i=2}^{k}(x_1 \wedge x_i)$, convergence takes on the order of n^k time!

Theorem 3.17 *Let (μ, α) be a growth process where α is a k-adic monotone connective such that $A_\alpha(p) < p$ on the interval $(0,1)$, $k > 2$ and $\beta_{k-1} = \frac{k-1}{k}$. For any $\varepsilon_0 < 0$, if $A_\alpha^i(1-n^{-1}) < 1-\varepsilon_0$, then for sufficiently large n,*

$$i > p(n)(\log(n) + \log(\varepsilon_0)),$$

where $(k-1)(k-2)n \leq p(n) \leq \frac{n^{k-2}}{k-1}$.

Proof: Let $\tilde{f}_i$ be a random variable with the distribution π_i. If $|x| = n-1$ then $P[\tilde{f}_0(x) = 1] = 1-n^{-1}$. Furthermore, by Lemma 3.15, for sufficiently large n,

$$A'_\alpha(1-n^{-1}) < (1-n^{-1})^{k-2}(k(k-2)n^{-1}+1).$$

Since $\gamma < A'_\alpha(1-n^{-1})$, therefore

$$\log(\gamma) < (k-2)\log(1-n^{-1}) + \log(k(k-2)n^{-1}+1),$$

implying that

$$\frac{1}{\log(\gamma)} > \left((k-2)\log(1-n^{-1}) + \log(k(k-2)n^{-1}+1)\right)^{-1} > \frac{n}{k^2-3k+2} + O(1).$$

Thus, if $A_\alpha^i(1-n^{-1}) < 1-\varepsilon_0$, then for sufficiently large n, $A_\alpha^i(1-n^{-1}) < \varepsilon_0$ implies that

$$i > (k-1)(k-2)2n(\log(n) + \log(\varepsilon_0)).$$

In fact, this is the best case. If $\alpha(x) = \vee_{i=2}^{k}(x_1 \wedge x_i)$, then $A_\alpha(p) = p - p(1-p)^{k-1}$. By Lemma 3.15, $\gamma < 1 + n^{2-k}(k-1-kn^{-1})$, implying that $\log(\gamma) < \log(1+n^{2-k}(k-1-kn^{-1}))$, and

$$\frac{1}{\log(\gamma)} > \left(\log(1+n^{2-k}(k-1-kn^{-1}))\right)^{-1} > \frac{n^{k-2}}{k-1}.$$

Thus, if $A_\alpha^i(1-n^{-1}) < 1-\varepsilon_0$, then for sufficiently large n, $i > \frac{n^{k-2}}{k-1}(\log(n) + \log(\varepsilon_0))$. ■

Consequently, a growth process that uses a connective whose characteristic polynomial has no fixed-point can be classified as either rapidly converging or slowly converging, with the value of the second last coefficient, β_{k-1}, determining rate of convergence!

When $A_\alpha(p)$ has a Fixed-point on $(0,1)$

When the characteristic polynomial $A_\alpha(p)$ does have a fixed-point on the interval $(0,1)$, a similar analysis is used.

Lemma 3.18 *Let $A_\alpha(p)$ be the characteristic polynomial of any k-adic monotone connective α. If $A_\alpha(p)$ has a fixed-point $s \in (0,1)$, then $A'_\alpha(s) \geq 1 + \frac{k-2}{2^{k-2}}$.*

Proof: Consider a projection connective, say $\chi(x) = x_b$, $b \in [1,n]$. The corresponding characteristic polynomial is

$$A_\chi(p) = p = p^k + p(1-p)^{k-1} + \sum_{i=2}^{k-1} \binom{k-1}{i} p^i (1-p)^{k-i},$$

whose fixed-point is everywhere and whose slope is 1. Note that $\beta_1 = 1$ and $\beta_{k-1} = \frac{k-1}{k}$. Let

$$\eta(x) = \left(\bigwedge_{i=2}^{k} x_i \right) \vee \left(\bigvee_{i=2}^{k} (x_1 \wedge x_i) \right),$$

be a k-adic monotone connective. The corresponding characteristic polynomial

$$\begin{aligned} A_\eta(p) &= p - p(1-p)^{k-1} + p^{k-1}(1-p) \\ &= A_\chi(p) - p(1-p)^{k-1} + p^{k-1}(1-p) \\ &= A_\chi(p) + (A_\eta - A_\chi(p)) \end{aligned}$$

has a fixed-point at $\frac{1}{2}$. Not surprisingly, this is almost $A_\chi(p)$ except that $\beta_1 = 0$ and $\beta_{k-1} = 1$, i.e., the difference is just two terms. We claim that $A'_\eta(\frac{1}{2}) \leq A'_\alpha(s)$.

The claim is proved by contradiction. Assume that $A'_\eta(\frac{1}{2})$ is not the minimum slope at a fixed-point, then there exists a k-adic monotone connective ζ, whose degree k characteristic polynomial $A_\zeta(p)$ has a fixed-point $t \in (0,1)$, such that $A'_\zeta(t) < A'_\eta(\frac{1}{2})$ and $A'_\zeta(t)$ is the minimum slope. Since $\beta_1 = 0$ and $\beta_{k-1} = 1$ must hold for A_ζ, we write $A_\zeta(p)$ in a manner similar to $A_\eta(p)$: $A_\zeta(p) = A_\eta(p) + (A_\zeta(p) - A_\eta(p))$. Specifically we are interested in the differences between $A_\zeta(p)$ and $A_\eta(p)$. In fact,

$$\begin{aligned} A_\zeta(p) &= A_\eta(p) + \left[p^{i_0}(1-p)^{k-i_0} + p^{i_1}(1-p)^{k-i_1} \ldots \right] \\ &\quad - \left[p^{j_0}(1-p)^{k-j_0} + p^{j_1}(1-p)^{k-j_1} \ldots \right], \end{aligned}$$

where $i_l \leq i_{l+1} \leq k-2$ and $j_l \geq j_{l+1} \geq 2$.

Since $A_\zeta(p) \neq A_\eta(p)$, and $A_\zeta(p)$ has a fixed-point on $(0,1)$, either i_0 or j_0 must exist. Without loss of generality assume that i_0 exists and consider the characteristic polynomial $A_\delta(p) = A_\zeta(p) - p^{i_0}(1-p)^{k-i_0}$. Since the connective corresponding to $A_\zeta(p)$ is monotone, $kt < i_0$. which implies that the derivative $p^{i_0-1}(1-p)^{k-i_0-1}(i_0 - kp)$ of the term $p^{i_0}(1-p)^{k-i_0}$ is positive for all $t \leq \frac{i_0}{k}$. Since the fixed-point u of $A_\delta(p)$ is bounded by $\frac{i_0+1}{k}$, $A'_\delta(u) < A'_\zeta(t)$, implying that $A'_\zeta(t)$ is not the minimum slope, which is a contradiction! In fact, iteratively subtracting the terms $p^{i_l}(1-p)^{k-i_l}$ and adding the terms $p^{j_l}(1-p)^{k-j_l}$, reduces $A_\zeta(p)$ to $A_\eta(p)$!.

Noting that $A'_\eta(\frac{1}{2}) = 1 + \frac{k-2}{2^{k-2}}$ completes the proof. ■

Theorem 3.19 *Let (μ, α) be a growth process where α is a k-adic monotone connective such that $A_\alpha(s) = s \in (0,1)$. There exists a constant c_α, such that for all $n > 0$, if*

$$i \geq k2^k \log(n) + c_\alpha,$$

then for all functions f, $|\pi_i(f) - \pi(f)| < 2^{-n}$.

Proof: Let $\tilde{f}_i$ be a random variable with the distribution π_i. Using an argument that is similar to Valiant's [Val84], we claim that if $i \geq k2^k \log(n) + c_\alpha$, then for all x such that $|x| < sn$, $P[\tilde{f}_i(x) = 1] < 2^{-2n}$, and for all x such that $|x| > sn$, $P[\tilde{f}_i(x) = 0] < 2^{-2n}$. We first argue the former.

Assuming that all variables are independent, if $|x| < sn$, $P[\tilde{f}_0(x) = 1] \leq s - n^{-1}\varepsilon_\alpha(n)$, where $\varepsilon_\alpha(n) = \min_{j\in\mathbb{Z}} |s - \frac{j}{n}| = |s - \frac{j_0}{n}|$. Since s is a root of degree k polynomial, by Liouville's Approximation Theorem [Apo97]

$$\varepsilon_\alpha(n) = \left|s - \frac{j_0}{n}\right| > \frac{e_\alpha}{n^k},$$

where the constant e_α depends only on the connective.

For $i > 0$, $P[\tilde{f}_i(x) = 1] = A^i_\alpha(p)$. Expanding $A_\alpha(p)$ around s,

$$A_\alpha(p) = A_\alpha(s) + A'_\alpha(s)(p-s) + O((p-s)^2),$$

yields

$$A_\alpha(s-\varepsilon) = s - \varepsilon A'_\alpha(s) + O(\varepsilon^2).$$

By Lemma 3.18, fix $\gamma = 1 + 2^{-k+1}$; there exists an ε_0 such that for all $\varepsilon < \varepsilon_0$, $A_\alpha(s-\varepsilon) < s - \varepsilon\gamma$. Since $P[\tilde{f}_0(x) = 1] \leq s - n^{-1}\varepsilon_\alpha(n)$, if

$$i \geq \log(n\,\varepsilon_\alpha(n)^{-1}\varepsilon_0)/\log(\gamma) \geq \log(n^{k+1}e_\alpha^{-1}\varepsilon_0)/\log(\gamma),$$

then

$$A^i_\alpha(s-\varepsilon) < s - \varepsilon\gamma^i < s - \varepsilon_0.$$

An additional constant number of iterations, say d_α, yields

$$A^{d_\alpha}_\alpha(s-\varepsilon_0) < c.$$

By Lemma 3.9, $A_\alpha(p) < k^2p^2$, thus, we fix $c < \frac{1}{2k^2}$ and let $j = \log(n) + 1$. Hence,

$$A^j_\alpha(c) < (k^2c)^{2^j} < 2^{-2^j} = 2^{-2n}.$$

Therefore, if

$$i \geq k2^k \log(n) + \frac{\log(e_\alpha^{-1}\varepsilon_0)}{\log(\gamma)} + d_\alpha + 1,$$

for all x such that $|x| < sn$, $P[\tilde{f}_i(x) = 1] < 2^{-2n}$.

By the same argument, if $|x| > sn$, $P[\tilde{f}_i(x) = 0] < 2^{-2n}$. Since $|x| > sn$, for $P[\tilde{f}_0(x) = 0] \leq 1 - s - n^{-1}\varepsilon_\alpha(n)$, $P[\tilde{f}_1(x) = 0] = \bar{A}_\alpha(p) = 1 - A_\alpha(1-p)$, and $P[\tilde{f}_j(x) = 0] = \bar{A}^j_\alpha(p)$, $j > 0$. Just as in the preceding case, the composition of $\bar{A}_\alpha$ with itself first yields a first order divergence from $1 - s$, followed by a second order convergence towards zero. Therefore, $|\pi_i(f) - \pi(f)| < 2^{-n}$. ■

To reduce the constant in front of the log term, one solution is to use a nonuniform initial distribution to ensure that $\varepsilon_\alpha(n)$ is bounded from below by a constant, as is done by Valiant [Val84].

3.4.2 Growth Processes that Use Balanced Connectives

In this subsection, it will be convenient to assume that the support of μ contains both constants, as well as the projections, and to deal later with the cases in which one or both constants are missing from the support of μ. If the connective is balanced, then by Lemma 3.8, its characteristic polynomial has a fixed-point of $\frac{1}{2}$. If the number n of variables is odd, then the fraction of inputs that are true for any assignment is bounded away from $\frac{1}{2}$, that is, for any $j \in \{1,2,..,n+1\}$, $|\frac{1}{2} - \frac{j}{n+2}| \geq \frac{1}{2n+4}$. Hence, by the standard amplification argument, the limiting distribution will be concentrated on the n-adic majority function $T_{\lceil n/2 \rceil}$. In fact, by Theorem 3.19, the convergence to the majority function is rapid; if $i \geq k2^k \log(n) + O(1)$, then $|\pi_i(f) - \pi(f)| < 2^{-n}$. When the number of variables is even, however, something completely different happens.

The family of slice functions, denoted $\mathcal{S}_{m,n}$ and defined by Berkowitz [Ber82], are monotone n-adic functions that assume the value 1 for all assignments of weight greater than m, assume the value 0 for all assignments of weight less than m, and may take on either value for assignments of weight m. Unlike other growth processes where the distribution is either concentrated on a single function or is uniform on the support of the growth process, the following growth processes have a limiting distribution that is uniform on the set $\mathcal{S}_{n/2,n}$. This set includes a large number, $2^{\binom{n}{n/2}}$, of functions; but according to a result of Korshunov [Kor80], includes only a tiny fraction, less than $\exp-(\binom{n}{n/2+1}2^{-n/2})$, of the support of the growth process, which is the set of all monotone functions $\mathcal{M}_n$.

Define the n-adic functions

$$\chi_n(x) = \begin{cases} 1, & |x| = \frac{n}{2} \\ 0, & otherwise \end{cases} \quad \text{and} \quad \upsilon_n(x) = \begin{cases} 1, & |x| > \frac{n}{2} \\ 0, & otherwise \end{cases}.$$

Claim 3.20 *The Fourier coefficients of the probability distribution π that is uniform on the slice functions in $\mathcal{S}_{n/2,n}$ are given by*

$$\Delta(f) = \begin{cases} 0, & \langle f, \chi_n \rangle \neq 0 \\ (-1)^{\langle f, \upsilon_n \rangle}, & \langle f, \chi_n \rangle = 0 \end{cases},$$

where the inner product is over $\mathbb{Z}$.

Proof: Let $c = |\mathcal{S}_{\frac{n}{2},n}|^{-1} = 2^{-\binom{n}{n/2}}$. If $\langle f, \chi_n \rangle = 0$, then

$$\Delta(f) = \sum_{g \in \mathcal{F}_n} (-1)^{\langle f,g \rangle} \pi(g) = c \sum_{g \in \mathcal{S}_{\frac{n}{2},n}} (-1)^{\langle f,g \rangle} = c \sum_{g \in \mathcal{S}_{\frac{n}{2},n}} (-1)^{\langle f, \upsilon_n \rangle} = (-1)^{\langle f, \upsilon_n \rangle}.$$

Otherwise let w be a singleton such that $w \leq f \wedge \chi_n$ and let $W = \{g \in \mathcal{S}_{\frac{n}{2},n} : g \geq w\}$. Then

$$\Delta(f) = c \sum_{g \in \mathcal{S}_{\frac{n}{2},n}} (-1)^{\langle f,g \rangle} = c \sum_{g \in W} (-1)^{\langle f,g \rangle} + (-1)^{\langle f, g \oplus w \rangle} = 0.$$

■

We combine amplification with Fourier methods to obtain our result in this case. The following "Restriction Lemma" is the key to doing this.

Claim 3.21 *Let (μ, α) be a growth process, where α is a balanced monotone connective and where n is even, and let f be a Boolean n-adic function. If $f(x) = 1$ for some x with $|x| < n/2$, or if $f(x) = 0$ for some x with $|x| > n/2$, then $\lim_{i\to\infty} \pi_i(f) = 0$.*

Proof: This follows from Theorem 3.19. ■

Lemma 3.22 (The Restriction Lemma)
Let (μ, α) be a growth process where α is a balanced monotone connective. Then for all $w \in \mathcal{F}_n$,

$$\lim_{i\to\infty} \Delta_i(w) = (-1)^{\langle \upsilon_n, w\rangle} \lim_{i\to\infty} \Delta_i(w \wedge \chi_n).$$

Proof: We begin with the definition

$$\Delta_i(w) = \sum_{v\in B_n} (-1)^{\langle v,w\rangle} \pi_i(v),$$

then rewrite the equation as

$$\Delta_i(w) = \sum_{v\in B_n} (-1)^{\langle v,w\rangle} \pi_i(v) = \sum_{t\le\chi_n} \sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u,w\rangle} \pi_i(t \vee u),$$

and consider the restriction of w to the slice $\frac{n}{2}$, that is, $w \wedge \chi_n$. Since $\lim_{i\to\infty} \pi_i(t \vee u) = 0$ if $u \neq \upsilon_n$, $\lim_{i\to\infty} \Delta_i(w \wedge \chi_n)$ can be rewritten as

$$\begin{aligned}
\lim_{i\to\infty} \Delta_i(w \wedge \chi_n) &= \lim_{i\to\infty} \sum_{v\in B_n} (-1)^{\langle v, w\wedge\chi_n\rangle} \pi_i(v) \\
&= \lim_{i\to\infty} \sum_{t\le\chi_n} \sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u, w\wedge\chi_n\rangle} \pi_i(t \vee u) \\
&= \sum_{t\le\chi_n} \sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u, w\wedge\chi_n\rangle} \lim_{i\to\infty} \pi_i(t \vee u) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\wedge\chi_n\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t, w\wedge\chi_n\rangle} (-1)^{\langle \upsilon_n, w\wedge\chi_n\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t, w\wedge\chi_n\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t, w\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n).
\end{aligned}$$

This, in conjunction with

$$\begin{aligned}
\lim_{i\to\infty} \Delta_i(w) &= \lim_{i\to\infty} \sum_{t\le\chi_n} \sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u, w\rangle} \pi_i(t \vee u) = \sum_{t\le\chi_n} \sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u, w\rangle} \lim_{i\to\infty} \pi_i(t \vee u) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n) = \sum_{t\le\chi_n} (-1)^{\langle t, w\rangle} (-1)^{\langle \upsilon_n, w\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n) \\
&= (-1)^{\langle \upsilon_n, w\rangle} \sum_{t\le\chi_n} (-1)^{\langle t, w\rangle} \lim_{i\to\infty} \pi_i(t \vee \upsilon_n) = (-1)^{\langle \upsilon_n, w\rangle} \lim_{i\to\infty} \Delta_i(w \wedge \chi_n),
\end{aligned}$$

yields the identity. ■

Hence, all we need to show is that $\lim_{i\to\infty}\Delta_i(w) = 0$ for w such that $0 < w \leq \chi_n$. To do this we use Savický's [Sav90] argument, which uses induction on the weight of w and the recurrence

$$\Delta_{i+1}(w) = \sum_{j=1}^{k} a_j(w)\Delta_i(w)^j + y_i(w), \tag{3.3}$$

where

$$\begin{aligned} a_j(w) &= \sum_{\substack{t\in B_k \\ |t|=j}} S_\alpha(t)^{|w|}, \\ y_i(w) &= \sum_{\substack{\mathbf{v}\in\mathcal{T}_n^k \\ 0<\mathbf{v}_j<w}} \prod_{\substack{a\in B_n \\ w(a)=1}} S_\alpha(\mathbf{v}(a)) \prod_{j=1}^{k} \Delta_i(\mathbf{v}_j), \\ S_\alpha(t) &= \frac{1}{2^k}\sum_{r\in B_k} (-1)^{\langle r,t\rangle}(-1)^{\alpha(r)}. \end{aligned}$$

Note that unlike the analysis of growth processes with linear connectives in Proposition 3.1, this recurrence is much more complicated. We first state two small lemmas of Savický's [Sav90] and then the main proposition.

Lemma 3.23 (Savický, 1990, Lemma 5.1 in [Sav90])
Let $x_{i+1} = \sum_{j=1}^{k} a_j x_i^j$, *such that* $|x_0| < 1$, $\sum_{j=1}^{k} |a_j| \leq 1$, *and* $|a_1| < 1$. *Then* $|x_{i+1}| \leq a|x_i|$, *where* $a \leq |a_1| + |x_0|(1-|a_1|)$. *Hence,* $|x_i| \leq a^i|x_0|$ *and* $\lim_{i\to\infty} x_i = 0$.

Lemma 3.24 (Savický, 1990, Lemma 5.2 in [Sav90])
Let $x_{i+1} = y_i + \sum_{j=1}^{k} a_j x_i^j$, *such that* $|x_i| \leq 1$, *and* $a = \sum_{j=1}^{k} |a_j| < 1$. *Then* $|x_{i+1}| \leq a|x_i| + y_i$ *and* $\lim_{i\to\infty} x_i = 0$.

Proposition 3.25 *Let* (μ,α) *be a growth process where* α *is a monotone balanced nonprojection connective,* n *is even, and the support of* μ *comprises the projection functions and constants. If* $0 < w \leq \chi_n$, *then* $\lim_{i\to\infty}\Delta_i(w) = 0$.

Proof: Let $w \leq \chi_n$ and recall equation 3.1:

$$\Delta_0(w) = \sum_{f\in F_n} \pi_0(f)(-1)^{\langle f,w\rangle} = \frac{1}{n+2}\sum_{f\in supp(\mu)} (-1)^{\langle f,w\rangle}.$$

To prove this proposition we substitute our base cases into Theorem 5.3 in [Sav90]. We need only show that $\Delta_0(w) = 0$ if $|w| \doteq 1$, and that $|\Delta_0(w)| < 1$ if $|w| = 2$. In the first case, since $w \leq \chi_n$, w is true on a single assignment of weight $n/2$. Hence, $\langle w,x_i\rangle = 1$ for exactly half the projections, where x_i is the ith projection function. Hence, the projections cancel each other out. Similarly, the two constants annihilate one another. Hence, $\Delta_0(w) = 0$ if $|w| = 1$.

The latter case, $|w| = 2$, is only slightly harder. Since the constant 0 is part of the support, there will be at least one positive contribution, $-1^{\langle \mathbf{0},w\rangle}\pi_0(\mathbf{0}) = \frac{1}{n+2}$. Hence, if $|\Delta_0(w)| = 1$, all other contributions must also be positive, specifically, $\langle w,x_i\rangle = 0$ for all x_i; by the pigeonhole principle

this is not possible. Hence, $|\Delta_0(w)| < 1$ if $|w| = 2$. Since $|w| = 2$, $y_i(w) = 0$ in equation 3.3 allowing us to invoke Lemma 3.23.

For the case of $|w| > 2$, by induction $\lim_{i\to\infty} y_i(w) = 0$ and, $|\Delta_i(w)| \leq 1$, hence we can invoke Lemma 3.24 to complete the proof. ■

Proposition 3.25, together with Claim 3.20, yields one of our main results.

Theorem 3.26 *Let (μ, α) be a growth process, where α is a monotone balanced nonprojection connective, n is even and the support of μ comprises the projection functions and constants. Then the limiting distribution is uniform on the functions in $\mathcal{S}_{n/2,n}$.*

Convergence Bounds

To bound the convergence within the slice we use a theorem of Savický [Sav95a]; the conditions of the theorem are verified in Proposition 3.25.

Theorem 3.27 (Savický, 4.8 in [Sav95a]) *If α is balanced and nonlinear, $\Delta_0(w) = 0$ for every w such that $|w| = 1$, $\Delta_0(w) < 1$ for every w such that $|w| = 2$, and there exists a w such that $|w| = 2$ and $\Delta_0(w) > 0$, then*

$$\max_{f\in\mathcal{F}_n} |\pi_i(f) - \pi(f)| = e^{-\Omega(i)}.$$

A more explicit bound, in terms of the number of arguments and the arity of the connective, is possible. We use a more explicit version of Lemma 3.22 and bound the convergence of the growth processes characterized by Theorem 5.3 in [Sav90]. A corollary of Lemma 3.28 is that the same bound also applies to the growth processes on monotone formulas whose limiting distribution is uniform over the slice functions. We first prove Lemma 3.28, then prove a sequence of lemmas that culminate in the bound: Theorem 3.34.

Lemma 3.28 *Let (μ, α) be a growth process, where α is a balanced monotone connective, then for some fixed $\varepsilon < 0$ and for all $w \in \mathcal{F}_n$,*

$$\Delta_i(w) = O(\varepsilon^{2^i}) + (-1)^{\langle \upsilon_n, w\rangle}\Delta_i(w \wedge \chi_n).$$

Proof: Following the proof Lemma 3.22 begin with the definition

$$\Delta_i(w) = \sum_{v\in B_n} (-1)^{\langle v,w\rangle}\pi_i(v),$$

and rewrite the equation as

$$\Delta_i(w) = \sum_{v\in B_n} (-1)^{\langle v,w\rangle}\pi_i(v) = \sum_{t\leq\chi_n}\sum_{u\leq\overline{\chi_n}} (-1)^{\langle t\vee u,w\rangle}\pi_i(t\vee u).$$

Consider the restriction of w to the slice $\frac{n}{2}$, that is, $w \wedge \chi_n$. By Theorem 3.19, after $O(\log(n))$ iterations $\pi_i(t \vee u) = O(\varepsilon^{2^i})$, if $u \neq \upsilon_n$. Hence, for $i > O(\log(n))$, $\Delta_i(w \wedge \chi_n)$ can be rewritten as

$$\begin{aligned}
\Delta_i(w \wedge \chi_n) &= \sum_{v \in B_n} (-1)^{\langle v, w\wedge\chi_n\rangle}\pi_i(v) \\
&= \sum_{t\le\chi_n}\sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u, w\wedge\chi_n\rangle}\pi_i(t\vee u) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\wedge\chi_n\rangle}\pi_i(t\vee\upsilon_n) + \sum_{t\le\chi_n}\sum_{u\le\overline{\chi_n}, u\neq\upsilon_n} (-1)^{\langle t\vee u, w\wedge\chi_n\rangle}\pi_i(t\vee u) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\wedge\chi_n\rangle}\pi_i(t\vee\upsilon_n) + \sum_{t\le\chi_n}\sum_{u\le\overline{\chi_n}, u\neq\upsilon_n} O(\varepsilon^{2^i}) \\
&= O(\varepsilon^{2^i}) + \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\wedge\chi_n\rangle}\pi_i(t\vee\upsilon_n) \\
&= O(\varepsilon^{2^i}) + \sum_{t\le\chi_n} (-1)^{\langle t, w\wedge\chi_n\rangle}(-1)^{\langle \upsilon_n, w\wedge\chi_n\rangle}\pi_i(t\vee\upsilon_n) \\
&= O(\varepsilon^{2^i}) + \sum_{t\le\chi_n} (-1)^{\langle t, w\wedge\chi_n\rangle}\pi_i(t\vee\upsilon_n) \\
&= O(\varepsilon^{2^i}) + \sum_{t\le\chi_n} (-1)^{\langle t, w\rangle}\pi_i(t\vee\upsilon_n).
\end{aligned}$$

This, in conjunction with

$$\begin{aligned}
\Delta_i(w) &= \sum_{t\le\chi_n}\sum_{u\le\overline{\chi_n}} (-1)^{\langle t\vee u, w\rangle}\pi_i(t\vee u) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\rangle}\pi_i(t\vee\upsilon_n) + \sum_{t\le\chi_n}\sum_{u\le\overline{\chi_n}, u\neq\upsilon_n} (-1)^{\langle t\vee u, w\rangle}\pi_i(t\vee u) \\
&= \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\rangle}\pi_i(t\vee\upsilon_n) + \sum_{t\le\chi_n}\sum_{u\le\overline{\chi_n}, u\neq\upsilon_n} O(\varepsilon^{2^i}) \\
&= O(\varepsilon^{2^i}) + \sum_{t\le\chi_n} (-1)^{\langle t\vee\upsilon_n, w\rangle}\pi_i(t\vee\upsilon_n) \\
&= O(\varepsilon^{2^i}) + \sum_{t\le\chi_n} (-1)^{\langle t, w\rangle}(-1)^{\langle \upsilon_n, w\rangle}\pi_i(t\vee\upsilon_n) \\
&= O(\varepsilon^{2^i}) + (-1)^{\langle \upsilon_n, w\rangle}\sum_{t\le\chi_n} (-1)^{\langle t, w\rangle}\pi_i(t\vee\upsilon_n) \\
&= O(\varepsilon^{2^i}) + (-1)^{\langle \upsilon_n, w\rangle}\Delta_i(w\wedge\chi_n),
\end{aligned}$$

yields the identity. ■

Lemma 3.29 (Savický, 1990, Lemma 4.7 in [Sav90])
For any k-adic connective α, 1) $\sum_{t\in B_k} S_\alpha(1)^2 = 1$, and 2) α is balanced and nonlinear if and only if $S_\alpha(\mathbf{0}) = 0$ and for all $s \in B_k$, $|S_\alpha(s)| < 1$.

Theorem 3.30 (Savický, 5.3 in [Sav90]) *Let (μ, α) be a growth process, where α is a k-adic nonlinear balanced connective, and assume that the initial distribution μ is uniform over the projections, their negations, and constants. For all $w \in \mathcal{F}_n$, such that $w > \mathbf{0}$, $\lim_{i\to\infty}\Delta_i(w) = 0$.*

Lemma 3.31 *Let (μ,α) be a growth process, where α is a k-adic nonlinear balanced connective, and assume that the initial distribution μ is uniform over the projections, their negations, and constants. For any $w \in \mathcal{F}_n$ such that $|w| = 2$, and any positive $c < 1$, if*

$$i > \log(c^{-1})n2^k,$$

then $|\Delta_i(w)| \leq c$.

Proof: By Theorem 5.3 in [Sav90],

$$\Delta_i(w) = \sum_{j=1}^{k} a_j(w)\Delta_{i-1}(w)^j,$$

where

$$a_j(w) = \sum_{\substack{t\in B_k \\ |t|=j}} S_\alpha(t)^{|w|} = \sum_{\substack{t\in B_k \\ |t|=j}} S_\alpha(t)^2.$$

By corollary 3.23, $|\Delta_i(w)| \leq a^i|\Delta_0(w)|$, where $a \leq a_1(w) + \Delta_0(w)(1-a_1(w))$. By Theorem 5.3 in [Sav90], $\Delta_0(w) < \frac{n-1}{n+1}$, hence,

$$a \leq a_1(w) + \frac{n-1}{n+1}(1-a_1(w)) = 1 - \frac{2}{n+1}(1-a_1(w)),$$

and since $a_1(w) \leq 1-2^{-k}$,

$$a \leq 1 - \frac{2}{n+1}2^{-k}.$$

Setting $x_0 = \Delta_0(w)$ and solving for i in the inequality $a^i|\Delta_0(w)| \leq c$ yields

$$i \geq \frac{\log(c) - \log|x_0|}{\log(a)},$$

and substituting for a on the right,

$$\begin{aligned}
\frac{\log(c) - \log|x_0|}{\log(a)} &= \frac{\log(c) - \log|x_0|}{\log\left(1 - \frac{2}{n+1}2^{-k}\right)} \\
&< ((n+1)2^{k-1} + \frac{1}{2})(\log(c^{-1}) + \log|x_0|) \\
&< ((n+1)2^{k-1} + \frac{1}{2})\log(c^{-1}) \\
&< n2^k\log(c^{-1}).
\end{aligned}$$

Thus, for $i > \log(c^{-1})n2^k$, $|\Delta_i(w)| \leq c$. ■

Lemma 3.32 *Let $x_{i+1} = y_i + \sum_{j=1}^{k} a_j x_i^j$, such that $|x_i| \leq 1$, and $a = \sum_{j=1}^{k} |a_j| < 1$. If $y_i < (i-l+2)^k a^{i-l+1} < a < 1$ for some $k \geq 0$ and $l > 0$. Then $|x_i| \leq (i-l+1)^{k+1}a^{i-l}$ for all $i \geq l$.*

Proof: By Lemma 3.24, $|x_{i+1}| \leq a|x_i| + y_i$. Hence, by induction on i

$$\begin{aligned} |x_{i+l}| &\leq a^i(|x_l| + (i-l+2)^k a^l) \\ &\leq a^{i-l}\left(1 + \sum_{j=l}^{i}(i-l+2)^k\right) \\ &\leq a^{i-l}(i-l+2)^{k+1}. \end{aligned}$$

■

Lemma 3.33 *Let (μ, α) be a growth process, assume that the conditions of Theorem 3.30 are satisfied, and let*

$$a = \sum_{t \in B_k} |S_\alpha(t)|^3 < 1 - 2^{-k}.$$

If $|w| = d \geq 2$ and

$$i_d = n2^k \log(a^{-1}) + \sum_{j=3}^{d} \frac{(k+1)^j j}{\log(a^{-1})}, \tag{3.4}$$

then $|\Delta_i(w)| \leq a^{i-i_d} b_d(i)$, where $b_d(i) = (i - i_2 + 2)^{(k+1)^{d-3}}$, and $b_2(i) = 1$.

Proof: The proof is by induction on j. By Theorem 5.3 in [Sav90],

$$\Delta_i(w) = y_i(w) + \sum_{j=1}^{k} a_j(w)\Delta_{i-1}(w)^j,$$

where

$$a_j(w) = \sum_{\{t \in B_k:\ |t|=j\}} S_\alpha(t)^{|w|}, = \sum_{\{t \in B_k:\ |t|=j\}} S_\alpha(t)^d,$$

and

$$y_i(w) = \sum_{\mathbf{v} \in G_w^k} \left(\prod_{a \in B_n | w(a)=1} S_\alpha(\mathbf{v}(a))\right)\left(\prod_{j=1}^{k} \Delta_i(\mathbf{v}_j)\right),$$

where $G_w = \{v \in B_{2^n} : \mathbf{0} < v < w\}$.

Since $\sum_{t \in B_k} S_\alpha(1)^2 = 1$ and $|S_\alpha(s)| < 1$ for all $s \in B_k$,

$$\begin{aligned} a(w) &= \sum_{j=1}^{k} a_j(w) = \sum_{j=1}^{k} \sum_{t \in B_k : |t|=j} S_\alpha(t)^{|w|} \\ &= \sum_{t \in B_k} S_\alpha(t)^{|w|} \leq \sum_{t \in B_k} |S_\alpha(t)|^{|w|} \\ &\leq \sum_{t \in B_k} |S_\alpha(t)|^3 < \sum_{t \in B_k} S_\alpha(t)^2 = 1. \end{aligned}$$

The base case, $d = 2$, is proved by Lemma 3.31. For the base case, $d = 3$, we first bound $y_i(w)$. By Theorem 5.3 in [Sav90], $\Delta_i(w) = 0$ if $|w| = 1$ and by Lemma 3.31, $\Delta_i(w) \leq a^{i-n2^k \log(a^{-1})}$ if $|w| = 2$.

Hence, for all $w \in G_w$, $\Delta_i(w) \leq a^{i-n2^k \log(a^{-1})}$. If we let $i_2 = n2^k \log(a^{-1})$, then $\Delta_i(w) \leq a^{i-i_2} b_2(i)$ and

$$\begin{aligned} y_i(w) &= \sum_{\mathbf{v} \in G_w^k} \left(\prod_{a \in B_n | w(a)=1} S_\alpha(\mathbf{v}(a)) \right) \left(\prod_{j=1}^{k} \Delta_i(\mathbf{v}_j) \right) \\ &< \sum_{\mathbf{v} \in G_w^k} \prod_{j=1}^{k} \Delta_i(\mathbf{v}_j) \\ &< \sum_{\mathbf{v} \in G_w^k} \left(a^{i-i_{d-1}} b_{d-1}(i) \right)^k \\ &= \left(\sum_{j=1}^{d-1} \binom{d}{j} \right)^k \left(a^{i-i_{d-1}} b_{d-1}(i) \right)^k \\ &< \left(2^d a^{i-i_{d-1}} b_{d-1}(i) \right)^k. \end{aligned}$$

Solving for i in the inequality

$$\left(2^d a^{i-i_{d-1}} b_{d-1}(i) \right)^k < a$$

yields

$$\frac{d + \log(b_{d-1}(i))}{\log(a^{-1})} + k^{-1} + i_{d-1} \leq \frac{(k+1)^d d}{\log(a^{-1})} + i_{d-1} = i_d \leq i,$$

which reduces to equation 3.4 when evaluated at $d = 3$. Hence, for $i \geq i_d$

$$y_i(w) < \left(2^d a^{i-i_{d-1}} b_{d-1}(i) \right)^k < b_{d-1}(i)^k a^{i-i_d} < a,$$

and by Lemma 3.24 and Corollary 3.32 we complete the base case:

$$\begin{aligned} |\Delta_{i+1}(w)| &\leq a|\Delta_i(w)| + y_i(w) \\ &\leq a|\Delta_i(w)| + a^{i-i_d} b_{d-1}(i)^k \\ &\leq a^{i-i_d} \left(1 + \sum_{j=i_d}^{i} 1 \right) \\ &\leq a^{i-i_d} (i - i_d + 2) \\ &\leq a^{i-i_d} (i - i_2 + 2) \\ &= a^{i-i_d} b_d(i). \end{aligned}$$

Assume that the hypothesis holds for all w of weight less than some fixed d and let $|w| = d$. Repeating the above calculations in terms of d we get an identical bound for $y_i(w)$,

$$y_i(w) < \left(2^d a^{i-i_{d-1}} b_{d-1}(i) \right)^k < b_{d-1}(i)^k a^{i-i_d} < a,$$

where, by the inductive hypothesis,

$$\begin{aligned} i_d &= \frac{(k+1)^d d}{\log(a^{-1})} + i_{d-1} \\ &= \frac{(k+1)^d d}{\log(a^{-1})} + n2^k \log(a^{-1}) + \sum_{j=3}^{d-1} \frac{(k+1)^j j}{\log(a^{-1})} \\ &= n2^k \log(a^{-1}) + \sum_{j=3}^{d} \frac{(k+1)^j j}{\log(a^{-1})} \end{aligned}$$

and hence

$$\begin{aligned} |\Delta_{i+1}(w)| &\leq a|\Delta_i(w)| + y_i(w) \\ &\leq a|\Delta_i(w)| + a^{i-i_d} b_{d-1}(i)^k \\ &\leq a^{i-i_d} \left(1 + \sum_{j=i_d}^{i} b_{d-1}^k\right) \\ &\leq a^{i-i_d} \left(1 + \sum_{j=i_d}^{i} (i-i_2+2)^{(k+1)^{d-1-3}k}\right) \\ &\leq a^{i-i_d}(i-i_d+1)(i-i_2+2)^{(k+1)^{d-1-3}k} \\ &\leq a^{i-i_d}(i-i_2+2)(i-i_2+2)^{(k+1)^{d-1-3}k} \\ &= a^{i-i_d}(i-i_2+2)^{(k+1)^{d-3}} \\ &= a^{i-i_d} b_d(i), \end{aligned}$$

completing inductive step. ■

Theorem 3.34 *Let (μ, α) be a growth process, assume that the conditions of Theorem 3.30 are satisfied, let a be as in Lemma 3.33, and let*

$$I = n2^k \log(a^{-1}) + \frac{2^{2n}(k+1)^{2^n}}{\log(a^{-1})}.$$

For any positive $c < 1$, if $w \neq \mathbf{0}$ and

$$i \geq \frac{\log(c)}{\log(a)} + \frac{\log(i-I+2)}{\log(a^{-1})}(k+1)^{2^n} + I, \tag{3.5}$$

then $|\Delta_i(w)| \leq c$.

Proof: By Lemma 3.33 the coefficient of weight 2^n has the greatest converging bound:

$$|\Delta_{i+1}(w)| \leq a^{i-i_{2^n}}(i - n2^k \log(a^{-1}) + 2)^{(k+1)^{2^n}},$$

where

$$\begin{aligned} i_{2^n} &= n2^k \log(a^{-1}) + \sum_{j=3}^{2^n} \frac{(k+1)^j j}{\log(a^{-1})} \\ &\leq n2^k \log(a^{-1}) + \frac{2^{2n}(k+1)^{2^n}}{\log(a^{-1})} = I. \end{aligned}$$

Solving for i in the inequality

$$a^{i-i_{2^n}}(i-n2^k\log(a^{-1})+2)^{(k+1)^{2^n}} < c$$

completes the proof. ■

Thus, by equation 2.2,

$$\begin{aligned}
\pi_i(g) &= \frac{1}{2^{2^n}}\sum_{f\in\mathcal{F}_n}(-1)^{\langle f,g\rangle}\Delta_i(f) \\
&= \frac{1}{2^{2^n}}+\frac{1}{2^{2^n}}\sum_{f\in\mathcal{F}_n\setminus\mathbf{0}}(-1)^{\langle f,g\rangle}\Delta_i(f) \\
&\leq \frac{1}{2^{2^n}}+\frac{1}{2^{2^n}}\sum_{f\in\mathcal{F}_n\setminus\mathbf{0}}|\Delta_i(f)| \\
&\leq \frac{1}{2^{2^n}}+\max_{f\in\mathcal{F}_n\setminus\mathbf{0}}|\Delta_i(f)|,
\end{aligned}$$

implying that for all $g\in\mathcal{F}_n$, $|\pi(g)-\pi_i(g)|\leq c$ if i satisfies equation 3.5. Although, the convergence of the growth process is eventually exponentially fast, by Theorem 3.34, it may take an exponential number of iterations (in n) before this occurs.

Varying the Initial Distribution

Proposition 3.25 can easily be modified to cover the cases in which one of the constants is missing from the support of μ: in these cases there is concentration on a single function when n is even and uniform distribution on a set of slice functions when n is odd. When the support of μ consists only of the projection functions, however, the situation can be more complicated. If α is not self-dual or n is odd, the result is the same as when both constants are present. If α is self-dual and n is even, however, the limiting distribution is uniform on a subset of the slice functions.

Theorem 3.35 *Let (μ,α) be a growth process where α is a monotone self-dual nonprojection connective, n is even, and the support of μ comprise the projection functions. Then the limiting distribution is uniform on the self-dual functions in $\mathcal{S}_{n/2,n}$.*

Proof: This is similar to that of theorem 3.5. ■

We note that there are $2^{\frac{1}{2}\binom{n}{n/2}}$ self-dual functions in $\mathcal{S}_{n/2,n}$. According to Sapozhenko's [Sap89] result, this is only a tiny fraction, less than $\exp-(\binom{n}{n/2+1}2^{-n/2-1})$, of the support of the growth process, which is the set of self-dual monotone functions.

3.5 Growth Processes that Use Other Functions

We can use the same method to analyze other growth processes. For example, the uniform distribution on the set of bipreserving functions (that is, those functions satisfying $f(0,\ldots,0)=0$ and

$f(1,\ldots,1) = 1$) can be generated by a growth process that uses the bipreserving selection connective $\alpha(x,y,z) = xy \vee \bar{x}z$ and an initial distribution that is uniform on the projection functions. The same technique as in the monotone case is sufficient to prove this; the corresponding restriction lemma yields the identity

$$\lim_{i\to\infty} \Delta_i(w) = (-1)^{\langle w,\eta_n\rangle}\Delta(w \wedge \kappa_n),$$

where the two functions are

$$\eta_n = \bigwedge_{j=1}^{n} x_j \quad \text{and} \quad \kappa_n = \left(\bigvee_{j=1}^{n} x_j\right) \wedge \overline{\bigwedge_{j=1}^{n} x_j}.$$

A similar analysis for the 0-preserving and 1-preserving functions follows easily.

Chapter 4

Growth Processes on Reversible Circuits

In this chapter we investigate growth processes on reversible circuits. The investigation is divided into three parts. In Section 4.2 we show that the result of a growth process on reversible circuits is either a limiting distribution or two alternating distributions. We show that in almost all cases the limiting distribution does exist and is uniform over the set of functions that can be realized by the growth process. In Section 4.3 we investigate what functions are realizable by a growth process on reversible circuits, and in Section 4.4 we bound the convergence rate of growth processes on reversible circuits. We first cover the basic definitions before proceeding to our results.

4.1 Definitions

Let S_{2^n} denote the symmetric group on 2^n points and let A_{2^n} denote the alternating group on 2^n points; the groups will be regarded as acting on the points of the Boolean cube, B_n.

Let C denote an m-gate reversible circuit comprising n lines (wires) and let $|C| = m$ denote the size of C (the number of gates). A reversible circuit takes an assignment, x, of size n and yields an output, denoted $C(x)$, which is also of size n. Unless noted otherwise, reversible circuits are assumed to comprise the standard reversible gates that include: the identity gate, the unary NOT gate, the binary controlled-NOT gate (C-NOT), and the ternary Toffoli gate (controlled-controlled-NOT gate).

An m-gate reversible circuit C is specified by a word of length m, $C = g_1 g_2 \dots g_m$, over an alphabet Σ representing the set of possible gates; each gate, g_i, is specified by its operation and the lines on which it operates. A NOT gate, operating on line i, is denoted $\oplus_i$; a controlled-NOT gate, operating on line j and controlled by line i, is denoted $\oplus_j^i$; and a Toffoli gate, operating on line k and controlled by lines i and j, is denoted by $\oplus_k^{i \wedge j}$. The identity gate, denoted ε, serves an important purpose in our analysis, but performs no operation on the circuit. Since each of the gates is its own inverse, the inverse of a gate, denoted g_i^{-1}, is g_i.

The composition of two n-line reversible circuits $C = g_1 \dots g_m$ and $C' = g'_1 \dots g'_{m'}$ is the concatenation of the two words $C'' = CC' = g_1 \dots g_m g'_1 \dots g'_{m'}$. Composition is also denoted as $C'(C)$, and the output of the composition is denoted $C''(x) = C'(C(x))$. Since each standard gate is its own

inverse, the inverse of a circuit, denoted C^{-1}, is simply

$$C^{-1} = (C)^{-1} = (g_1 \dots g_m)^{-1} = g_m^{-1} \dots g_1^{-1} = g_m \dots g_1.$$

Analogously, the inverse of a composition of two circuits is $(CC')^{-1} = C'^{-1}C^{-1}$.

Let S be a set of n-line reversible circuits. Since every element in S realizes a permutation on the Boolean cube B_n, each reversible circuit corresponds to an element of S_{2^n} and the closure of S (under composition), corresponds to a subgroup of S_{2^n}. Thus, every element in the subgroup is realized by some composition of circuits from S. If a reversible circuit C realizes a permutation $\sigma \in S_{2^n}$, then we say that $C \sim \sigma$; if two circuits, C and C', realize the same permutation, then we say that $C \sim C'$. An immediate consequence of the correspondence between reversible circuits and elements of S_{2^n} is that the group axioms also apply to the composition and manipulation of reversible circuits. We use the group-theoretic setting to prove many of our results dealing with reversible circuits.

A growth process on reversible circuits is denoted by a pair (μ, X) where μ is a uniform distribution on a subset of the gate set X. Let $X = X_n$, where unless otherwise noted, the set X_b, $1 \leq b \leq n$, comprises

- 1 identity gate: ε, (no operation is performed)
- b NOT gates: $\bigoplus_i$, $i = n-b+1, \dots, n$,
- $b(n-1)$ controlled-NOT gates: $\bigoplus_j^i$, $j = n-b+1, \dots, n$, $i = 1, \dots, j-1, j+1, \dots, n$, and
- $b\binom{n-1}{2}$ Toffoli gates: $\bigoplus_k^{i \wedge j}$, $k = n-b+1, \dots, n$, $j = 1, \dots, k-1, k+1, \dots, n$, $i = j+1, \dots, k-1, k+1, \dots, n$.

Since a reversible circuit realizes an element of S_{2^n}, the ith iteration of the growth process induces a distribution π_i on S_{2^n} in the following way. The distribution π_0 is concentrated on the identity element ε, i.e., $\pi_0(\varepsilon) = 1$, and if $\sigma \in S_{2^n}$, then $\pi_i(\sigma) = \Pr[C_{\tilde{\sigma}_{i-1}} C_{\tilde{g}} \sim \sigma]$, where $\tilde{\sigma}_{i-1}$ is a random variable distributed according to distribution π_{i-1}, and $\tilde{g}$ is a random variable distributed according to distribution μ; $C_{\tilde{\sigma}_i}$ and $C_{\tilde{g}}$ denote the corresponding circuits that realize the elements of the respective sample spaces.

Recalling the definitions from Chapter 3, a growth process has a **limiting distribution**, π, if π_i approaches π as i (the number of iterations) tends infinity; and a growth process has **alternating distributions** if π_{2i} and π_{2i+1} approach two distinct distributions as i tends to infinity. The **support** of a growth process is the set of all functions that are realizable by the process, i.e., $\cup_{i>0} supp(\pi_i)$.

We say that a gate set S is **alternating**, if there does not exist a circuit C over S such that $|C|$ is odd and $C \sim \varepsilon$. A **gate-symmetric** gate set $S \subseteq X_b$ satisfies the condition that if a Toffoli (controlled-NOT or NOT) gate is in S, then all Toffoli (controlled-NOT or NOT) gates that are in X_b are also in S. Intuitively this means that no line within the circuit is preferred to any other line.

If $supp(\mu) \subseteq X_b$, $b < n$, the support of a growth process (μ, X) will only contain circuits whose bandwidth is less than n, i.e., some fixed set of lines is guaranteed to be read-only. We call such a growth process **bandwidth-limited** or a **bandwidth-b** growth process, if b is known. If for each of the n lines the support of μ includes gates that modify each of the n lines, then the growth process is not bandwidth-limited. For conciseness, an n-bit input to a bandwidth-limited circuit is denoted $xx' = (x_1, \dots, x_r, x'_1, \dots, x'_b)$, where x comprises the read-only inputs and x' comprises the read-write inputs.

4.2 The Limiting Distribution of Growth Processes on Reversible Circuits

Amazingly, the question of whether a growth process on reversible circuits has a limiting distribution is decoupled from the question of what that limiting distribution is. Assuming that the distribution μ on the gate set X is uniform, there is straightforward proof that the limiting distribution, if one exists, will be uniform over some set of permutation functions, which corresponds to a subgroup of S_{2^n}. In this section we prove the following theorem:

Theorem 4.1 *Let (μ, X) be a growth process on reversible circuits, where μ is a distribution on X and let $\Sigma = supp(\mu)$. The growth process has a limiting distribution if and only if Σ is not alternating. Otherwise, the result of the growth process is two alternating distributions.*

To prove this we first argue that the growth process is an ergodic Markov process with periodicity at most 2 and then argue that the Markov process is aperiodic if and only if the support of distribution μ is not alternating.

Lemma 4.2 *If (μ, X) is a growth process, where X contains the standard gates and μ is a distribution on X, then the growth process is an ergodic irreducible Markov process with periodicity at most* 2.

Proof: The state space of this Markov process is a subgroup of S_{2^n} that is generated by the set of elements that are realized by the gates in the support of μ. Assuming that the transition matrix is indexed by the elements of the subgroup, entry $M_{\sigma,\tau}$ of the transition matrix is equal to $\mu(g)$ where $C_g \sim \sigma^{-1}\tau$, i.e., the probability of drawing gate g according to distribution μ, such that composing σ with the element realized by g yields τ. Since all gates are their own inverses, $M_{\sigma,\tau} = M_{\tau,\sigma}$, implying that M is symmetric. Therefore, each state can be returned to after two transitions, which means that, by definition, the period of the process is at most 2. Furthermore, since the state space is equal to the subgroup, every state is reachable from the identity, and the identity is reachable from every state. Hence, the Markov process is ergodic, irreducible, and has a period of at most 2. ■

Lemma 4.3 *Let (μ, X) be a growth process, where X contains the standard gates and let $\Sigma = supp(\mu)$. The Markov process is aperiodic if and only if Σ is not alternating.*

Proof: By Lemma 4.2 the periodicity of the process is at most 2; hence, the process is either aperiodic or has periodicity 2. If Σ is not alternating, there exists a circuit $C \in \Sigma^+$, such that $m = |C|$ is odd and $C \sim \varepsilon$. Hence, there is an odd length cycle in the graph induced by process. Since $\gcd(2, m) = 1$, the process is aperiodic by definition.

Conversely, if the process is aperiodic, there must be at least one odd cycle in the respective graph. Concretely, starting in some state, say σ, after an odd number of transitions, the process can end up in the same in the same state. Let $C \sim \sigma$ and let the transitions be denoted by g_i, for $i = 1, \ldots, m$, where m is odd. Thus, $Cg_1g_2 \ldots g_m \sim C$. Prefixing C^{-1} to both sides yields that

$$g_1g_2 \ldots g_m \sim C^{-1}Cg_1g_2 \ldots g_m \sim C^{-1}C \sim \varepsilon,$$

completing the proof. ■

Corollary 4.4 *Let (μ, X) be a growth process, where X contains the standard gates and let $\Sigma = supp(\mu)$. The Markov process has periodicity 2 if and only if Σ is alternating.*

Additionally, we make use of a well known result from the Markov Theory.

Theorem 4.5 (Theorem 6.2, Page 132 in [MR95])
Any irreducible, finite, and aperiodic Markov process has the following properties:

1. *All states are ergodic.*
2. *There is a unique limiting distribution π such that for each state s, $\pi(s) > 0$.*
3. *If $N(s,t)$ is the number of times that process visits state s in t steps, then $\lim_{t\to\infty} \frac{N(s,t)}{t} = \pi(s)$.*

Proof of Theorem 4.1: By Lemma 4.2 the growth process is an ergodic and irreducible Markov process. By Lemma 4.3 and by Theorem 4.5, the process has a limiting distribution.

The "otherwise" part follows from the Frobenius-Perron Theorem [Fro12, Per07]. Let M be the transition matrix of the process, and let π_0 be the initial distribution. The ith distribution may be written as $\pi_i = M^i\pi_0$. Following Lovàsz [Lov96], π_i can be written

$$\pi_i = M^i\pi_0 = \sum_{j=1}^{m} \lambda_j^i v_j v_j^T \pi_0,$$

where m is the size of the subgroup of S_{2^n}, described in the proof of Lemma 4.2, and v_j is the jth eigenvector of M. We assume that the eigenvectors are orthogonal and that the eigenvalues, λ_i, are in decreasing order. If the periodicity of the process is 2, then by the Frobenius-Perron Theorem [Fro12, Per07], all but two of the eigenvalues have magnitude less than 1, $\lambda_1 = 1$, and $\lambda_m = -1$. Therefore

$$\begin{aligned} \lim_{i\to\infty} \pi_{2i} &= (v_1 v_1^T + v_m v_m^T)\pi_0, \\ \lim_{i\to\infty} \pi_{2i+1} &= (v_1 v_1^T - v_m v_m^T)\pi_0, \end{aligned}$$

which are the two alternating distributions. ■

The question of whether growth process (μ, X) has a limiting distribution is therefore reduced to whether the support of μ is alternating or not. The answer, in most cases, is that the support is not alternating. The following proposition enumerates sufficient conditions that ensure that the support of μ is not alternating.

Proposition 4.6 *Let (μ, X) be a growth process, let $\Sigma = supp(\mu)$, and assume that X comprises all standard gates that operate on lines $1, \ldots, n$. If one of the following conditions hold:*

1. *the identity gate ε is in Σ,*
2. *$n \geq 2$ and for some distinct $i, j \in [1,n]$, the gates $\bigoplus_i^j$, $\bigoplus_j$, and $\bigoplus_i$ are in Σ (see Figure 4.1),*
3. *$n \geq 3$ and for some distinct $i, j, k \in [1,n]$, the gates $\bigoplus_i^j$, $\bigoplus_j^k$, and $\bigoplus_i^k$ are in Σ (see Figure 4.2),*

4. $n \geq 3$ *and for some distinct* $i,j,k \in [1,n]$, *the gates* $\bigoplus_i^{k\wedge j}$, $\bigoplus_j^k$, *and* $\bigoplus_i^k$ *are in* Σ *(see Figure 4.3),*

5. $n \geq 3$ *and for some distinct* $i,j,k \in [1,n]$, *the gates* $\bigoplus_i^{k\wedge j}$, $\bigoplus_k$, $\bigoplus_j$, *and* $\bigoplus_i$ *are in* Σ *(see Figure 4.4), or*

6. $n \geq 4$ *and for some distinct* $i,j,k,l \in [1,n]$, *the gates* $\bigoplus_i^{k\wedge j}$, $\bigoplus_j^{l\wedge k}$, *and* $\bigoplus_i^{l\wedge k}$ *are in* Σ *(see Figure 4.5),*

then Σ *is not alternating.*

Proof: By inspection. ■

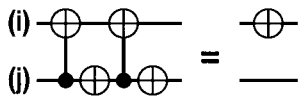


Figure 4.1: A NOT gate.

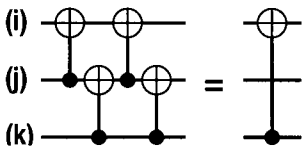


Figure 4.2: A C-NOT gate.

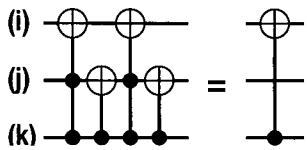


Figure 4.3: A C-NOT gate.

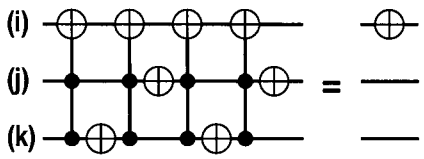


Figure 4.4: A NOT gate.

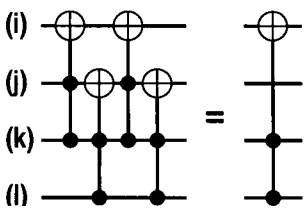


Figure 4.5: A Toffoli gate.

If the support of μ on X is gate-symmetric and the circuit width is greater than or equal to four, then Proposition 4.6 applies. Proposition 4.6 does not apply to growth processes on reversible circuits of width 2 and 3 that only use controlled-NOT gates (and Toffoli gates, respectively)—even-though the support of μ may be gate-symmetric. In general determining whether the support is alternating may be as difficult as determining the support of the growth process.

If a limiting distribution does exist, then it will be a uniform distribution on the support of the growth process. This result stems from the following lemma.

Lemma 4.7 (Lemma 6.3, Page 132 in [MR95])
Let $G = (V,E)$ *be a nonbipartite connected graph, let* $d(v)$ *denote the degree of vertex* $v \in V$, *and let* M *be a transition matrix for a Markov process whose states are the vertices of* G, *such that*

$$M_{u,v} = \begin{cases} d(u)^{-1}, & (u,v) \in E, \\ 0, & (u,v) \notin E \end{cases},$$

where $u,v \in G$. *The limiting distribution* π *is given by* $\pi(v) = \frac{d(v)}{2|E|}$.

Corollary 4.8 *If $G = (V,E)$ is a d-regular, nonbipartite, connected graph, then the limiting distribution of the Markov process defined in Lemma 4.7 is uniform over V.*

Using this corollary we can characterize the shape of the limiting distribution of a growth process on reversible circuits by the support of the growth process. The key observation is that a growth process induces a Cayley graph of the subgroup of S_{2^n}, which is generated by the elements in $\Sigma = supp(\mu)$ and is the support of the process. The vertices correspond to the elements of the subgroup, where the degree of each vertex is equal to the size of Σ. The following theorem relates the support of the process to the limiting distribution.

Theorem 4.9 *Let (μ, X) be a growth process, where μ is a uniform distribution on $\Sigma \subseteq X$ and X comprises the standard reversible gates. If Σ is not alternating, then the limiting distribution is a uniform distribution over the support of the growth process.*

Proof: Since Σ is not alternating, by Theorem 4.1, a limiting distribution exists. The growth process corresponds to the Markov process in Lemma 4.7, where G is the Cayley graph induced by growth process. By Corollary 4.8, the limiting distribution is uniform over the vertices of G, which correspond to the support of the process. ■

Thus, the limiting distribution is completely characterized by the support of the growth process.

4.3 The Support of Growth Processes on Reversible Circuits

The general problem of determining the support of a growth process reduces to the problem of determining what group is generated by a given set of generating elements; this is a notoriously hard problem. In the present context the generating set corresponds to the support of μ, a uniform distribution on the set X of standard gates. Since it is natural to decouple gate type, i.e., NOT, controlled-NOT, or Toffoli, from the lines on which a gate operates, we assume that the support of μ is gate-symmetric. Additionally, under this assumption we can characterize the support of the growth processes. We first discuss processes that are not bandwidth-limited, and then extend the discussion to bandwidth-limited growth processes. In the former case, we first consider processes on circuits with a small number of lines, $n = 2$ (Theorem 4.10) and $n = 3$ (Theorem 4.11), and then we consider processes on larger circuits with various supports of μ (Theorems 4.13–4.15). Similarly, for the bandwidth-limited case, Theorems 4.16, 4.17, and 4.19 characterize growth processes on circuits with bandwidth 1, 3, and 2, respectively, and Theorem 4.18 characterizes growth processes on circuits with bandwidth greater than 3.

4.3.1 The Support of Growth Processes that are not Bandwidth-limited

The maximum fan-in of any gate in the gate set X is three (the fan-in of the Toffoli gate). Consequently, there are three cases, with respect to circuit width, that we need to consider. Namely, we need to consider circuits of width $n = 2$, $n = 3$, and $n > 3$. The first two cases can easily be done by computational enumeration, while the last case requires an analytical approach. The following two theorems characterize the support of growth processes on two and three lines.

Theorem 4.10 *Let (μ, X) be a growth process on two lines, where μ is a uniform distribution on a subset of X such that $\Sigma = supp(\mu)$ is gate-symmetric with respect to X. Then*

1. *If Σ contains both controlled-NOT and NOT gates, then the support of the process corresponds to the group S_4.*
2. *If Σ contains only controlled-NOT gates, then the support of the process corresponds to the group S_3 on the inputs $\{1,2,3\}$ and the input 0 is a fixed-point.*
3. *If Σ contains only the NOT gates, then the support of the process corresponds to the group $C_2 \times C_2$.*

Proof: By enumeration. ■

Theorem 4.11 *Let (μ, X) be a growth process on three lines, where μ is a uniform distribution on a subset of X such that $\Sigma = supp(\mu)$ is gate-symmetric with respect to X. Then*

1. *If Σ contains both Toffoli and NOT gates, then the support of the process corresponds to the group S_8.*
2. *If Σ contains only Toffoli gates, then the support of the process corresponds to the group S_4 on the inputs $\{3,5,6,7\}$ and the remaining inputs are fixed-points.*
3. *If Σ contains both Toffoli and controlled-NOT gates, then the support of the process corresponds to the group S_7 on all inputs except the input 0, which is a fixed-point.*

Proof: By enumeration. ■

If we restrict a growth process on $n > 2$ lines to using only controlled-NOT gates and/or NOT gates, then the resulting support will correspond to a subgroup of affine transformations on the Boolean cube [CG75]. Hence, all interesting growth processes on $n > 2$ lines must use Toffoli gates. Consequently, the three cases in Theorem 4.11 are the three sub-cases that need to be considered when analyzing growth processes on $n > 2$ lines. To do this we use a theorem of Coppersmith and Grossman [CG75].

A k-function, $k \leq n$, is a permutation, $\sigma \in S_{2^n}$, on the Boolean cube defined by

$$((a_0 a_2 ... a_{n-1})\sigma)_m = \begin{cases} a_m, & m \neq j, \\ a_j \oplus f(a_{i_1}, a_{i_2}, ..., a_{i_k}), & m = j \end{cases}, 0 \leq j \leq n-1,$$

where $a = (a_0 a_1 ... a_{n-1}) \in B_n$, and f is any k-adic Boolean function. If f is a singleton function, i.e., true for only one input, then the k-function, σ, is also called a **singleton**. For a fixed k, the **basic set** comprises the $2^k \binom{n}{k+1}$ k-functions, corresponding to each of the 2^k singleton functions and the $\binom{n}{k+1}$ possible choices of inputs and output.

Theorem 4.12 (Coppersmith and Grossman, [CG75])
Let $G_{k,n}$ denote the subgroup of S_{2^n} generated by basic set of k-functions, where $n > 1$. Then

1. *If $n > 3$ and $1 < k < n-1$, then $G_{k,n} = A_{2^n}$.*
2. *$G_{n-1,n} = S_{2^n}$, $(k = n-1)$.*
3. *$G_{1,n}$ is the group of affine transformations on B_n, (k=1).*

A reversible gate that takes k control lines, computes a Boolean value, and performs an XOR with the result on another line, realizes a k-function. Concretely, a Toffoli gate realizes a singleton 2-function—the conjunction of the two control lines—and placing NOT gates on one or both control lines, before and after the gate, realizes the remaining three singleton 2-functions; see Figure 4.6. Thus, Theorem 4.12 can be used to characterize the support of growth processes on $n > 3$ lines that use Toffoli and NOT gates.

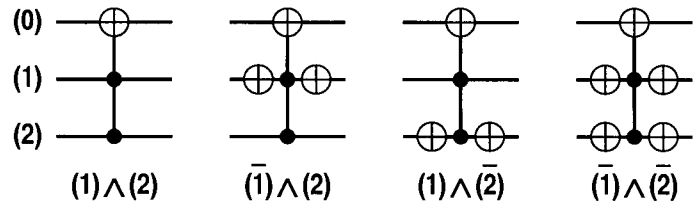


Figure 4.6: The four singleton 2-functions.

Theorem 4.13 *Let (μ, X) be a growth process on $n > 3$ lines, where μ is a uniform distribution on a subset of X such that $\Sigma = supp(\mu)$ is gate-symmetric with respect to X and contains both the Toffoli and NOT gates. Then the support of the growth process corresponds to the alternating group A_{2^n}.*

Proof: First, each of the four singleton 2-functions is a realized by a Toffoli gate along with 0, 2, or 4 NOT gates that bracket the control lines of the Toffoli gate. Second, one of the four singleton 2-functions corresponds to a Toffoli gate, and a sequence of the four singleton 2-functions corresponds to the NOT gate. Thus, by Theorem 4.12, the support of the growth process corresponds to the alternating group. ■

If the support of μ does not contain both Toffoli and NOT gates, i.e., the support comprises either solely Toffoli gates or, both Toffoli and controlled-NOT gates, then the support of the corresponding growth process is an alternating group, on the set of inputs of weight greater than one and zero, respectively; a similar phenomenon is observed in our analysis of growth processes on two and three lines. The subsequent two theorems formalize this notion.

Let $A_{2,n}$ denote the set of all even permutations on the Boolean cube B_n where all inputs of weight less than two are fixed-points; the set $A_{2,n}$ corresponds to the alternating group A_{2^n-n-1} on all points of B_n that are of weight two or more. Similarly, let $A_{1,n}$ denote the set of all even permutations on the Boolean cube B_n where the input of weight zero is a fixed-point. Naturally, $A_{1,n}$ corresponds to the alternating group A_{2^n-1} on all points of B_n except the one of weight zero.

Theorem 4.14 *Let (μ, X) be a growth process on $n > 3$ lines, where μ is a uniform distribution on a subset of X such that $\Sigma = supp(\mu)$ is gate-symmetric with respect to X and contains only the Toffoli gates. Then the support of the growth process is $A_{2,n}$.*

Proof: Since Σ is a proper subset of the the support of μ in Theorem 4.13, the corresponding subgroup must be a subgroup of A_{2^n}, and hence cannot have any odd permutations. Furthermore,

Toffoli gates are triggered only when both their control lines are 1. Thus, every input of weight less than two will not trigger any Toffoli gate and cannot be permuted by a circuit of Toffoli gates. Consequently, the $n+1$ inputs of weight less than two must be fixed-points and the support of the growth process can be at most $A_{2,n}$. Thus, we only need to argue that the growth process can realize all functions in $A_{2,n}$.

The proof is by induction on n. The base case, $n = 4$, follows from the fact that the 3-cycles that generate $A_{2,4}$ can be constructed from at most five Toffoli gates; these are listed in Table 4.1 and were determined via computational enumeration. The generators for $A_{2,4}$, which correspond to

$$\begin{array}{ll} \oplus_4^{1\wedge 2}\oplus_4^{1\wedge 3}\oplus_3^{2\wedge 4}\oplus_4^{1\wedge 3}\oplus_3^{2\wedge 4} \sim (3\ 7\ 11) & \oplus_3^{1\wedge 2}\oplus_3^{1\wedge 4}\oplus_2^{3\wedge 4}\oplus_3^{1\wedge 2}\oplus_2^{3\wedge 4} \sim (9\ 11\ 13) \\ \oplus_4^{1\wedge 2}\oplus_4^{1\wedge 3}\oplus_2^{3\wedge 4}\oplus_4^{1\wedge 2}\oplus_2^{3\wedge 4} \sim (5\ 7\ 13) & \oplus_3^{1\wedge 2}\oplus_3^{2\wedge 4}\oplus_1^{3\wedge 4}\oplus_3^{1\wedge 2}\oplus_1^{3\wedge 4} \sim (10\ 11\ 14) \\ \oplus_4^{1\wedge 2}\oplus_4^{2\wedge 3}\oplus_1^{3\wedge 4}\oplus_4^{1\wedge 2}\oplus_1^{3\wedge 4} \sim (6\ 7\ 14) & \oplus_2^{1\wedge 3}\oplus_1^{2\wedge 4}\oplus_2^{1\wedge 3}\oplus_1^{2\wedge 4}\oplus_2^{3\wedge 4} \sim (12\ 14\ 15) \end{array}$$

Table 4.1: The 3-cycles that span $A_{2,4}$.

the 12 different Toffoli gates, are listed in Table 4.2; however, by Proposition 4.6, only eight are necessary. For a fixed $n > 3$ assume that the support of a growth process on n lines is $A_{2,n}$. We now

$$\begin{array}{lll} \oplus_3^{1\wedge 2} \sim (3\ 7)(11\ 15) & \oplus_4^{1\wedge 2} \sim (3\ 11)(7\ 15) & \oplus_2^{1\wedge 3} \sim (5\ 7)(13\ 15) \\ \oplus_4^{1\wedge 3} \sim (5\ 13)(7\ 15) & \oplus_2^{1\wedge 4} \sim (9\ 11)(13\ 15) & \oplus_3^{1\wedge 4} \sim (9\ 13)(11\ 15) \\ \oplus_1^{2\wedge 3} \sim (6\ 7)(14\ 15) & \oplus_4^{2\wedge 3} \sim (6\ 14)(7\ 15) & \oplus_1^{2\wedge 4} \sim (10\ 11)(14\ 15) \\ \oplus_3^{2\wedge 4} \sim (10\ 14)(11\ 15) & \oplus_1^{3\wedge 4} \sim (12\ 13)(14\ 15) & \oplus_2^{3\wedge 4} \sim (12\ 14)(13\ 15) \end{array}$$

Table 4.2: Generators of $A_{2,4}$.

argue that the support of growth processes on $n+1$ is $A_{2,n+1}$.

To prove the inductive step we show that all permutations in $A_{2,n+1}$, which can be represented by a 3-cycle, can be realized; this implies that all functions $A_{2,n+1}$ are realizable. In fact, it suffices to show that at least one 3-cycle can be realized. Fix the permutation $\sigma' = (3\ 7\ 15)$, let circuit $C' \sim \sigma'$, which by the induction hypothesis can be realized by the growth process on $n > 3$ lines; and assume that C' comprises the lines labeled $1, \ldots, n$. For $\sigma = (7\ 15\ 31)$, we construct $C \sim \sigma$, by modifying C' in the following manner. Assuming that the $n+1$ line in C is labeled 0, replace each Toffoli gate in C' by the construction of Barenco et al. [BBD+95], exhibited in Figure 4.7. Thus, circuit C, exhibited in Figure 4.8, is a controlled version of C' that is controlled by line 0, that is, $C \sim \sigma$.

To construct a permutation represented by the 3-cycle $(x\ 15\ 31)$, where $|x| > 1$, we construct an additional circuit C'' such that $C'' \sim \tau = (x7)(yz)$ where y and z are not 15 and 31, and conjugate σ by τ to yield $\tau\sigma\tau^{-1} = (x\ 15\ 31)$. The circuit for τ can be realized in a manner similar to realizing circuit C. Thus, every permutation in $A_{2,n+1}$, which is represented by a 3-cycle can be realized, implying that every permutation in $A_{2,n+1}$ can be realized by the growth process. ■

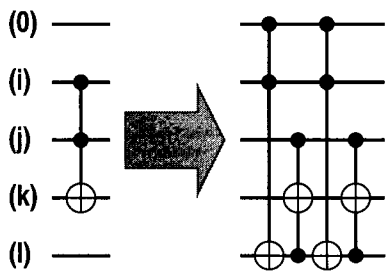


Figure 4.7: A (0)-controlled Toffoli.

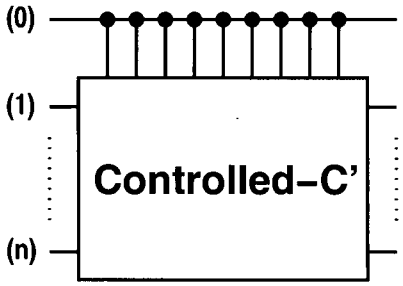


Figure 4.8: The circuit $C \sim (7\ 15\ 31)$.

Theorem 4.15 *Let (μ, X) be a growth process on $n > 3$ lines, where μ is a uniform distribution on a subset of X such that $\Sigma = supp(\mu)$ is gate-symmetric with respect to X and contains the Toffoli and controlled-NOT gates. Then the support of the growth process is $A_{1,n}$.*

Proof: Same argument as in Theorem 4.14. ■

Since the limiting distribution, if one exists, is uniform over the support of the growth process, the limiting distribution of non-bandwidth-limited is uniform over a large fraction of the symmetric group. One way to reduce the support is to use bandwidth-limited growth processes.

4.3.2 The Support of Growth Processes that are Bandwidth-limited

The support of bandwidth-limited growth processes is significantly smaller than the support of non-bandwidth-limited growth processes. Since some fraction of the lines are fixed, i.e., read-only, the realizable permutations correspond to a quotient group of the alternating group A_{2^n}. The n lines of a bandwidth-limited growth process comprise $r < n$ read-only lines and $b = n - r$ read-write lines. Without loss of generality, we assume that the first r lines, $1, \ldots, r$ are read-only and the remaining lines, $r+1, \ldots, n$ are read-write. Consequently, all realizable permutations are elements of the quotient group A_{2^n}/B_r, i.e., each element can be factored into 2^r disjoint permutations, denoted σ^x, such that the points of each permutation share the same unique r-bit prefix, $x \in B_r$. That is, for input xx', x remains fixed and x' is permuted by σ^x, where σ^x is a function of x.

Since some lines are read-only, the support of distribution μ is naturally restricted to the set of gates that only modify read-write lines. In this context, the notion of gate-symmetry is therefore taken to be with respect to X_b, the set of gates that may comprise a circuit of bandwidth b.

The analysis of bandwidth-limited growth processes can be divided into four parts. Namely, we consider processes where the number of read-write lines is 1, 2, 3, or greater than 3, i.e., $b = 1$, $b = 2$, $b = 3$, or $b > 3$. In all but the $b = 2$ case the respective support of the growth processes can be explicitly obtained. In the case of $b = 2$, determining the support is related to an open question that is discussed in Chapter 5. We discuss the cases where $b \neq 2$ and then look at what happens when $b = 2$.

Theorem 4.16 *Let (μ, X_1) be a bandwidth-limited growth process on $n > 1$ lines where $b = 1$, $r = n - 1$ and $\Sigma = supp(\mu)$ is gate-symmetric.*

1. *If Σ contains only the controlled-NOT gates, then the support of the process corresponds to permutations σ of the form:*

$$(x_1, \ldots, x_{n-1}, x'_1)\sigma = (x_1, \ldots, x_{n-1}, x'_1 \oplus f_1(x)),$$

   where $f_1(x) = \oplus_{i=1}^{n-1} c_i x_i$ and $c_i \in \{0, 1\}$.
2. *If Σ contains only Toffoli gates, then the support of the process corresponds to permutations σ of the form:*

$$(x_1, \ldots, x_{n-1}, x'_1)\sigma = (x_1, \ldots, x_{n-1}, x'_1 \oplus f_2(x)),$$

   where $f_2(x) = \oplus_{i=1}^{n} \oplus_{j=i+1}^{n} c_{i,j} x_i \wedge x_j$ and $c_{i,j} \in \{0, 1\}$.
3. *If Σ contains both Toffoli and controlled-NOT gates, then the support of the process corresponds to permutations σ of the form:*

$$(x_1, \ldots, x_{n-1}, x'_1)\sigma = (x_1, \ldots, x_{n-1}, x'_1 \oplus f_1(x) \oplus f_2(x)).$$

4. *If the NOT gate is also present then the size of each of the supports is doubled, containing the negation of the functions f_1 and f_2 as well.*

Proof: Since all gates correspond to elements that commute with each other and each gate is its own inverse, all circuits have a canonical form, i.e., either a circuit has one instance of the gate or not. This corresponds to taking a linear sum (modulo 2) of the outputs of the gates and the last line of the circuit. ■

In the present context, disallowing certain gates from Σ results in the same phenomenon that was observed in the preceding subsection, namely, certain inputs are fixed-points of any permutation realized by the growth process. Since we are interested in the general structure of the respective support, we restrict our attention to growth processes whose distribution, μ, is uniform over the entire set X_b, i.e., the set Σ is gate-symmetric and contains all the NOT gates, the controlled-NOT gates, and the Toffoli gates. We now characterize the support of growth processes on circuits with three read-write lines and processes on circuits with more than three read-write lines.

Theorem 4.17 *Let (μ, X_3) be a bandwidth-limited growth process on $n > 3$ lines, where $b = 3$, $r = n - 3$ and μ is a uniform distribution on X_3. Then, the support of the growth process comprises all permutations σ of the form $(xx')\sigma = x(x'\tau\sigma^x)$, where τ is a permutation on lines $n-2$, $n-1$, and n, which is independent of x, and σ^x is a permutation on the three lines that is even and depends on x.*

Proof: To prove this we use a modification of Barrington's [Bar85, Bar89] technique, which was described in Section 2.5.2. Let ρ be a permutation on B_3, and let circuit C be a realization of ρ on the lines $n-2$, $n-1$, and n. We first need to show that if ρ is even, there is a corresponding controlled circuit $C^{(i)}$, i.e., that there exists a circuit that realizes permutation ρ on lines $n-2$, $n-1$, and n, if

and only if line i, $i \leq r$, has a value of 1. Since we cannot implement a controlled-Toffoli gate on three lines [CG75], we cannot use the construction in Theorem 4.13. However, we can leverage the fact that ρ is even to avoid this problem.

By Theorem 4.11 we know that circuit C exists. Since the only gates that realize an odd permutation on the three read-write lines are Toffoli gates and ρ must be even, there must be an even number of Toffoli gates in C. Since there are three types of Toffoli gates $\bigoplus_n^{n-2\wedge n-1}$, $\bigoplus_{n-1}^{n-2\wedge n}$, and $\bigoplus_{n-2}^{n-1\wedge n}$, we replace the latter two with the former, via the construction illustrated in Figure 4.9. Note, that this does not change the number of Toffoli gates. Next replace all controlled-NOT gates with Toffoli gates and all NOT gates with controlled-NOT gates, connecting the additional input to line i, i.e., all controlled-NOT and NOT gates become (i)-controlled gates. Observe that if line i has value 0, then none of the controlled gates will work; this leaves only the Toffoli gates, which are all of the same type. Since Toffoli gates are their own inverses and there is an even number of them, their computations will cancel each other out. Thus, if i has value 0, the resulting permutation is the identity. However, if i has value 1, then all the controlled gates will trigger if and only they would have triggered in the original circuit C; the resulting permutation is ρ. Thus, we have constructed a controlled circuit $C^{(i)}$ that computes the even permutation ρ on lines $n-2$, $n-1$, and n.

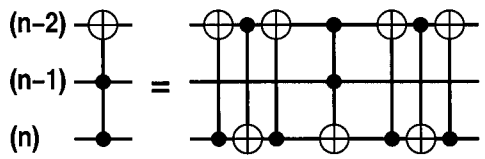


Figure 4.9: A simulation of one Toffoli gate with another.

Since we can construct controlled circuits that can compute any permutation $\rho \in A_8$, using Barrington's [Bar89] conjunction construction we can construct a circuit C^x which realizes the permutation σ^x, that is, the circuit performs permutation ρ on lines $n-2$, $n-1$, and n if and only if the lines $1, \ldots, r$ have the value x.

A controlled circuit that realizes an odd permutation can not be constructed. Using contradiction, suppose that we were able to construct a controlled circuit $C^{(i)}$ where C realizes an odd permutation ρ. There exists an even permutation τ—which can be realized by a controlled circuit—such such that $\rho\tau \sim \bigoplus_n^{n-2\wedge n-1}$, i.e., realizes a Toffoli gate. Hence, the controlled circuit realizes a Toffoli gate with three control lines, something that is impossible according to Coppersmith and Grossman [CG75].

To complete the proof we show how an oblivious permutation on the lines $n-1$, $n-1$, and n is realized. Consider a circuit comprising a $C^{(i)}$ control circuit and an oblivious circuit D on the three lines. Let $\sigma^{x_i} \sim C^{(i)}$ and let $\tau \sim D$. Observe that $\sigma^{x_i}\tau = \tau(\tau^{-1})^{x_i}\sigma^{x_i}\tau^{x_i} = \tau(\tau^{-1}\sigma\tau)^{x_i}$; note that $\tau^{-1}\sigma\tau$ is also an even permutation. Thus, $C^{(i)}D \sim DC'^{(i)}$, where $C' \sim \tau^{-1}\sigma\tau$. Note that the oblivious circuit D can realize any permutation in S_8 on the lines $n-2$, $n-1$, and n. ■

A corollary of Theorem 4.17 is the following theorem, which characterizes the support of bandwidth-b growth processes for $b > 3$. The proof parallels that of Theorem 4.17, except that

the Toffoli transformation argument is unnecessary because the construction technique used in Theorem 4.14 suffices.

Theorem 4.18 *Let (μ, X_b) be a bandwidth-limited growth process on $n > b > 3$ lines, $r = n - b$, and μ is a uniform distribution on X_3. Then, the support of the growth process comprises all permutations σ of the form $(xx')\sigma = x(x'\tau\sigma^x)$, where τ is an even permutation on the b read-write lines that is independent of x, and σ^x is an even permutation on the b read-write lines that depends on x.*

Proof: Since only even permutations are realizable on the lines $n-r+1,\ldots,n$, the parity of τ must also be even. Similarly, any even permutation ρ on B_b, can be realized by some circuit C. For any line i, $i \in [1,r]$, the circuit can be transformed into an (i)-controlled circuit via the construction depicted in Figure 4.8. The rest of the proof mirrors Theorem 4.17. ■

Unfortunately, the case when $b = 2$ is much more challenging. Unlike the two proceeding characterizations, characterizing the support of a bandwidth-2 growth process with currently known techniques is not possible. In fact, such a characterization would go a long way to solving a problem that has remained open for nearly 20 years; see Chapter 5 for an in depth discussion of the problem.

We complete this section by proving that a natural set of functions that are in the support of bandwidth-b growth processes, where $b > 2$, are not in the support of bandwidth-2 growth processes.

Theorem 4.19 *Let (μ, X_2) be a bandwidth-limited growth process on $n > 2$ lines, where $b = 2$, $r = n - 2$, and μ is a uniform distribution on X_2. If a realizable permutation is of the form $(xx')\sigma = x(x'\sigma^x)$, where σ^x is a permutation on lines $n-1$ and n, and is performed for only a specific value of x, then σ^x must be even.*

Proof: The proof consists of two steps. First, we argue that for any i, $1 \leq i \leq r$, and any permutation on lines $n-1$ and n of a bandwidth-2 circuit, an (i)-controlled and $(\bar{i})$-controlled circuits that realize the permutation can be constructed. Second, we show that if σ^x, for a particular x, can be realized, then σ^x for any x can be realized. Finally, without loss of generality, we fix x to be $\mathbf{1}$, and argue that any circuit that realizes σ^x must realize an even permutation.

Any permutation on B_2 may be constructed from three permutations, (01), (02), and (03). Figure 4.10 illustrates the controlled realizations for each of these permutations. Thus, any (i)-controlled or $(\bar{i})l$-controlled permutation on lines $n-1$ and n is realizable by combining these constructions in the natural way.

Suppose we have a circuit C that performs permutation σ^y if $x = y \neq \mathbf{1}$ and the identity if input $x \neq y$. To construct a circuit C' that realizes the same permutation if and only if $x = \mathbf{1}$, we replace all gates that are controlled by lines i such that $y_i = 0$. Since each such gate realizes an (i)-controlled permutation, we replace it with a circuit that realizes the $(\bar{i})$-controlled permutation. The only exception is a Toffoli gate that is doubly controlled by lines i and j. However, each Toffoli gate can be replaced by a sequence of gates that are not doubly controlled. For example, consider a Toffoli gate that is controlled by lines i and j, $i, j < n-1$ and operates on line $n-1$. This may be written as $((01)(23))^{(i)\wedge(j)}$. Mimicking a technique of Barrington's [Bar85], this gate may be replaced by four singly controlled permutations $(012)^{(i)}$, $(132)^{(j)}$, $(021)^{(i)}$, $(123)^{(j)}$. Once all doubly controlled gates are eliminated, the transformation is applied.

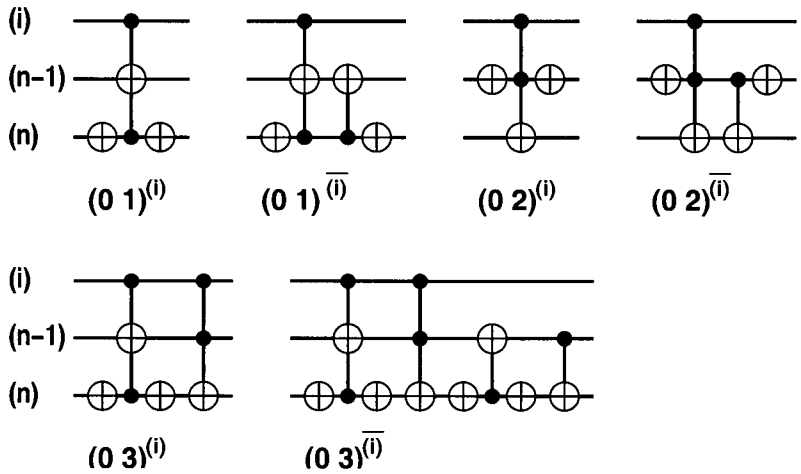


Figure 4.10: Realizations of controlled 2-cycles.

Thus, without loss of generality we limit our attention to circuits that compute permutation $\sigma^{\mathbf{1}}$ if $x = \mathbf{1}$, and ε if $x \neq \mathbf{1}$. Let C be such a circuit and by the preceding argument, assume that C comprises a sequence of uncontrolled and controlled circuits $C^{(i)}$, $1 \leq i \leq r$. If any of the r lines are 0, then C performs the identity permutation, ε, on the two read-write lines, and if all lines are 1 then the permutation $\sigma^{\mathbf{1}}$ is performed. Thus, if all lines are 0 then only gates that are not controlled by the read-only lines will operate. Since the circuit performs ε, which is an even permutation, the uncontrolled parts of the circuit must realize an even permutation. Now consider all singleton inputs, i.e., where line i has value 1 and all other read-only lines are 0. On such an input, all parts of the circuit controlled by line i will operate, however, since C must also perform the identity permutation, the parity of the permutations performed by the controlled parts must be even. Thus, since parity is preserved, $\sigma^{\mathbf{1}}$ must be even. Given any even permutation on the two read-write lines, a circuit of length $O(2^n)$ can be constructed by mirroring the construction in [Bar85]. ■

While the support of bandwidth-2 growth process includes a large number of permutations, it lacks the natural set of permutations that perform a odd permutations for a single input. At the same time, the support is much more complicated than that of bandwidth-1 growth processes. The nature of the support of bandwidth-2 growth processes plays an important role in a well known open problem regarding the computational power of bandwidth-4 permutation branching programs, which are studied in Chapter 5.

4.4 Convergence of Growth Processes on Reversible Circuits

We have shown that a growth process on reversible circuits will either have a limiting distribution or two alternating distributions. Unlike in the case of growth process on formulas, the limiting distribution will be uniform on the support of the growth process. In fact, the size of the support is usually quite large, on the order of $\Theta((2^b!)^{2^{n-b}} n^{-c})$, where b is the bandwidth of the growth process

and c is some positive constant. Furthermore, the support of μ, the distribution on the gate set, is relatively small, on the order of $O(bn^2)$ gates. Hence, it is not surprising that the convergence rate of such processes is relatively slow, compared to growth processes on formulas.

To bound the rate of convergence we refer to the current literature on random walks on groups and look at the Cayley graph induced by the growth process. The graph is d-regular, where the degree d, is equal to the cardinality of $\Sigma = supp(\mu)$, and hence bounded by $O(bn^2)$. Additionally, the size of the graph is equal to the size of the support of the growth process, and hence, is on the order of $\Theta((2^b!)^{2^{n-b}} n^{-c})$. Thus, the diameter of the graph is bounded from below by $\Omega(\frac{b2^n}{\log n})$, which means that the convergence rate is also bounded from below by $\Omega(\frac{b2^n}{\log n})$, and for non-bandwidth-limited growth processes will in fact be polynomial in 2^n. Thus, the convergence of growth processes on reversible circuits is significantly slower than in the case of growth processes on formulas.

Using well known bounds we can bound the second eigenvalue; thus, bounding the rate of convergence. The lower bound inequality, which was "discovered by many people" [Fri91] such as McKay [McK81] and Alon [Alo86], bounds the second eigenvalue from below,

$$\lambda_2 \geq \frac{2\sqrt{d-1}}{d} + O\left(\frac{1}{d \log_d n}\right).$$

To bound the second eigenvalue from above we use the inequality exhibited by Diaconis and Saloff-Coste [DSC93], versions of which also appeared in [Ald87] and in [Bab91]:

$$\lambda_2 \leq 1 - \frac{1}{dl^2},$$

where l is the diameter of G. Thus, the second eigenvalue is bounded by

$$\frac{1}{O(n\sqrt{b})} + O\left(\frac{\log n}{b^2 n^2 2^n}\right) \leq \lambda_2 \leq 1 - O\left(\frac{(\log n)^2}{b^3 n^2 2^{2n}}\right).$$

Thus, the rate of convergence will be polynomial in the diameter of the Cayley graph.

Chapter 5

Reversible Circuit Complexity

In this chapter we investigate the reversible circuit complexity of finite Boolean functions. This chapter is divided into three parts. In Section 5.2 we investigate the complexity of finite Boolean functions in the context of bounded bandwidth reversible circuits, in particular, when the bandwidth is on the order of a couple of lines. The problem of characterizing the support of growth processes on bandwidth-2 reversible circuits, which was discussed in Chapter 4, is closely related to the problem of determining the complexity of conjunction within the context of bandwidth-2 reversible circuits. Section 5.3 discusses polylogarithmic bandwidth reversible circuits, defines a natural hierarchy of classes of problems that are decidable by reversible circuits with polylogarithmic bandwidth, and relates the hierarchy to the **SC** hierarchy. Finally, in Section 5.4 we investigate the complexity and constructions of several common families of Boolean functions and obtain some general criteria for bounding the complexity of finite Boolean functions within the context of non-bandwidth-limited reversible circuits.

5.1 Definitions

Recall from Chapter 4 that a reversible circuit C on n lines comprises a sequence of m gates, $C = g_1 \cdots g_m$, each of which is controlled by zero, one, or two lines, and each of which XOR another line with the conjunction of the control lines; the gates are the NOT gate ($\oplus_i$), the controlled-NOT gate ($\oplus_i^j$), and the Toffoli gate ($\oplus_i^{j \wedge k}$), where i, $i = 1, \ldots, n$, label the lines of the circuit. The output of circuit C on input $x \in B_n$, is denoted $C(x) \in B_n$, and the composition and inverse of circuits corresponds respectively to the concatenation and reversal of the circuits' gate sequences. A circuit C is **bandwidth-limited** if at least one line in the circuit is read-only; a **bandwidth-b** circuit, where $b \leq n$, comprises b read-write lines and $r = n - b$ read-only lines. We assume that lines $1, \ldots, r$ are read-only lines and the remaining lines, $r+1, \ldots, n$, are read-write lines.

Each reversible circuit realizes a permutation on the Boolean cube B_n, which corresponds to an element of the group S_{2^n}. We write $C \sim \sigma \in S_{2^n}$ if C realizes σ and we write $C \sim C'$ if C and C' realize the same permutation. An **(i)-controlled** circuit, denoted $C^{(i)}$, performs two different permutations depending on the value of line i. If line i has value 1, the circuit performs a permutation in which line i is fixed, otherwise the circuit performs the identity permutation. Additionally, we say that a circuit

is x-controlled, for some $x \in B_k$, $k < n$, if the circuit, denoted C^x, is controlled by a fixed subset of lines of size k, called control lines. If the lines hold the value x, then C^x performs a permutation in which the control lines are fixed, and performs the identity permutation otherwise. Unless otherwise stated, $x \in B_r$, and the control lines are the read-only lines, $1, \ldots, r$.

Using the definitions of Barrington [Bar85, Bar89], a width-w permutation branching program (w-PBP) P on n variables $x_1, \ldots, x_n$ is a sequence of instructions of the form σ^{x_i} or σ^1, where instruction σ^{x_i} **yields** permutation $\sigma \in S_w$ if $x_i = 1$ and yields the identity, ε, otherwise; instruction σ^1 always yields σ. On input x, program P yields a permutation $P(x) \in S_w$, where $P(x)$ is the product of the yields of the instructions of P. A program P over group G is defined in a similar manner [BST90], except that each instruction, σ^{x_i}, yields either a group element $\sigma \in G$, or the identity, and the yield of the program, $P(x)$, is also an element of G. Thus, w-PBPs are also programs over groups where $G = S_w$. The **length** of program P, denote $|P|$, is the number of instructions in P.

Let P be a program over group G on n variables, $x_1, \ldots, x_n$. Program P **strongly accepts** language $L \subseteq B_n$ if there exists a $\sigma \in G$, $\sigma \neq \varepsilon$, such that for all $x \in B_n$

$$P(x) = \begin{cases} \varepsilon, & x \notin L \\ \sigma, & x \in L \end{cases}.$$

Program P **weakly accepts** language $L \subseteq B_n$ if for all $x \in B_n$

$$P(x) = \begin{cases} \varepsilon, & x \notin L \\ \sigma \neq \varepsilon, & x \in L \end{cases}.$$

Since every finite group G can be embedded into a permutation group S_w for appropriate w, the following definition makes sense for programs over any finite group. Adapting a definition due to Borodin et al. [BDFP83], program P **monotonically accepts** $L \subseteq B_n$ if there exists a point $p \in [1, w]$, such that for all $x \in B_n$, p is not a fixed-point of yield $P(x)$ if $x \in L$, and $P(x) = \varepsilon$ if $x \notin L$. Observe that all strongly accepting programs are also monotonically accepting and all monotonically accepting programs are also weakly accepting. Additionally, if f is the characteristic Boolean function of L, i.e., $f(x) = 1$ if and only if $x \in L$, then we say that program P **strongly computes**, **weakly computes**, or **monotonically computes** function f.

Just as in the case of reversible circuits, branching programs may be composed in several ways. The concatenation of two programs P and P', denoted PP', is the concatenation of the instruction sequences of P and P'. Since the inverse of a concatenation $(PP')^{-1}$ is $P'^{-1}P^{-1}$, the inverse of a program P is the inverted sequence and reversed of the instructions of P. A **commutator** of programs P and P', $[P, P'] = PP'P^{-1}P'^{-1}$, mirrors the group-theoretic definition of a commutator. Finally, the kth power of P is the concatenation of k copies of P, i.e., $P^k = PP \ldots P$.

Let P be a program over group G. A **program transformation** Ψ takes a program P as an argument and yields a new program via a basic set of operations. A transformation is either

1. a constant: $\Psi(P) = \tau^1$ for some $\tau \in G$,
2. a projection: $\Psi(P) = P$,
3. an inverse: $\Psi(P) = P^{-1}$, or
4. a concatenation: $\Psi(P) = \Phi(P)\Upsilon(P)$ of two transformations.

For conciseness, the composition of transformations, $\Psi(\Phi(P))$, is commonly used, but is not a basic operation because it is constructible from the preceding rules. An important feature is that the resulting program $\Psi(P)$ has length that is only a constant factor larger than the original program. Concretely, a transformation is a word comprising placeholders and oblivious instructions, e.g., τ^1, where each placeholder is replaced by an instance of program P, or its inverse, when the transformation is applied to P.

Applying a transformation to a constant program, which is a permutation, yields a permutation. Hence, a program transformation is also map from G to itself. This concise characterization is used extensively in our discussion; however, while each transformation is a map from G to itself, not all maps are valid transformations.

5.2 Bounded Bandwidth Reversible Circuits

Bounded bandwidth reversible circuits share many similarities with bounded width permutation branching programs (w-PBP). Thus, it is not surprising that many of the results obtained within the context of the permutation branching program framework are also applicable to the reversible circuit model; particularly, within the framework of bounded bandwidth reversible circuits. Thus, we first derive the explicit relationship between bounded bandwidth reversible circuits and bounded permutation branching programs, and then use the PBP framework to characterize bounded bandwidth reversible circuits.

Theorems 5.1 and 5.3 show the equivalence between bandwidth-2 reversible circuits and width-4 permutation branching programs. While Theorem 5.16 shows how weakly accepting 4-PBPs can be transformed into monotonically accepting 4-PBPs of comparable size, Theorem 5.18 shows that weakly accepting 4-PBPs cannot be transformed into strongly accepting 4-PBPs. This is in contrast to Theorem 5.24, which shows how to transform a weakly accepting PBP of width 5 or greater into a strongly accepting PBP!

5.2.1 Simulating Reversible Circuits with Branching Programs

In order to compare the two models we need to define what it means for one model to simulate the other. A width-w permutation branching program P realizes a map from the Boolean cube to the symmetric group, $P : B_n \rightarrow S_w$, and a reversible circuit C realizes a map $C : B_n \rightarrow B_n$. However, a bandwidth-b reversible circuit C can be viewed in a different manner. Since the circuit comprises b read-write lines and r read-only lines, depending on the values of the r read-only lines, circuit C performs a permutation on the b read-write lines. Thus, C may also be viewed as a map of the form $C : B_r \rightarrow S_{2^b}$. We say that a bandwidth-b reversible circuit C **simulates** a w-PBP on n variables, if C realizes the map $C : B_r \rightarrow S_{2^b}$, where $r = n$ and $w \leq 2^b$, such that C^x performs permutation $P(x)$ on the b read-write lines. Similarly, a w-PBP on n variables simulates a bandwidth-b circuit, if $w = 2^b$, $r = n$, and $P(x)$ is the permutation that C^x performs on the b read-write lines.

One immediate observation is that 2-PBPs cannot simulate bandwidth-1 reversible circuits if the circuits contain Toffoli gates. Since each instruction of a 2-PBP depends on at most one variable,

i.e., σ^{x_i}, and $\sigma \in S_2$ must either be an identity, or the permutation (01), then each 2-PBP yields $(01)^{f(x)}$, where $f(x) = c_0 \oplus \bigoplus_{i=1}^{n} c_i x_i$, and $c_i \in \{0,1\}$. Namely, 2-PBPs can strongly compute any linear function and cannot compute, even weakly, any nonlinear function. Since the Toffoli gate computes a conjunction of two inputs, no 2-PBP can simulate a Toffoli gate.

Another negative simulation result is that if $2^b - 2 \leq w \leq 2^b$, then bandwidth-b reversible circuits cannot simulate all w-PBPs. A corollary of the results of Coppersmith and Grossman [CG75] is that odd parity permutations cannot be realized by reversible circuits on more than three lines. Thus, no instruction that yields an odd permutation can be simulated by a reversible circuit. However, if $w < 2^b - 2$, there exist two points, $p_1 = 2^b - 1$ and $p_2 = 2^b - 2$, such that any odd permutation $\sigma \in S_w$, can be embedded into A_{2^b} as the even permutation $\sigma(p_1 p_2) \in A_{2^b}$. Thus, the permutations yielded by the program P are preserved. Fortunately, 4-PBPs and bandwidth-2 reversible circuits can simulate each other. For the remainder of our discussion we assume that all gates, Toffoli, controlled-NOT, and NOT gates, are available.

Theorem 5.1 *If P is a 4-PBP on n variables and length m, then there exists a bandwidth-2 reversible circuit of length $4m$ that simulates P.*

Proof: There are two types of instructions that we need to consider, controlled instructions, σ^{x_i}, and oblivious instructions, σ^1. Let σ^{x_i} be an instruction of P. By Theorem 4.19, the corresponding controlled circuit, $C^{(i)}$ can be realized by concatenating in some order the controlled circuits that realize the permutations (01), (02), and (03). In fact, via computational enumeration, all 24 permutations can be realized in four gates or less. An oblivious instruction can be realized in nearly identical manner by replacing all controlled gates with uncontrolled ones. Since each instruction can be simulated by at most four gates, the result follows. ■

An immediate corollary of Theorem 5.1 is that any w-PBP P, $w < 2^b - 1$, can be simulated by a bandwidth-b circuit C; furthermore, $|C| \in O(|P|)$.

Corollary 5.2 *If P is a w-PBP on n variables such that $w < 2^b - 1$, for some b, then there exists a bandwidth-b reversible circuit of length $O(|P|)$ that simulates P.*

Proof: By Theorems 4.17and 4.18, any even permutation on the b read-write lines may be realized either in a controlled or oblivious form. Since every permutation in S_w can be embedded into A_{2^b}, using a number of gates that depends only on b, a w-PBP P can be simulated by a bandwidth-b circuit that comprises $O(|P|)$ gates. ■

The converse of Theorem 5.1 is also true.

Theorem 5.3 *If C is a bandwidth-2 circuit comprising n read-only lines and m gates, then there exists a 4-PBP of length $4m$ that simulates C.*

Proof: To prove the latter, we need to show that each reversible gate: NOT, controlled-NOT, and Toffoli gate, can be simulated by one or more instructions. There are three classes of gates that we need to consider: gates that are not attached to any of the read-only lines, gates that are attached to exactly one of the read-only lines, and gates that are attached to two read-only lines. Each gate

that is not attached to any read-only lines and realizes a permutation, σ, on the two lines, can be simulated by a single oblivious instruction σ^1. Similarly, each gate that is attached to exactly one read-only line, which performs a permutation σ on the two lines if the control line is 1, can be simulated by a single controlled instruction σ^{x_i}.

The only gate that is attached to two read-only lines is a Toffoli gate that is controlled by two read-only lines and negates one of the two read-write lines, i.e., $\bigoplus_i^{j \wedge k}$, where $j,k < n+1$ and $i = n+1, n+2$. If both control lines j and k have value 1, then the Toffoli gate performs either permutation $(01)(23)$, if $i = n+1$, or permutation $(02)(13)$, if $i = n+2$, on the two read-write lines. The former Toffoli gate can be simulated via four instructions that form a commutator

$$[(012)^{x_j}, (132)^{x_k}] = (012)^{x_j}(132)^{x_k}(021)^{x_j}(123)^{x_k} = ((01)(23))^{x_j \wedge x_k},$$

and a similar set of four instructions simulate the latter Toffoli gate. Observe, that if either x_j or x_k are 0, the commutator yields the identity and if both are 1, then the commutator yields $(01)(23)$. Thus, every gate can be simulated by either one or four instructions, completing the proof. ■

A corollary of Theorem 5.3 is that any circuit C of bandwidth b can be simulated by a 2^b-PBP that is at most four times the length of C.

Corollary 5.4 *If C is a bandwidth-b circuit comprising n read-only lines and m gates, then there exists a 2^b-PBP of length $4m$ that simulates C.*

Consequently, all characterizations of 4-PBPs are immediately applicable to bandwidth-2 reversible circuits. Additionally, an immediate characterization of all bounded bandwidth reversible circuits, where $b > 2$, is possible, namely:

Corollary 5.5 *The class of problems decidable by bandwidth-b reversible circuits, where $b > 2$, is exactly* $\mathbf{NC}^1$ *[Bar89].*

Proof: This is an immediate consequence of Corollaries 5.2 and 5.4, and Barrington's characterization of w-PBP, $w > 5$. ■

5.2.2 On the Power of Bandwidth-2 Reversible Circuits and 4-PBPs

The power of 4-PBPs has long been a major open issue. It is well known that 3-PBP must be of length exponential in n to compute the conjunction of n variables [Bar85], while a 5-PBP requires a polynomial number of instructions to compute the conjunction on n variables [Bar89]. Thus, a bandwidth-3 reversible circuit can compute the conjunction of n lines using only a polynomial number of gates. In fact, bounded bandwidth reversible circuits of bandwidth greater than two, have the computing power corresponding to the complexity class $\mathbf{NC}^1$. In the case of bandwidth-2 circuits, the question of whether polynomial length circuit can compute the conjunction of n lines remains open. We further study this problem, by investigating programs over groups.

Programs over Subgroups of S_4

Our first observation is that programs over the subgroups of S_4, i.e., 4-PBPs whose instructions may only yield elements of a subgroup of S_4, are inherently not very powerful. There are few interesting subgroups of S_4; interesting in the sense of programs over the respective subgroup. Apart from the groups $S_3 = D_3$, D_4, and A_4, all remaining subgroups of S_4 are isomorphic to direct products of cyclic groups, or the groups D_3 and D_4 [Bur11]. Since we are primarily interested in strongly accepting programs, programs over direct products have the same computational power as programs over one of the components of the direct product. Thus, we need only consider the three groups listed above.

Barrington [Bar85] showed that programs over S_3, which is also equal to D_3, the dihedral group of degree 3, can compute the conjunction of n variables, albeit using programs of exponential length. Barrington et al. [BST90] also proved that programs over nilpotent groups are even weaker, they cannot compute the conjunction of n variables for sufficiently large n. Since the dihedral group of degree 4, D_4, is nilpotent, no program over D_4 can compute the conjunction of n variables either. We give a tight bound proving that no program over D_4 can compute the conjunction over three variables. The lower bound, a conjunction on two variables, is realized by the program $(13)^{x_1}((01)(23))^{x_2}(13)^{x_1}((01)(23))^{x_2}$.

Recall that an embedding D_4 in S_4 is generated by the elements $\{(0123),(13)\}$. Using a technique similar to the one in [Bar85] the following sequence of lemmas proves our claim. We first show that the computational power of programs over D_4, is equal to the computational power of 4-2-parity circuits and then argue that a 4-2-parity circuit cannot compute the conjunction function.

A 4-2-parity circuit is a two-level circuit comprising a single unbounded fan-in modulo-4 gate fed by unbounded fan-in parity gates, whose inputs are the variables. A 4-2-parity circuit C computes a Boolean function f if

$$C(x) = \begin{cases} \iota, & f(x) = 1 \\ 0, & f(x) = 0 \end{cases}.$$

for some fixed $\iota \in \{1,2,3\}$.

Lemma 5.6 *Let C be a 4-2-parity circuit that computes f on n variables and contains s parity gates. Then, there exists a program over D_4 of length $O(ns)$ that strongly computes f.*

Proof: Let n_i be the fan-in of the ith parity gate and let x_{j_k} be the kth input to the gate. The gate is simulated by sequence P_i of $2n_i$ instructions of the form

$$P_i = \sigma^{x_{j_1}}\sigma^{x_{j_2}}\ldots\sigma^{x_{j_{n_i}}}\tau^{x_{j_1}}\tau^{x_{j_2}}\ldots\tau^{x_{j_{n_i}}}$$

where $\sigma = (01)(23)$ and $\tau = (02)$ are in D_4. Since σ and τ are their own inverses, the yield of P_i is the identity if the parity of the inputs is even. Otherwise, if the parity of the inputs is odd, the yield is $\sigma\tau = (0123)$, which is an increment modulo-4.

Concatenating the instruction sequences of length $2n_i \leq 2n$ for each of the s parity gates yields a program of size at most $2ns$ that strongly computes f. ■

Lemma 5.7 *Let P be a program over D_4 on n variables and of length m that strongly computes function f. Then there is a 4-2-parity circuit C comprising $O(m)$ parity gates that computes f.*

Proof: For each instruction σ^{x_i} in P, rewrite σ as a product of the two generators: $\sigma = \tau^b\rho^c$, where $\tau = (13)$, $\rho = (0123)$, $b \in \mathbb{Z}_2$, and $c \in \mathbb{Z}_4$, and hence, $\sigma^{x_i} = (\tau^{x_i})^b(\rho^{x_i})^c$. For conciseness, the ith instruction is written as $\tau^{a_ib_i}\rho^{a_ic_i}$, where $a_i \in \{x_1, \ldots, x_n, 1\}$, $b_i \in \mathbb{Z}_2$, and $c_i \in \mathbb{Z}_4$. Thus, program P is a sequence of two types of instructions

$$P = \tau^{a_1b_1}\rho^{a_1c_1}\tau^{a_2b_2}\rho^{a_2c_2}\ldots\tau^{a_mb_m}\rho^{a_mc_m},$$

which we reorder such that all instructions of the form $\tau^{a_ib_i}$, $i = 1, \ldots, n$, precede all instructions of the form $\rho^{a_ic_i}$, $i = 1, \ldots, n$. Observe that if $c_i \equiv 0 \bmod 2$, then the instruction $\rho^{a_ic_i}$ commutes with all instructions of the form $\tau^{p(x)}$—this is a short form for the sequence $\tau^{x_1q_1}\tau^{x_2q_2}\ldots\tau^{x_nq_n}$, $q_i \in \mathbb{Z}_2$, where $p(x) = \oplus_{i=1}^{n} q_ix_i$i, $q_i \in \mathbb{Z}_2$, is a linear function on n variables. Thus, we need only worry about the case where $c_i \equiv 1 \bmod 2$.

If $c_i \equiv 1 \bmod 2$, via the identities $\rho\tau = \tau\rho^3$ and $\rho^3\tau = \tau\rho$, rewrite $\rho^{a_ic_i}\tau^{p(x)}$ as $\tau^{p(x)}(\rho^2)^{a_ip(x)}\rho^{a_ic_i}$. Starting from the end of the sequence and working towards the front, collect all $\tau^{a_ib_i}$ instructions and shuffle them towards the front of the sequence. The resulting sequence is of the form

$$P = \tau^{p(x)}\rho^{g_1(x)}\rho^{g_2(x)}\ldots\rho^{g_m(x)},$$

where $g_i(x)$ is a function of the form $a_i(2d_ip_i(x) + c_i)$, $p_i(x)$ is a parity function that corresponds to a partial collection of instructions yielding τ, and $d_i = c_i \bmod 2$. For example, the step $(m-k)$ of the shuffle looks like

$$\begin{aligned} P &= \tau^{a_1b_1}\rho^{a_1c_1}\tau^{a_2b_2}\rho^{a_2c_2}\ldots\underline{\tau^{a_kb_k}\rho^{a_kc_k}\tau^{p_{k+1}(x)}}\rho^{g_{k+1}(x)}\ldots\rho^{g_m(x)} \\ &= \tau^{a_1b_1}\rho^{a_1c_1}\tau^{a_2b_2}\rho^{a_2c_2}\ldots\underline{\tau^{a_kb_k}\tau^{p_{k+1}(x)}(\rho^2)^{a_kp_{k+1}(x)}\rho^{a_kc_k}}\rho^{g_{k+1}(x)}\ldots\rho^{g_m(x)} \\ &= \tau^{a_1b_1}\rho^{a_1c_1}\tau^{a_2b_2}\rho^{a_2c_2}\ldots\underline{\tau^{p_k(x)}\rho^{g_k(x)}}\rho^{g_{k+1}(x)}\ldots\rho^{g_m(x)}. \end{aligned}$$

Finally, since the modulo-4 gate always counts from zero, this corresponds to applying the yield of P to point 0. Since 0 is a fixed-point of permutation τ, we can drop the prefix $\tau^{p(x)}$ from the transformed program, ending up with the program

$$P' = \rho^{g_1(x)}\rho^{g_2(x)}\ldots\rho^{g_m(x)}.$$

Since ρ corresponds to an increment modulo-4, each instruction performs $g_i(x)$ increments modulo-4. Hence, we need only show how to compute $g(x)$ using parity gates. Since $g_i(x) = a_i(2d_ip_i(x) + c_i) = 2a_id_ip_i(x) + a_ic_i$, $\rho^{g_i(x)} = \rho^{2a_id_ip_i(x)}\rho^{a_ic_i}$, where c_i and $d_i = c_i \bmod 2$ are constants. If $d_i = 0$, the resulting product is $\rho^{a_ic_i}$ which is realized by c_i parity gates, with input a_i, feeding into the modulo-4 gate. Otherwise, if $d_i = 1$, the first part of instruction $\rho^{2a_ip_i(x)}\rho^{a_ic_i}$ is rewritten as

$$\rho^{2a_ip_i(x)} = \rho^1\rho^1\rho^{a_i\oplus p_i(x)}\rho^{a_i\oplus 1}\rho^{1\oplus p_i(x)},$$

which is realized using five parity gates; this concludes the proof. ■

An immediate corollary of the preceding lemmas is that programs over D_4 and 4-2-parity circuits are equivalent.

Corollary 5.8 *The function f is strongly computable by a program over D_4 if and only if the function is computable by a 4-2-parity circuit. Furthermore, the length of the program and the number of parity gates in the circuit are within factor of $O(n)$ of each other.*

To complete our claim we prove that 4-2-parity circuits cannot compute the conjunction of three variables, implying that neither can programs over D_4.

Lemma 5.9 *No 4-2-parity circuit can compute the conjunction of three variables.*

Proof: A 4-2-parity circuit on three variables consists of at most seven different parity gates: the power set of $\{x_0, x_1, x_2\}$ minus the constant parity gate—since conjunction is 0-preserving, constants cannot be used. We need not consider parity gates that perform negation since each such parity gate may be simulated by a constant parity gate followed by three copies of the unnegated gate, i.e., if a parity gate computes $1 \oplus p(x)$, this is the same as four parity gates: a constant 1 and three gates that compute $p(x)$. Each type of gate is used at most three times since all the parity gates feed into the mod-4 gate. Hence, to construct the circuit we need to solve the following set of linear equations modulo-4 where $a_{p(x)}$ denotes the number of times the parity gate that computes $p(x)$ is used and the right-hand side denotes the resulting value of the circuit—the circuit must yield a nonzero value if and only if the conjunction is true.

$$
\begin{array}{ccccccccc}
a_{x_0}+ & & & a_{x_0\oplus x_1}+ & a_{x_0\oplus x_2}+ & & a_{x_0\oplus x_1\oplus x_2} & = & 0 \bmod 4 \\
 & a_{x_1}+ & & a_{x_0\oplus x_1}+ & & a_{x_1\oplus x_2}+ & a_{x_0\oplus x_1\oplus x_2} & = & 0 \bmod 4 \\
 & & a_{x_2}+ & & a_{x_0\oplus x_2}+ & a_{x_1\oplus x_2}+ & a_{x_0\oplus x_1\oplus x_2} & = & 0 \bmod 4 \\
a_{x_0}+ & a_{x_1}+ & & & a_{x_0\oplus x_2}+ & a_{x_1\oplus x_2} & & = & 0 \bmod 4 \\
a_{x_0}+ & & a_{x_2}+ & a_{x_0\oplus x_1}+ & & a_{x_1\oplus x_2} & & = & 0 \bmod 4 \\
 & a_{x_1}+ & a_{x_2}+ & a_{x_0\oplus x_1}+ & a_{x_0\oplus x_2} & & & = & 0 \bmod 4 \\
a_{x_0}+ & a_{x_1}+ & a_{x_2}+ & & & & a_{x_0\oplus x_1\oplus x_2} & = & y \neq 0 \bmod 4
\end{array}
$$

Summing these equations together results in the unsatisfiable equation $0 = y \neq 0$. Hence, a 4-2-parity circuit cannot compute the conjunction of three or more variables. ■

This lemma, in conjunction with the preceding corollary, implies our claim.

Theorem 5.10 *No program over D_4 can strongly compute the conjunction of three variables.*

Proof: By Corollary 5.8 the computational powers of 4-2-parity circuits and of programs over D_4 are the same. By Lemma 5.9, 4-2-parity circuits cannot compute the conjunction of three or more variables. Hence, neither can programs over D_4. ■

The remaining proper subgroup of S_4 is the alternating group A_4. While programs over A_4 can compute the conjunction on n variables, via a construction that is similar to the one used in [Bar85], Barrington et al. [BST90] showed that programs over A_4 that strongly compute the conjunction of n

variables must be of exponential length. In [Bar89, BST90] it is conjectured that no program over a solvable group—all finite groups of order less than 60 are solvable [DH92]—can compute the conjunction on n variables. If the conjecture is true, then the alternating group A_5 is the smallest group over which polynomial length programs can compute the conjunction on n variables. This would imply that a reversible circuit must be of bandwidth-3 or greater in order to compute the conjunction over n variables. Since the conjecture remains open, there may yet be a clever construction that realizes a polynomial length program over S_4 that computes a conjunction. To get further insight into this problem we differentiate between programs based on modes of acceptance, i.e., strongly accepting, weakly accepting, or monotonically accepting. We study the computational power of each class of programs and the relationships between the classes.

The Computational Power of Weakly Accepting and Monotonically Accepting 4-PBPs

The acceptance mode of a permutation branching program significantly affects the computational power of the model. Since each mode of acceptance—weak, monotone, and strong—is subsumed by its predecessors, a natural complexity hierarchy is formed. We show that weakly accepting width-4 branching programs may be transformed into monotone accepting width-4 branching programs, and more importantly, that such programs cannot necessarily be transformed into strongly accepting programs. However, in the case of width-w permutation branching programs, $w > 4$, weak acceptance is equal to strong acceptance. Furthermore, transforming a weakly accepting program into a monotonically accepting or a strongly accepting program requires only a linear increase in size of the original program; the magnification factor only depends on w. Thus there is a natural gap between width-4 and width-5 permutations branching programs. These comparisons are particularly useful in the context of bounded bandwidth reversible circuits because it illustrates a natural trade-off between the complexity of the computation and the complexity of the post-processing phase that entails 'reading' the result from the outputs of the circuit. We first compare the computational power of programs that accept weakly versus programs that accept strongly. Namely, we prove that weakly accepting programs can be significantly shorter than their strongly accepting analogues. We first need the following definition and lemma. An ordered sequence of permutations T is said to be **faithful** if the product of any nonempty subsequence of T is not the identity.

Lemma 5.11 *For any integer $w > 1$, there exists a faithful sequence T_w of all $\binom{w}{2}$ distinct 2-cycles of S_w.*

Proof: The proof is by induction on w. In the base case where, $w = 2$, S_2 only has one 2-cycle. Thus, the only nonempty subsequence of T_2 is T_2, which contains one 2-cycle and whose product is not the identity; hence T_2 is faithful.

Assume that for some $w_0 > 1$ there exists a faithful sequence T_{w_0}. For $w = w_0 + 1$ we use T_{w_0} to construct T_w.

Each 2-cycle in S_w either belongs to T_{w_0}, or is of the form (iw) where $i < w$. Let U_w be any sequence comprising all 2-cycles of the form (iw). We claim that the concatenation $T_w = T_{w_0}U_w$ is faithful. Let τ be the product of any nonempty subsequence of T_{w_0} and υ be the product of any nonempty subsequence of U_w. First, υ can be represented by a single cycle $(i_1 i_2 ... i_k\ w)$, $i_j < w$,

i.e., the product will never be the identity. Second, by the inductive hypothesis, τ is not the identity. Finally, since w is not a fixed-point of υ and is a fixed-point of τ, we have $\tau \neq \upsilon^{-1}$, and thus the product of any nonempty subsequence of T_w is not the identity. ■

Corollary 5.12 *There is a linear time algorithm for constructing the sequence T_w.*

Using the sequence T_w as the basis of our construction, we show that for some languages, such as the disjunction on n variables, the weakly accepting program can be significantly shorter than a strongly accepting one.

Theorem 5.13 *Let $g(n)$ be the complexity of strongly computing the disjunction on n variables by a w-PBP, where $w > 4$. The complexity of weakly computing the disjunction of n variables by a w-PBP is at most $c \cdot g(\frac{n}{c})$ where $c = \binom{w}{2} - 1$.*

Proof: First, divide the n variables into groups of size $\frac{n}{c}$ and for each group construct a strongly accepting permutation branching program that computes their disjunction such that its complexity is $g(\frac{n}{c})$. Furthermore, each of these subprograms, denoted P_i, will yield a specific permutation, σ_i, if the disjunction is true, and ε otherwise.

Second, concatenate these subprograms $P = P_0 P_1 \dots P_c$. If we choose the permutations appropriately, program P will yield ε if and only if each of the subprograms, P_i, yields ε, i.e., all of $\frac{n}{c}$ disjunctions are false. It only remains to choose the permutation.

By the Lemma 5.11, for any symmetric group S_w there is a sequence of $\binom{w}{2}$ 2-cycles such that the product of a subsequence is equal to ε if and only if the subsequence is empty; let $T = \tau_1 \tau_2 \dots \tau_{\binom{w}{2}}$ be that sequence. If $\sigma_i = \tau_i \tau_{i+1}$, the sequence $\sigma_1 \sigma_2 \dots \sigma_c$ also has this property. Thus, P_i yields σ_i if and only if the disjunctions of its share of variables is satisfied.

Hence, program P yields ε if and only if the disjunction of the n variables is false. ■

Since the yield $P(x)$ of a strongly accepting program must either be ε or a fixed nonidentity permutation, if the function being computed is nonlinear, in all likelihood the yield is an even permutation. Thus, Theorem 5.13 assumes that each subprogram P_i yields an even permutation. In all cases but one, the cycle-structure of $P_i(x)$ is not an issue either. Using a single commutator $[P_i(x), \sigma^1]$, where σ^1 is an oblivious instruction, transforms the yield into a 3-cycle, and conjugating the 3-cycle with another appropriate oblivious instruction yields the required 3-cycle; Table 5.1 lists the six different forms of the yield $P_i(x)$. If the yield of P_i is a 3-cycle or 2-cycle, no transformation, apart from the conjugation, is necessary. Otherwise, each program P_i can be transformed into one that yields a 3-cycle and whose length is at most $2|P_i| + 4$.

Only yields of the form $(rs)(tu)$ of programs over S_4 cannot be transformed. However, one can construct programs over S_4—albeit of exponential length—that compute the disjunction and yield a 3-cycle length; in this case Theorem 5.13 is also applicable. In the case of programs over A_4 a weakly accepting program $P = P_1 P_2$, which computes the disjunctions can be constructed, where P_1 and P_2 compute the disjunction of half the variables each and yield $(01)(23)$ and $(02)(13)$ respectively. Thus, P yields ε if the disjunction is false, or $(01)(23)$, $(02)(13)$, or $(03)(12)$ if the disjunction is true. Furthermore, program P is at most half the size of a strongly accepting program

Form of $P(x)$	w	σ^1
$(r_1 r_2 \ldots r_n)\ldots, n > 3$	$w \geq n$	$(r_1 r_2 \ldots r_m)$, $m = 2\lfloor n/2 \rfloor - 1$
$(rst)(uvw)\ldots$	$w \geq 6$	$(rwtus)$
$(rst)(uv)\ldots$	$w \geq 5$	$(rs)(uv)$
(rst)	$w \geq 3$	N/A
$(rs)(tu)(vw)\ldots$	$w \geq 6$	(rtv)
$(rs)(tu)$	$w \geq 5$	(rsv), $v \neq r,s,t,u$
(rs)	$w \geq 3$	N/A

Table 5.1: Forms of $P(x)$ and the corresponding σ^1.

over A_4 that computes the disjunction. However, weakly accepting programs are unsatisfying in the following sense.

Suppose C simulates a weakly accepting program over S_4. Hence, the read-only lines correspond to the variables and the two read-write lines represent the yield of the program. Suppose that the read-write lines are initially 0. Since the program being simulated is weakly accepting, some of the permutations that are yielded, e.g., (123) and (132), have 0 as a fixed-point. Thus, for those inputs that yield such permutations, the output of the read-write lines of C remains unchanged—assuming the read-write lines were initially 0. Hence, determining the result of the computation by reading the output lines is not possible and requires that the computation be performed twice with different initial values on the read-write lines. This requirement seems to inherently violate the notion of what it means for a circuit to compute a Boolean function.

To solve this problem we use a slightly stronger notion of acceptance, namely, monotone acceptance. Recall that a program monotonically accepts x if it yields a permutation with a particular fixed fixed-point if and only if x is not in the language. Thus, the corresponding circuit first initializes the read-write lines to the fixed-point, simulates the program, and then XORs the result with the fixed-point. The output is the disjunction of the read-write lines, which will be 0 if and only if the input x to the circuit was not in the language. Fortunately, we can convert a weakly accepting program over S_4 into a monotonically accepting one.

Theorem 5.14 . *Let P be a weakly accepting program over S_4, such that for all x, $P(x) \in A_4$. Then the program $\Psi(P)$ is a monotonically accepting program over S_4, where*

$$\Psi = P^3 \left((01)P^2(02)P^4(03)\right)^2 (02)(13).$$

Proof: The characteristic map of Ψ on elements of A_4 is

$$\begin{aligned} \Psi(\varepsilon) &= \varepsilon \\ \Psi((023)) = \Psi((01)(23)) = \Psi((031)) &= (01)(23) \\ \Psi((123)) = \Psi((012)) = \Psi((02)(13)) = \Psi((132)) = \Psi((032)) &= (02)(13) \\ \Psi((021)) = \Psi((03)(12)) = \Psi((013)) &= (03)(12). \end{aligned}$$

Since (01)(23), (02)(13), and (03)(12), have no fixed-points, the result follows. ■

Since transformation Ψ only works on programs whose yields are in A_4, we use the following theorem to transform programs that yield odd permutations into programs that only yield even permutations.

Theorem 5.15 *The characteristic map of transformation*

$$\Psi = [(0123), P](P(23))^6 P^6,$$

is such that $\Psi(S_4) \subseteq A_4$ *and* $\Psi(\sigma) = \varepsilon$ *if and only if* $\sigma = \varepsilon$.

Proof: By enumeration of the characteristic map on all 24 elements of S_4. ■

Composing these theorems yields the desired theorem.

Theorem 5.16 *There exists a transformation* Ψ *such that for any weakly accepting program over* S_4, $\Psi(P)$ *is a monotonically accepting program for the same language over* S_4.

In fact, the transformation increases the length of P by a factor of at most $14 \times 15 + 1 = 211$. This implies that the computational power of programs over S_4 that monotonically accept is the same as the power of weakly accepting programs over S_4. We can even say something stronger, namely that monotonically accepting programs over A_4 have the same power as weakly accepting programs over A_4.

Corollary 5.17 *If* P *is program over* A_4 *that weakly computes* f, *there exists a program over* A_4 *that monotonically computes* f *and is of length* $15|P| + O(1)$.

Proof: First apply Theorem 5.14 to P. Since the parity of the constant permutations in the transformation is even, we can collect the permutations at one end of the program via the standard conjugation technique used in Theorem 4.17. Thus, the resulting program will be over A_4. ■

The equivalence between weakly accepting and monotonically accepting width-4 programs begs the comparison between strongly accepting and monotonically accepting width-4 programs. The comparison yields a natural characterization of the difference in computational power of 4-PBPs versus 5-PBPs. Namely, we would like an analog of Theorem 5.16, that is, a general transformation Ψ that transforms any weakly accepting program into a strongly accepting one. Amazingly, such a transformation cannot exist for programs of width 4, but does exist for programs of greater width! This is a natural distinction between the computational powers of 4- PBPs and 5-PBPs!

Theorem 5.18 *There does not exist a transformation* Ψ *such that for any program* P *over* S_4, $\Psi(P)$ *is a strongly accepting program computing the same Boolean function as* P.

Proof: We shall consider weakly accepting programs over S_4 whose yields are from the Klein group

$$K = \{\varepsilon, (01)(23), (02)(13), (03)(12)\} \subset S_4.$$

By contradiction, assume that there exists a transformation Ψ that converts weakly accepting programs into strongly accepting ones. Let $\sigma_1 = (12)(34)$, let $\sigma_2 = (13)(24)$, and let $Q = \sigma_1^{x_1}\sigma_2^{x_2}$ be

a program. Now, consider the structure of $\Psi(Q)$. $\Psi(Q)$ is a word comprising constant instructions of the form τ^1 and nonconstant instructions: either $\sigma_1^{x_1}$ or $\sigma_2^{x_2}$. We can rewrite the program, without changing the function, i.e., $\sigma^{x_i}\tau^1 = \tau^1(\tau^{-1}\sigma\tau)^{x_i}$. Furthermore, since conjugation preserves structure, the structure of each instruction is preserved.

First, since $\Psi(Q)$ must yield ε if Q yields ε, all the constant instructions cancel each other out. Second, since all instructions of Q, and hence of $\Psi(Q)$, belong to the Klein group, the instructions commute. Hence, $\Psi(Q)$ can be rewritten as $\Psi(Q) = \tau_1^{x_1}\tau_2^{x_2}$, where $\tau_1, \tau_2 \in K$. Unless $\tau_1 = \tau_2$, $\Psi(Q)$ is not strongly accepting, and if equality holds then $\Psi(Q)$ yields ε on assignment $x_1 = x_2 = 1$, contrary to what Q yields. Hence, Ψ cannot exist. ■

In fact, a similar argument can be extended to programs over A_4. If a transformation did exist, then a polynomial length program computing the conjunction of n variables would be possible. Namely, we use Barrington's divide and conquer construction [Bar89]. If P and P' each strongly compute the conjunction on 2^k disjoint variables, where the positive yields of P and P' are distinct, then PP' weakly compute the conjunction of 2^{k+1} variables, and $\Psi(PP')$ strongly computes the conjunction of 2^{k+1} variables. Repeating this composition $\log n$ times yields polynomial length program, and this is impossible according to [BST90]. However, a general transformation does exist on programs of width greater than four.

Weak Acceptance Equals Strong Acceptance for w-PBPs, $w > 4$

If $w > 4$ then a weakly accepting w-PBP can be transformed into a strongly accepting program with only a linear increase in size. To outline the construction, the goal is to construct a transformation Ψ_π that is characterized by the map

$$\Psi_\pi(\sigma) = \begin{cases} \rho, & \sigma = \pi \\ \varepsilon, & \sigma \neq \pi \end{cases},$$

for some $\rho \neq \varepsilon$ and $\pi, \rho, \sigma \in S_w$. Hence, given program P, if $Q = \Psi_\pi(P)$, then Q yields ρ on input x if and only if P yields π on input x, and yields ε otherwise. In fact, we need only exhibit the construction for a fixed π_0, say $\pi_0 = (01 \ldots w-1)$ since

$$\Psi_\pi(P) = \Psi_{\pi_0}(P\pi^{-1}\pi_0).$$

Concatenating $|S_w| - 1$ copies of $\Psi_\pi(P)$ for each $\pi \in S_w \setminus \varepsilon$, yields the desired transformation.

To implement Ψ_π, the transformation verifies the point order of permutation π_0 by comparing the adjacent points within the permutation. The comparison is performed by a simpler transformation Υ_τ on transpositions, where $\Upsilon_\tau(\upsilon) = \varepsilon$ if and only if $\upsilon \neq \tau$. Computing the conjunction of these comparisons via the standard commutator transformation of [Bar89] completes the construction. We begin with the construction for Υ_τ.

Lemma 5.19 *Let $w > 4$ and let $\tau \in S_w$ be a 2-cycle. There exists a transformation Υ_τ such that $\Upsilon_\tau(\varepsilon) = \varepsilon$ and for every 2-cycle $\upsilon \in S_w$, $\Upsilon_\tau(\upsilon) \neq \varepsilon$ if and only if $\upsilon = \tau$. Furthermore, the permutation $\Upsilon_\tau(\tau)$ can be represented by two disjoint 2-cycles.*

Proof: Let $\tau = (rt) \in S_w$ and let (su) be a 2-cycle such that $[(rt),(su)] = \varepsilon$. The straight-line composition of Υ_τ is:

$$\begin{aligned}
\Upsilon_0 &= [(rs),\upsilon] \\
\Upsilon_1 &= [(rus),\Upsilon_0] \\
\Upsilon_2 &= \Upsilon_1(su) \\
\Upsilon_3 &= (\Upsilon_2)^2 \\
\Upsilon_4 &= \Upsilon_3(tu) \\
\Upsilon_{(rt)} &= (\Upsilon_4)^6
\end{aligned}$$

If υ commutes with (rs)—υ is either a disjoint 2-cycle or the identity—then $\Upsilon_0(\upsilon) = \varepsilon$, implying that $\Upsilon_1(\upsilon) = \Upsilon_3(\upsilon) = \Upsilon_{(rt)}(\upsilon) = \varepsilon$. Otherwise, there are two cases: either $\upsilon = (rv)$, $v \neq s$ or $\upsilon = (vs)$, $v \neq r$.

If $\upsilon = (rv)$, then $\Upsilon_0((rv)) = (rvs)$. If $v = u \neq t$ then $\Upsilon_1((rv)) = \varepsilon$, and so does $\Upsilon_{(rt)}((rv))$. Otherwise, $\Upsilon_1((rv)) = (rs)(uv)$, $\Upsilon_2((rv)) = (ruvs)$, and $\Upsilon_3((rv)) = (rv)(su)$. If $v \neq t$, then $\Upsilon_4((rv)) = (rv)(stu)$ and hence $\Upsilon_{(rt)}(rv) = \varepsilon$. On the other hand, if $v = t$, then $\Upsilon_4((rt)) = (rust)$ and hence, $\Upsilon_{(rt)}(rt) = (rs)(tu)$.

If $\upsilon = (vs)$, then $\Upsilon_0((vs)) = (rsv)$. If $v = u \neq t$. then $\Upsilon_1((vs)) = \varepsilon$, and so does $\Upsilon_{(rt)}((vs))$. Otherwise, $\Upsilon_1((vs)) = (rv)(us)$, $\Upsilon_2((vs)) = (rv)$, and $\Upsilon_3((vs)) = \varepsilon$, implying that $\Upsilon_{(rt)}(vs) = \varepsilon$.

Hence, $\Upsilon_{(rt)}(\upsilon) \neq \varepsilon$ if and only if $\upsilon = (rt)$, and $\Upsilon_{(rt)}((rt)) = (rs)(tu)$. ■

To construct the main transformation, Ψ_π, we need to formalize the behaviour of the conjugate operation.

Lemma 5.20 (Selection Lemma) *Let σ be a cycle of the form $(rs\ldots tu\ldots)$ where the ellipses $(\ldots)$ represent zero or more additional points within the cycle. Then $\sigma(su)\sigma^{-1} = (rt)$.*

Proof: We can rewrite σ as:

$$\begin{aligned}
\sigma &= (rs\ldots tu\ldots) \\
&= (rs\ldots t)(ru\ldots) \\
&= (trs\ldots)(ru\ldots) \\
&= (trs)(t\ldots)(ru)(r\ldots) \\
&= (trs)(ru)(t\ldots)(r\ldots) \\
&= (turs)(t\ldots)(r\ldots).
\end{aligned}$$

Setting $\rho = (t\ldots)(r\ldots)$, and rewriting the conjugate

$$\begin{aligned}
\sigma(su)\sigma^{-1} &= (turs)\rho(su)\rho^{-1}(tsru) \\
&= (turs)(su)(tsru) \\
&= (rt),
\end{aligned}$$

yields the result. ■

Corollary 5.21 *If σ is a permutation that is represented by the cycle $(rst\ldots)$, then $\sigma(st)\sigma^{-1} = (rs)$.*

To check the adjacent order of the points in a permutation, we use corollary 5.21 in conjunction with lemma 5.19. To verify that the permutation is $(12\ldots w)$, w checks must be performed, to ensure that 1 follows w, 2 follows 1, and so on. Each of these checks is performed by a conjugation transformation composed with Υ_τ transformation. The conjunction of these w checks is then computed. This formalized in the following theorem.

Theorem 5.22 *Let $w > 4$ and let $\pi_0 = (01\ldots w-1) \in S_w$. There exists a width-$w$ program transformation Ψ_{π_0} that is characterized by the map that sends every element except π_0 to ε and sends π_0 to some nonidentity element.*

Proof: Let $\Xi_r = P(st)P^{-1}$, $r = 0\ldots w-1$, be the conjugation transformation where $s = (r+1 \bmod w)$ and $t = (s+1 \bmod w)$, i.e., rst corresponds to a triple of points in permutation π_0.

Without loss of generality, assume that the characteristic map of $\Upsilon_{(rs)}$ maps (rs) to $(rs)(tu)$, where $u = (t+1 \bmod w)$. Compose the Ξ_r transformation with $\Upsilon_{(rs)}$ to construct the adjacency checking transformation

$$\Phi_r(P) = \phi_r \Upsilon_{(rs)}(\Xi_r(P))\phi_r^{-1},$$

where $\phi_r = (0r)(01)(0s)(03)(0t)(04)(0u)$. Conjugating by ϕ_r is characterized by a map that sends ε to ε and sends $(rs)(tu)$ to $(01)(34)$.

Finally, Ψ_{π_0} is a combination of the transformations Φ_r, $r = 1\ldots w-1$. Define the recursive commutator of an ordered set of permutations to be

$$[\{\rho_1, \rho_2, \ldots, \rho_k\}] = [\rho_1, [\{\rho_2, \ldots, \rho_k\}]],$$

where $[\{\rho_{n-1}, \rho_n\}] = [\rho_{n-1}, \rho_n]$. Since $[(01)(34), (012)] = (012)$, the construction of Ψ_{π_0} is the recursive commutator of the set $\{\Phi_1, \Phi_2, \ldots, \Phi_n, (012)\}$,

$$\Psi_{\pi_0} = [\{\Phi_1(P), \Phi_2(p), \ldots, \Phi_n(P), (012)\}].$$

The characteristic map of Ψ_{π_0} sends π_0 to (012) and sends all other permutations to ε. If the permutation is not π_0 then the characteristic maps of at least one Φ_i, $i = 1\ldots w-1$ will send the permutation to the identity—the recursive commutator of the yields of the Φ_is yields the identity. On the other hand, if all Φ_is yield $(01)(34)$, the yield of the recursive commutator is (012). Finally, note that the size of the transformation Φ_{π_0} only depends on S_w. Hence, the transformation increases program size by a constant factor. ■

Corollary 5.23 *For all $w > 4$ the width-w program transformation $\Psi_\pi = \Psi_{\pi_0}(P\pi^{-1}\pi_0)$ is characterized by the map that sends every element except π to ε and sends π to (012).*

Using the preceding corollary we can now construct the transformation Ψ that transforms any weakly accepting program into a strongly accepting one.

Theorem 5.24 *If P is a weakly accepting program over S_w, $w > 4$, which computes a Boolean function f, then $\Psi(P)$ strongly computes Boolean function f, where transformation Ψ is*

$$\Psi = \prod_{\pi \in S_w \setminus \varepsilon} \Psi_\pi(P).$$

Proof: Each of the Ψ_π transformations transforms P into a program that yields (012) if and only if P yields π. Hence, if P yields a nonidentity, exactly one of the transformed programs will yield (012) and the rest will yield ε. Thus, for any input x, $\Psi(P)$ yields (012) if and only if P does not yield ε; $\Psi(P)$ yields ε otherwise. The length of $\Psi(P)$ is $O(|P|)$ and depends only on w. ■

Thus, the computational power of weakly accepting and strongly accepting programs of width-5 or greater is equivalent. The same is true for reversible circuit of bandwidth-3 or greater. Thus, there is a natural distinction between the power of 4-PBPs (bandwidth-2 reversible circuits) and (5+)-PBPs (bandwidth-(3+) reversible circuits)! Finally, by Corollary 5.5 the class of problems decidable by bounded bandwidth reversible circuits of bandwidth greater than two is $\mathbf{NC}^1$. We now shift our attention to polylogarithmic bandwidth reversible circuits.

5.3 Polylogarithmic Bandwidth Reversible Circuits

Polylogarithmic bandwidth (polylog-bandwidth) reversible circuits comprise $b \in O((\log n)^i)$ read-write lines and $r = n - b$ read-only lines. Since such circuits of unlimited length can trivially compute any Boolean function in $\mathbf{P}$, we focus on polynomial size polylog-bandwidth circuits.

Define the class $\mathbf{RC}^i$ to be the class of problems solvable by polynomial size reversible circuits of bandwidth $O((\log n)^i)$. Note that the class of problems decidable by bounded bandwidth polynomial size reversible circuits, which is also $\mathbf{NC}^1$, is the class $\mathbf{RC}^0$, i.e., $\mathbf{NC}^1 = \mathbf{RC}^0$. Define the class $\mathbf{RC} = \cup_{i>0}\mathbf{RC}^i$ to be the class of problems solvable by polynomial size polylog-bandwidth circuits. Since $\mathbf{RC}^i \subseteq \mathbf{RC}^{i+1}$, the complexity classes $\mathbf{RC}^i$ form a hierarchy, called the $\mathbf{RC}$ hierarchy.

There is a natural relationship between the $\mathbf{RC}$ hierarchy and the $\mathbf{SC}$ hierarchy, which comprises the classes $\mathbf{SC}^i$. Recall from Section 2.5.3 that $\mathbf{SC}^i$ is the class of problems that is decidable by a Turing machine that uses $O((\log n)^i)$ space and a polynomial amount of advice. In this section we prove the equivalence between $\mathbf{RC}$ and $\mathbf{SC}$ by proving two things: first, we show that $\mathbf{RC}^i \subseteq \mathbf{SC}^i$, and second, we show that $\mathbf{SC}^i \subseteq \mathbf{RC}^{2i}$.

Theorem 5.25 *For all $i > 0$, $\mathbf{RC}^i \subseteq \mathbf{SC}^i$.*

Proof: Let C be a polynomial size reversible circuit on n inputs that is of bandwidth $O(\log(n)^i)$. We construct a Turing machine T that simulates circuit C and uses a work tape of length $O(\log(n)^i)$, an input tape, and an advice tape. The input tape contains the input comprising the n input variables to C and the advice tape contains the circuit, encoded in $O(|C|\log(n))$ space as a sequence of 4-tuples (g, c_1, c_2, o), where each 4-tuple comprises four integers, the gate type, the control lines, and the output line. The work tape stores the state of the $O(\log(n)^i)$ read-write lines of C, and a counter of $O(\log n)$ bits to keep track of the input tape head and locate the read-write lines on the work tape.

Machine T simulates circuit C one gate at a time and stores the intermediate result on the work tape. The machine stores the gate type in its finite state, and uses the counter to locate the control and output lines on the work tape. The output is computed within the finite state and written over the previous state of the output line. The final output of T comprises the final state of all the read-write lines. Since C is polynomial in size the computation of T takes $O(|C|n\log^2 n)$ steps and uses $O(\log(n)^i)$ space. Thus, simulating C is in $\mathbf{SC}^i$, and hence $\mathbf{RC}^i \subseteq \mathbf{SC}^i$. ■

The converse is not necessarily true since the class $\mathbf{SC}^i$ is defined within the framework of general computation whereas the class $\mathbf{RC}^i$ requires that the problems within it be decided reversibly. However, via Bennett's [Ben89], all problems in $\mathbf{SC}^i$ are decidable by polynomial size reversible circuits of bandwidth $O((\log n)^{2i})$.

Theorem 5.26 *For all $i > 0$, $\mathbf{SC}^i \subseteq \mathbf{RC}^{2i}$.*

Proof: Let T be any Turing machine that only uses $O((\log n)^i)$ work space and a polynomial amount of advice, where n is the size of the input. Since $i > 0$, T may use at least $O(\log n)$ space, we can use Bennett's construction to construct a reversible Turing machine R that uses quadratic space, $O((\log n)^{2i})$, and polynomial time to simulate T [Ben89]. Fixing n, and hence the advice to R, we argue that R can be simulated by a polynomial size reversible circuit C of bandwidth $O((\log n)^{2i})$.

Let c be a constant that depends only the size of the input alphabet, work tape alphabet, finite state, and the degree of the polynomial $p(n)$, where $p(n)$ bounds the length of the advice tape. Circuit C comprises of several parts:

input-tape lines: n read-only lines corresponding to the input,
advice-head lines: at most $c\log n$ read-write lines corresponding to the advice tape (counter),
input-head lines: at most $c\log n$ read-write lines corresponding to the input tape head (counter),
work-head lines: at most $c\log n$ read-write lines corresponding to the work tape head (counter),
finite-state lines: at most $c\log n$ read-write lines, corresponding to the finite state control of R, and
work-tape lines: at most $c(\log n)^{2i}$ read-write lines corresponding to the work tape, where each cell of the tape is simulated by $O(1)$ read-write lines.

Using a simplification akin to the one in [Ben73], we assume that the instructions of R can either

- obliviously move the work tape head,
- read/write a cell of the work tape,
- obliviously move the input tape head,
- read a cell of the input tape,
- obliviously move the advice tape head,
- read a cell of the advice tape, or
- do nothing.

The three oblivious head movements correspond to an increment or decrement of the advice-head, input-head, and work-head counters, a permutation that can be realized in a polynomial number of

gates since the counters are at most $c \log n$ lines wide. The increments and decrements are controlled by the finite-state lines.

Since the advice tape can be thought of as a truth table of a Boolean function, say α, on at most $c \log n$ variables, which can be realized by a polynomial size controlled circuit on the advice-head lines. The circuit is controlled by the finite-state lines and the operation comprises three steps: perform α on the advice-head lines, save the output in the finite-state lines, and perform α^{-1} on the advice-head lines.

Similarly, reading the input corresponds to a controlled copy of a single input-tape line into the finite-state lines. The copy is controlled by the input-head lines and the finite-state lines. The reading and writing of the work tape is similarly performed, namely, a copy operation is performed to or from the finite-state lines and is controlled by the work-head lines and the finite-state lines. While the work-head lines are unnecessary because the work-head lines can be embedded into the work-tape lines, their explicit use simplifies the construction.

Finally, we only need to note the finite-state lines. A polynomial size circuit on a constant number of read-write lines is sufficient to realize a permutation that corresponds to the transitions of the finite state. There is only one issue. Since the Turing machine is reversible, we're guaranteed that the transition function is a permutation. However, unlike a Turing machine, a circuit cannot cease computing once the halting state is reached. A circuit must perform all m computational steps, where $m = \max_{x \in B_n} \text{TIME}(R,x)$. Since the transition function must be a permutation, the circuit simulates 'halting early' via an additional $O(m)$ halting states that are iterated through for $m - \text{TIME}(R,x)$ steps. The additional halting states perform no operation and serve only as place holders in the transition function. Since R takes polynomial time in n to complete, at most $c \log n$ lines suffice to represent the finite state. If we were guaranteed that all computation paths of R on inputs of size n were the same length, then we could make do with $O(1)$ finite-state lines.

Thus, a polynomial size bandwidth-$O((\log n)^{2i})$ reversible circuit can simulate a Turing machine T, which uses $O((\log n)^i)$ space and polynomial time. Thus, $\mathbf{SC}^i \subseteq \mathbf{RC}^{2i}$. ■

An immediate corollary of the preceding two theorems is that polylog-bandwidth circuits can decide exactly the problems contained in class **SC**.

Corollary 5.27 $\mathbf{RC} = \mathbf{SC}$.

Theorem 5.26 can easily be generalized to Turing machines that use polynomial space; the corresponding circuits essentially have unbounded bandwidth.

5.4 Unbounded Bandwidth Reversible Circuits

Unbounded bandwidth reversible circuits—circuits comprising only read-write lines—are more general than their bounded bandwidth analogues. Consequently, characterizing their computational power is much more challenging. By the same argument as the one used in Theorem 5.26, polynomial size unbounded bandwidth circuits can be used to decide exactly the same set of problems as Turing machines that use polynomial space and time, i.e., the problems of class **P**.

In many physical contexts, such as quantum computation, using garbage lines—in addition to the n lines containing the input—is expensive. Since the number of garbage lines must be kept to a minimum, we focus on circuits that realize functions on n variables and comprise $n+c$ read-write lines, where c is 0, 1, or 2, and the c garbage lines are initialized to 0.

To compute a Boolean function f, function f is first embedded into a permutation on either B_n, if f is balanced, or B_{n+1} otherwise [Tof80]. Usually the exact embedding does not matter as long as the resulting permutation σ induces a partition of B_n (or B_{n+1}) such that for some fixed $i \in [1,n]$, for all $x \in B_n$, $(x\sigma)_i = f(x)$. If f is Boolean function of the form $f : B_n \rightarrow B_k$, $1 < k \leq n$, a similar criterion can be derived. Unfortunately, many such functions require more than a constant number of garbage lines [Tof80]. A prime example of this is the multiplication function, that takes two n-bit strings and yields a $2n$ bit string. Since there are 2^{n+1} inputs that map to the same output, i.e., multiplication by zero, multiplication can only be embedded into a permutation on Boolean cube of order $4n$ or larger.

We limit our attention to Boolean functions that are either of the form $f : B_n \rightarrow B_1$, or permutations of the form $f : B_n \rightarrow B_n$. In either case f can be embedded into an even permutation on B_n or B_{n+1}. We first investigate how to realize permutations corresponding to some common functions; in particular we provide an interesting recursive construction for realizing threshold functions. Following this, we show that if a permutation has a polynomial size cycle representation, then the permutation can be realized by a polynomial size circuit (Theorem 5.34). Based on the observations that we derived from our constructions, we conclude this chapter by describing some heuristics for realizing Boolean functions.

5.4.1 Reversible Circuit Constructions

We first investigate how to realize common permutations such as incrementors (decrementors) and adders. After deriving realizations for these components, we use them to realize Boolean functions such as the consensus function and various threshold functions. In most cases the constructions are relatively efficient, requiring $O(n^2)$ gates. The exception is the majority function, which does not seem to have an efficient realization.

For conciseness, we use several schematic short forms. First the k-line controlled Toffoli gate, $k < n-1$, which computes the conjunction of k lines and XORs the output line. A k-line controlled Toffoli (k-Toffoli) gate can be constructed using $O(k)$ Toffoli gates [BBD+95] and is illustrated in Figure 5.1a. Second, the controlled k-NOT, comprises k controlled-NOT gates that are all controlled by the same line. In most cases, we will be using the controlled $(n-1)$-NOT, which is illustrated in Figure 5.1c. Additionally, we use blocks to denote a component of a circuit. A component may either be simple, controlled by another line, as in Theorem 4.14, or the component may control another line, i.e., compute a Boolean function on k lines and XOR another line with the results. The block components are illustrated in Figures 5.1d, 5.1b, and 5.1e. The controlled k-NOT and the k-Toffoli gate are examples of a controlled component and a component control.

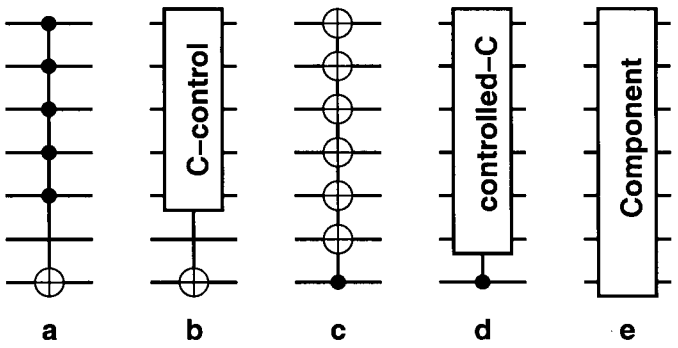


Figure 5.1: a) A k-Toffoli gate; b) a component control; c) a controlled n-NOT; d) a controlled component; e) a simple component.

Realization of Various Incrementors

We begin by constructing a half-incrementor, a permutation function on B_n that is represented by two disjoint cycles of the form

$$\pi = (0\ 1\ \ldots\ 2^{n-1}-1)(2^{n-1}\ 2^{n-1}+1\ \ldots\ 2^n-1).$$

Realizing a full incrementor on B_n is impossible because the permutation $(0\ 1\ \ldots\ 2^n-1)$ is odd, and hence cannot be realized using regular Toffoli and NOT gates [CG75]. However, a **nigh-incrementor**, corresponding to the permutation $(0\ 1\ \ldots\ 2^{n-1}-2)$, can be realized.

The half-incrementor can be realized via a sequence of k-Toffoli gates, where $k = n-2\ldots 0$. Observe that an incrementor modifies the ith least significant bit of the input if and only if the conjunction of the $i-1$ least significant bits of the input is equal to 1. Thus, the circuit comprises $n-1$ components (k-Toffoli gates), where the jth gate is an $(n-2-j)$-Toffoli gate that negates line $n-1-j$ and is controlled by the lines i, $i = 1\ldots n-2-j$; the construction is illustrated in Figure 5.2a.

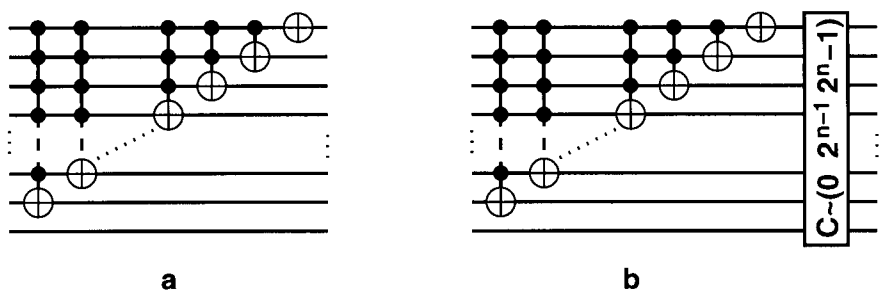


Figure 5.2: Realizations of a) a half-incrementor and b) a nigh-incrementor.

The last line in the circuit in Figure 5.2a may seem superfluous, but it is required in order to realize the $(n-2)$-Toffoli gate. The line is used as a temporary register and is returned to its original

value by the end of the computation of the $(n-2)$-Toffoli gate. By straightforward induction on n, it is easy to see that the circuit realizes permutation π. Since the realization of each k-Toffoli gate requires $O(k)$ normal Toffoli gates (2-Toffoli gates), the half incrementor may be realized in $O(n^2)$ gates. It follows that if we use an additional garbage line, then a complete incrementor can be realized, otherwise, the best we can hope to realize is a nigh-adder. Not surprisingly, the half-incrementor forms the basis of the realization of a nigh-incrementor.

The nigh-incrementor may be realized by concatenating an additional circuit onto the one that realizes a half-incrementor. Begin by noting that the nigh-incrementor corresponds to the permutation $\tau = (0\ 1\ \ldots\ 2^{n-1}-2)$. Let $\rho = \pi^{-1}\tau = (0\ 2^{n-1}\ 2^n-1)$. Thus, the circuit in Figure 5.2b, which realizes the nigh-incrementor, is a half-incrementor concatenated with a circuit that realizes permutation ρ. By Corollary 5.32, which we prove in a later subsection, any permutation that is represented by a 3-cycle can be realized by a circuit of size $O(n)$. Thus, the nigh-incrementor can be realized in $O(n^2)$ gates as well. The half-incrementor is also a primary component in the construction of the adder.

A Realization of the Adder

We consider an adder that takes two n-bit inputs, on $2n$ lines and outputs the result on the latter n lines, $n+1,\ldots,2n$, and the first summand on the former n lines, $1,\ldots,n$. The adder comprises a sequence of n controlled half-incrementors; see Figure 5.3. The kth half-incrementor is controlled by line k, $k \in [1,n]$, and increments the $n-k$ most significant lines of the second summand, i.e., the increment is performed on lines $n+k,\ldots,2n$. This follows from the observation, that adding 2^j to an n-bit value corresponds to performing an increment on the $n-j$ most significant bits. The adder does exactly that, performing a controlled increment for each of the n bits of the first summand.

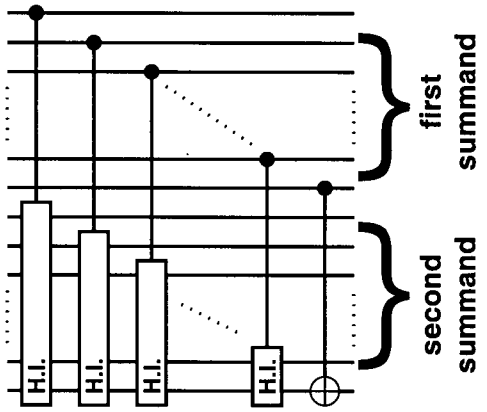


Figure 5.3: A realization of the adder.

Since each half-incrementor can be realized in $O(n^2)$ gates, the entire adder can be realized in $O(n^3)$ gates. Unfortunately, there seems little that can be done to reduce this bound. While one would think that a ripple adder could be implemented, each stage of the ripple adder loses

information—the preceding carry—implying that in order for a ripple adder circuit to work reversibly, all carry information needs to saved. Currently, we know of no way to accomplish this. The current realization trades space for time, which is the n stage adder just described.

In contrast to the incrementor, to realize an adder requires no additional lines, even though both functions are permutations. The difference being that the adder is an even parity permutation and the incrementor is not. The other sufficient condition for requiring the additional garbage line occurs when the function being realized is not a permutation, such as unbalanced Boolean functions of the form $f : B_n \rightarrow B_1$. Two families of such functions, the consensus and the threshold functions are studied next.

A Realization of the Consensus Function

The consensus function on n inputs evaluates to 0 for all but two inputs, **1** and **0**, i.e,

$$f(x) = \overline{\bigvee_{i=1}^{n} x_i} \vee \bigwedge_{i=1}^{n} x_i.$$

Since the function is unbalanced an additional line is required to reversibly realize the function. Without loss of generality, assume that the additional line, which is labeled 0 and initialized to zero, is also the output line. The consensus function corresponds to the permutation

$$\pi = (0\ \ 1)(2^{n+1} - 2\ \ 2^{n+1} - 1),$$

the first 2-cycle toggles line 0 if all lines have value 0, and the second 2-cycle toggles line 0 if all lines have value 1. The $(n-1)$-Toffoli gate that is controlled by the lines $2, \ldots, n$ and toggles line 0, realizes the permutation

$$\upsilon = (2^{n+1} - 4\ \ 2^{n+1} - 3)(2^{n+1} - 2\ \ 2^{n+1} - 1)$$

and differs from the required permutation only in the first 2-cycle. Thus, this gate forms the basis of the circuit.

To repair the difference, we use the following observation. For any 2-cycle, $\tau = (st)$, conjugating τ by $\sigma = (rsu)$ and then by $\rho = (qtu)$ yields yields the 2-cycle

$$\rho\sigma\tau\sigma^{-1}\rho^{-1} = (qtu)(rsu)(st)(rus)(qut) = (qtu)(rt)(qut) = (qr).$$

Setting $q = 0$, $r = 1$, $s = 2^{n+1} - 4$, $t = 2^{n+1} - 3$, and u to any other value but q, r, s, t, $2^{n+1} - 2$, and $2^{n+1} - 1$ specifies two 3-cycles, such that conjugating υ by them, yields permutation π. The consensus function can therefore be realized in the corresponding way, as is illustrated in Figure 5.4.

By Corollary 5.32, the circuits that realize the 3-cycles can be realized in $O(n)$ gates. Hence, the consensus function can be realized in $O(n)$ gates. As we shall see, the fact that the consensus function is heavily unbalanced means that a concise realization is possible. This trend is also evident in the realization of threshold functions.

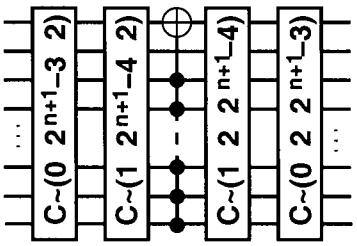


Figure 5.4: A realization of the consensus function.

A Realization of the Threshold functions T_n, T_1, T_{n-1}, T_2, T_{n-2}

Recall that a threshold function T_k on n variables is 1 if and only if the weight of the assignment is greater or equal to k. Since the threshold function is unbalanced—unless $k = \lceil n/2 \rceil$ and n is odd—an additional line, labeled 0, is required to realize the threshold function.

If the threshold functions T_n and T_1 are treated like the conjunction and disjunction functions on n variables, which correspond to an odd permutation, a realization of T_n would also realize an n-Toffoli gate on $n+1$ lines—something that is not possible [CG75]. However, the threshold function T_n can be embedded into an even permutation, say $(2^{n+1}-2\ 2^{n+1}-1)(0\ 2)$. By Corollary 5.33, which we prove later, this permutation is realizable in $O(n)$ gates. Hence, both threshold functions T_n and (by duality) T_1 can be realized in $O(n)$ gates. To gain further insight into reversible circuit constructions, we consider more direct realizations of the threshold functions T_k, $1 < k < n$.

We first show how to realize the threshold functions T_{n-1} and T_2. We only show how to realize T_{n-1} since the realization of T_2 easily follows by negating the inputs and outputs.

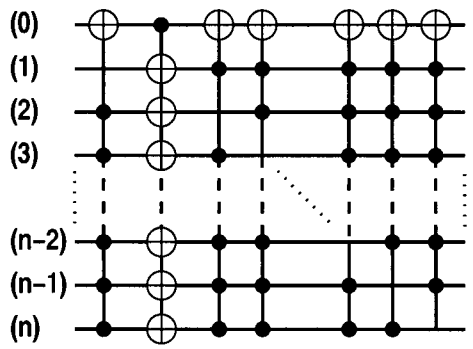


Figure 5.5: A realization of the threshold function T_{n-1}.

The realization, which is shown in Figure 5.5, comprises one controlled n-NOT gate and n $(n-1)$-Toffoli gates. The circuit is composed of three stages. Stage one contains one $(n-1)$-Toffoli gate that is controlled by lines $1,\ldots,n-1$ and toggles line 0. Stage two consists of the controlled n-NOT component that is controlled by line 0 and negates the lines $1,\ldots,n$. Finally, stage three consists of the remaining $(n-1)$-Toffoli gates, each of which toggles line 0 and is controlled by a

different subset of the lines $1,\ldots,n$; the subset $1,\ldots,n-1$ is naturally excluded as a choice.

If the input is of weight less than $n-1$, then given that line 0 is initialized to 0, none of the $(n-1)$-Toffoli gates performs the XOR; if the input is such that the lines $1,\ldots,n-1$ have value 1, then the first $(n-1)$-Toffoli gate toggles line 0, the second stage toggles all lines, which means that none of the $(n-1)$-Toffoli gates in stage three will toggle line 0; this also covers the case when the input is of weight n. Otherwise, if the input is of weight $(n-1)$, and line n is 1, then none of the gates in the first two stages will toggle any of the lines, and only one of the $(n-1)$-Toffoli gates in stage three toggles line 0. Consequently, line 0 will be toggled exactly once. Thus, the circuit realizes the threshold function T_{n-1} and since the circuit comprises $O(n)$ controlled-NOT and $(n-1)$-Toffoli gates, the circuit is of size $O(n^2)$. However, to realize the threshold function T_{n-2} requires $O(n^3)$ gates.

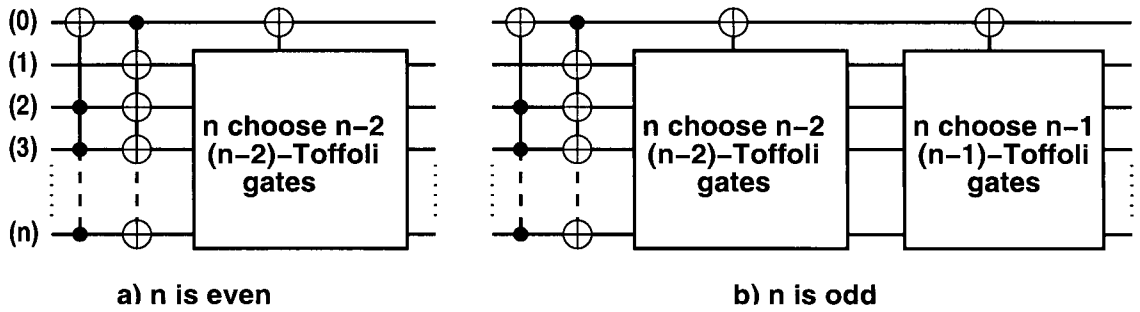


Figure 5.6: Realizations of the threshold function T_{n-2}.

The realization of the threshold function T_{n-2} depends on whether n is odd or even. If n is even then the realization (a) in Figure 5.6 is used, and if n is odd, realization (b) is used. The realizations comprise three and four stages, respectively. The first three stages of both realizations are identical to the ones in the realization of T_{n-1}. Stage three of both realizations comprises $\binom{n}{n-2}$ components, each of which is an $(n-2)$-Toffoli gate that is controlled by a different subset of the lines $1,\ldots,n$ and controls line 0—the gates are too numerous to draw in Figure 5.6. Stage four of the second realization comprises $\binom{n}{n-1}$ components, which are $(n-1)$-Toffoli gates, each controlled by a different subset of the lines $1,\ldots,n$ and each of which controls line 0.

Both realizations perform identically on inputs whose weight is n, $n-2$, or less than $n-2$; the dependence on the parity of n occurs only for inputs of weight $n-1$. For inputs of weight less than $n-2$, none of the $(n-1)$-Toffoli gates or $(n-2)$-Toffoli gates toggle line 0. For inputs of weight $n-2$, exactly one $(n-2)$-Toffoli gate in stage three toggles line 0. For inputs of weight n (or where the lines $1,\ldots,n-1$ all have value 1), the first two stages of both realizations, toggle line 0, and negate all n lines, $1,\ldots,n$, preventing all subsequent gates from toggling line 0. For inputs of weight $n-1$, where line n have value 1, the realizations perform slightly differently.

If n is even and the weight of the input is $n-1$, then line 0 will be toggled exactly $\binom{n-1}{n-2} = n-1$ times by the $(n-2)$-Toffoli gates in the last stage. Since n is even, $n-1$ is odd, and hence line 0 will be toggled an odd number of times. However, if n is odd, then $n-1$ is even and line 0

is toggled an even number of times. Since line 0 must be toggled an odd number of times if the weight of the input is greater or equal to $n-2$, the last stage in realization (b) ensures that the line is toggled one additional time if the input weight is $n-1$. Thus, the resulting realizations compute the threshold function T_{n-2}. The third stage of both realizations comprises $O(n^2)$ $(n-2)$-Toffoli gates, thus the size of the circuit is $O(n^3)$. Using these as realizations as base cases we recursively construct threshold functions T_k, where $1 < k < n$.

Realization of the Threshold functions T_k

A realization of threshold function T_k comprises two realizations of simpler threshold function realizations. Let $T_{k,m}$ denote a threshold function on m variables with threshold k—in the preceding discussion, the dependence on n was implicit, i.e., $T_k = T_{k,n}$. The two components of a realization of $T_{k,n}$ are a controlled realization of of $T_{k-1,n-1}$, and a controlled realization of $T_{k,n-1}$; the composition is illustrated in Figure 5.7.

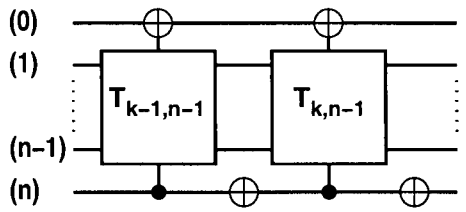


Figure 5.7: A realization of the threshold function T_k.

If line n has value 1, then the circuit needs only to check that the weight of the remaining $n-1$ lines is $k-1$ or greater. The first controlled component, which realizes $T_{k-1,n-1}$, performs this function. Otherwise, if line n has value 0, the weight of the remaining $n-1$ lines must be of weight k or greater if the threshold is to be met. The second controlled component, which realizes $T_{k,n-1}$, controlled by the negation of line n, performs this task. Each of the components are realized in the same way; the base cases, $T_{2,m}$ and $T_{m-2,m}$ are realized by the nonrecursive constructions presented in Figure 5.6.

Unfortunately, the complexity of this construction, particularly for the majority function, is exponential in n. The recurrence relation $R(k,n) = R(k-1,n-1) + R(k,n-1)$ describes the complexity of the construction in terms of the number of m-Toffoli gates, $0 \leq m < n$, where each of the two terms includes one of the two additional NOT gates in the construction. At the nth step of the recursive construction, each of the gates from the realizations of $T_{k-1,n-1}$ and $T_{k,n-1}$ are extended by one control line, which is attached to line n. Since the realizations for $T_{2,m}$ and $T_{m-2,m}$ are the base cases with complexity $c \in O(n^2)$, $R(2,m) = R(m-2,m) = c$. Recurrence $R(k,n)$ is the same as the one for binomial coefficients (with different boundary conditions), namely, $R(2,m) = R(m-2,m) = c = \binom{m}{2} \cdot (c/\binom{m}{2})$. Hence, by inspection, $R(k,n) = c \cdot \binom{n-4}{k-2}$. Finally, since each m-Toffoli gate requires $O(m)$ gates, our realization of the threshold function $T_{k,n}$ requires $O\left(n^3 \binom{n-4}{k-2}\right)$ standard gates, i.e., NOT, controlled-NOT, and Toffoli gates. Note, that as the thresh-

old function becomes more balanced—k approaches $\frac{n}{2}$—the realization becomes more complex. The problem then is to determine when a polynomial realization is possible.

5.4.2 Sufficient Conditions for Realizing Permutations by Polynomial Size Circuits

Some of the constructions in Section 5.4.1 assume that a permutation represented by a 3-cycle can realized by a circuit of size polynomial in n. In fact, any 3-cycle can be realized using $O(n)$ gates. We first prove this result in the subsequent lemma and two theorems and then discuss its implication, namely, that every permutation that can be concisely described using cycle notation, has a polynomial size realization! For the remainder of this discussion, assume that circuit $C_{(x,y,z)} \sim (xyz)$, $x,y,z \in B_n$, and in general $C_\sigma \sim \sigma$.

Lemma 5.28 *If $C_{(012)}$ is a reversible circuit on n lines, then for any $x,y \in B_n$, $x,y \neq 0$, there exists a reversible circuit C of size $O(n)$, such that the circuit $CC_{(012)}C^{-1} \sim C_{(0xy)}$.*

Proof: Select two lines i and j, setting $u = x_i x_j$, $v = y_i y_j$, $u,v \in B_2$, such that $u \neq v$ and $u,v \in \{1,2,3\}$. Such a choice is possible, otherwise $x = 0$, $y = 0$, or $x = y$, none of which can happen because $(0xy)$ is a 3-cycle. Call the lines i and j control lines. The circuit C consists of three stages. Stage one comprises $|x| - |u|$ Toffoli gates plus $|y| - |v|$ Toffoli gates. The first subsequence is bracketed by a pair of NOT gates on line i (j) if x_i (x_j) is 0; the second sequence is analogously bracketed if y_i (y_j) is 0. For each $k \neq i,j$ if $x_k = 1$ a Toffoli gate $\bigoplus_k^{i \wedge j}$, controlled by lines i and j, toggles line k. The second subsequence of Toffoli gates is analogously specified. Thus, on input x or y, all lines but lines i and j are toggled to 0.

Stage two swaps line i with line 1 and line j with line 2. This can be done using $O(1)$ gates. Finally, stage three manipulates lines 1 and 2 since these lines now hold the value of the control lines. If $u \neq 1$ and $v \neq 2$, then stage three maps u to 1 and v to 2; this also takes $O(1)$ gates. Therefore, circuit C maps input 0 to 0, input x to 1, and input y to 2, using $O(|x| + |y|) \subseteq O(n)$ gates. The circuit may permute other points in B_n, but this is of no consequence.

Since $C \sim (x\,1\ldots y\,2\ldots)$, composing circuit C with $C_{(012)}$ in the form of a conjugate yields

$$CC_{(012)}C^{-1} \sim (x\,1\ldots y\,2\ldots)(012)(x\,1\ldots y\,2\ldots)^{-1} = (0xy) \sim C_{(0xy)},$$

which completes the proof. ■

Corollary 5.29 *If $C_{(0xy)}$ is a reversible circuit on n lines, then there exists a reversible circuit C of size $O(n)$, such that the circuit $CC_{(0xy)}C^{-1} \sim C_{(012)}$.*

Theorem 5.30 follows easily from the lemma and the corollary.

Theorem 5.30 (3-cycle Hardness Theorem) *If $C_{(xyz)}$ is a reversible circuit on n lines, then for any distinct $x',y',z' \in B_n$, there exists a circuit C of size $O(n)$, such that $CC_{(xyz)}C^{-1} \sim C_{(x'y'z')}$.*

Proof: Since XORing the input with a constant bit vector can be performed by $O(n)$ NOT gates, we can transform $C_{(xyz)}$ into $C_{0\bar{y}\bar{z}}$, where $\bar{y} = y \oplus x$ and $\bar{z} = z \oplus x$. Similarly, for a circuit $C_{(0\bar{y}'\bar{z}')}$, where $\bar{y}' = y' \oplus x'$ and $\bar{z}' = z' \oplus x'$, can be transformed into $C_{(x'y'z')}$ using $O(n)$ gates. Let C_x be the

circuit comprising $|x|$ not gates such that $C_x C_{(xy\bar{z})} C_x^{-1} \sim C_{(0y\bar{z})}$ and correspondingly let $C_{x'}$ be such that $C_{(x'y'z')} \sim C_{x'} C_{(0\bar{y}'\bar{z}')} {C_{x'}}^{-1}$.

By Corollary 5.29, circuit $C_{(0y\bar{z})}$ can be transformed into $C_{(012)}$ and by Lemma 5.28, this circuit can be transformed into $C_{(0\bar{y}'\bar{z}')}$, also in $O(n)$ gates. Let C_1 and C_2 be the circuits such that $C_{(012)} \sim C_1 C_{(0y\bar{z})} C_1^{-1}$ and $C_{(0\bar{y}'\bar{z}')} \sim C_2 C_{(012)} C_2^{-1}$.

Since

$$C_{(x'y'z')} \sim C_{x'} C_2 C_1 C_x C_{(xyz)} C_x^{-1} C_1^{-1} C_2^{-1} {C_{x'}}^{-1},$$

setting $C = C_{x'} C_2 C_1 C_x$, which is of size $O(n)$, completes the proof. ■

Thus, all 3-cycles are equally hard to realize in the sense that, given a polynomial size realization of one 3-cycle, any other 3-cycle can be realized by using an additional $O(n)$ gates. Fortunately, a 3-cycle can be realized by a reversible circuit of size $O(n)$.

Theorem 5.31 *For $n > 1$ there exists reversible circuit $C_{(012)}$ on n lines of size $O(n)$.*

Proof: If $n \leq 3$ we can construct a reversible circuit that realizes any permutation and uses a constant number of gates.

For $n > 3$, observe that (012) may be factored into $\tau_1 = (01)(63)$ and $\tau_2 = (02)(63)$. Thus, we need only demonstrate that permutations τ_1 and τ_2 can be realized in $O(n)$ gates.

First, the permutation $(01)(23)$ (respectively $(02)(13)$) may be realized by $O(n)$ gates. The circuit comprises three stages: a negation, followed by a toggling, followed by a negation. Stages one and three negate the $n-2$ lines $3,\ldots,n$. The middle stage is an $(n-2)$-Toffoli gate, controlled by lines $3,\ldots,n$, that toggles line 1 (respectively line 2); each stage requires $O(n)$ gates. Let $C_{(01)(23)} \sim (01)(23)$ and $C_{(02)(13)} \sim (02)(13)$ respectively.

Next, we construct a reversible circuit $C_{(01)(63)}$ that realizes permutation $(01)(63)$. The reversible circuit $C_{\sigma_1} = \bigoplus_1 \bigoplus_3^{1\wedge 2} \bigoplus_1$ realizes a permutation that transposes 2 and 6 and whose fixed-points include all points that are congruent to 0, 1, or 3 modulo 4. Since the conjugate $\sigma_1(01)(23)\sigma_1 = (01)(63)$, therefore $C_{(01)(63)} = C_{\sigma_1} C_{(01)(23)} {C_{\sigma_1}}^{-1}$.

Similarly, we construct $C_{(01)(63)}$. The circuit $C_{\sigma_2} = \bigoplus_2 \bigoplus_3^{1\wedge 2} \bigoplus_2$ transposes points 1 and 5, with fixed-points comprising all points that are congruent to 0, 2, and 3 modulo 4. Using conjugation, we construct circuit $C_{(02)(53)} = C_{\sigma_2} C_{(02)(13)} C_{\sigma_2}$. The circuit $C_\rho = \bigoplus_1^{2\wedge 3} \bigoplus_2^{1\wedge 3} \bigoplus_1^{2\wedge 3}$, switches the values of the lines 1 and 2, using line 3 as the control. Permutation ρ transposes 5 and 6; only points congruent to 5 or 6 modulo 8 are permuted. Since the conjugate $\rho(02)(53)\rho^{-1} = (02)(63)$, therefore $C_{(02)(63)} = C_\rho C_{(02)(53)} C_\rho^{-1}$.

The required circuit is $C_{(012)} = C_{(01)(63)} C_{(02)(63)}$ is of size $O(n)$. ■

In conjunction with Theorem 5.30, we get the following two corollaries.

Corollary 5.32 *Any 3-cycle can be realized by a reversible circuit on $n > 1$ lines of size $O(n)$.*

Corollary 5.33 *A permutation on B_n comprising two disjoint transpositions, can be realized by a reversible circuit on n lines of size $O(n)$.*

Proof: If $\sigma = (ab)(cd)$, then σ can be factored into two 3-cycles $\sigma = (abc)(cad)$. ■

Since every cycle comprising m points can be represented by $m-1$ transpositions, another corollary is that a permutation can be realized using $O(n^{k+1})$ gates if its cycle representation—the number of non-fixed-points—is $O(n^k)$ in size.

Theorem 5.34 *A permutation on B_n that permutes $O(n^k)$ points can be realized by a reversible circuit on n lines of size $O(n^{k+1})$.*

Thus, any function that can be embedded into an even permutation that has a polynomial size cycle representation can be realized by a polynomial size reversible circuit. The converse is not true. There are many functions, such as negation or the incrementor, that have an exponential cycle representation, but a concise reversible circuit realization. This chapter culminates with a description of some design heuristics for reversible circuits.

5.4.3 Techniques and Heuristics for Reversible Circuit Constructions

Several techniques and heuristics have emerged for realizing functions with reversible circuits. These techniques include: using commutators, using conjugation, embedding functions within permutation groups, and taking advantage of "don't cares" to yield polynomially realizable permutations.

The commutator of two circuits $[C_\sigma, C_\tau] = C_\sigma C_\tau C_\sigma^{-1} C_\tau^{-1}$—assuming that permutations σ and τ do not commute—provides a useful mechanism for constructing controlled circuits in a bandwidth-limited environment. Although this is not a major issue in the unbounded-bandwidth framework, it is useful for combining circuits that are controlled by distinct sets of control lines into one that is controlled by union of the control lines. This technique is used by Barenco et al. [BBD+95] to construct $(n-2)$-Toffoli gates.

Conjugating one group element by another, $\sigma\tau\sigma^{-1}$, preserves the cycle structure of τ, but changes the points on which the permutation operates. Conjugating one reversible circuit with another, $C_\sigma C_\tau C_\sigma^{-1}$, performs the same function, the structure of the realized permutation remains unchanged, but the inputs that the resulting circuit permutes are changed. This technique is most useful for adapting a circuit that does 'almost the right thing' to one that performs the required permutation. Conjugation was heavily used in the preceding subsection, particularly in the construction and transformation of 3-cycles. Conjugation allows the circuit designer to decouple circuit structure from input representation, i.e., if the structure of the permutation that is realized by the circuit is correct, then the circuit can easily be adapted to work on the right set of inputs.

Permutation functions induce a strict structure on the reversible circuit, namely, the circuit must perform a very specific permutation, modulo the parity issue—odd parity functions need to be embedded into even parity permutations. Boolean functions of the form $f : B_n \rightarrow B_1$, need to be embedded into a permutation on either B_n, B_{n+1}, or B_{n+2}—in the case of the conjunction and disjunction functions. Since the right embedding allows a much more efficient realization of the function than a wrong embedding, choosing the right embedding is paramount. An example of this is choosing a polynomially representable permutation rather than an exponentially representable one. In this

case, we have a simple method to construct a polynomial size circuit. First, decompose the permutation into transpositions. Create a realization for each transposition using the technique from Corollaries 5.32 and 5.33. Concatenate the transpositions to yield the final circuit.

If a function is heavily unbalanced, i.e., the ratio of 1s to 0s in the truth table is $2^{\pm\Omega(n)}$, then the number permutations that must toggle the output line is small. Thus, the circuit can be constructed by realizing a transposition for each entry belonging to the minority of the truth and concatenating the realizations. This does not always yield an optimal circuit; by allowing the circuit to perform additional permutations that do not affect the the output line, a size reduction may be obtained.

The realization of the threshold function T_{n-1} permutes two of the 2^n inputs. If we were to use the transposition composition technique for realizing the circuit, the number of gates, used in the alternative realization would be much greater than in the hand-crafted realization. Since we "don't care" what the circuit does for certain inputs, we can craft the circuit, by ignoring entire ranges of inputs, a similar method, involving Karnaugh-maps [Kar53], is used for optimizing general combination circuits.

Unfortunately, if a function is balanced or the permutation is not polynomially representable, it is not known how to determine if an efficient realization is possible or how to derive one. One approach is to decompose the function into a linear component and a nonlinear component, i.e., $f(x) = L(x) \oplus N(x)$, where L and N are the linear and nonlinear components. The former can be realized in $O(n)$ gates and if the latter can be embedded into a polynomially representable permutation, then f can be efficiently realized. However, in the case of some functions, such as the majority function, this technique does not suffice; it is not even clear if the majority function can even be efficiently realized.

Chapter 6

Conclusion and Future Work

In this thesis we investigated growth processes on formulas and reversible circuits, and the complexity of finite Boolean functions in the reversible circuit model. First, we analyzed growth processes on formulas, characterizing the processes based on the existence and shape of their limiting distribution. Since a comparable characterization of growth processes on general circuits is intractable, due to the dependencies between the probabilities associated with various circuit components, we introduced a growth process over reversible circuits. As before, we characterized these processes based on the existence and shape of their limiting distribution. Second, we investigated the complexity of finite Boolean functions within the framework of reversible circuits and derived relationships between reversible circuits and other models of computation.

In the first part of the thesis we derived a method—applicable under a broad set of conditions—for characterizing growth processes on formulas. The characterizations included growth processes that use linear, self-dual, and monotone connectives. Unlike growth processes that use linear or self-dual connectives, where the limiting distribution is either uniform over the support or concentrated on a single function, we showed that growth processes that use balanced monotone connectives have a limiting distribution that is uniform over the slice functions. To prove this, we created a novel technique for analyzing growth processes that combines amplification arguments [Val84, Bop85] and spectral analysis [Raz88, Sav90, Sav95a].

Additionally, we derived convergence bounds for most of these growth processes and proved that the convergence rate of a process strongly depends on the connective. In most cases we showed that the convergence is logarithmic in the number of variables. Our analysis of the characteristic polynomial of the monotone connectives yielded a well defined set of monotone connectives that have no internal fixed-points, but whose corresponding growth processes take exponentially more time to converge!

A general theory for all connectives—not just balanced [Sav90], linear, or monotone—remains to be developed. Initial empirical work has shown that the limiting distribution of a growth process ceases to be uniform if the characteristic polynomial of the corresponding connective has multiple internal fixed-points; current techniques are inapplicable in this situation. While there is evidence of intriguing relationships between the characteristic polynomial of a connective and the limiting distribution, we cannot make any conjecture at this time. A first step is to consider connectives

such that for every depth d greater than some constant, the projection function can be realized by a formula that is a complete depth-d tree built from the connective.

In contrast to growth processes on formulas, we showed that the limiting distribution of a growth process on reversible circuits—if one exists—is guaranteed to be uniform over the support of the growth process. Furthermore, we showed that either a limiting distribution exists, or the growth process has two alternating distributions. Hence, we derived a broad set of sufficient conditions under which a limiting distribution exists. We introduced the notion of gate-symmetry and showed that apart from two cases, the limiting distribution exists if the support of the initial distribution is gate-symmetric.

The notion of gate-symmetry also proved useful in characterizing the support of a growth process. We showed that if the support of the initial distribution of a growth process is gate-symmetric, then in all cases but one, the support of the respective growth process could be characterized. As part of the characterization we introduced the notion of the bandwidth of a reversible circuit; the growth processes whose supports were not amenable to characterization were growth processes on bandwidth-2 circuits.

Characterizing the support of the growth processes allowed us to bound their convergence rates. We showed that the convergence rate is very slow—polynomial in the diameter of the corresponding Cayley graph—in contrast to the convergence rates of growth processes on formulas. This is due to the fact that unlike growth processes on formulas, where the formula is grown by a constant multiple every iteration, growth processes on reversible circuits grow the circuit by an additive constant every iteration. Thus, it is not surprising that the convergence rate of the latter processes is much slower.

In the third part of the thesis we investigated the reversible circuit complexity of finite Boolean functions. We first showed that bounded-bandwidth reversible circuits can be simulated by bounded-width permutation branching programs and vice-versa; bandwidth-2 reversible circuits have exactly the same power as width-4 permutation branching programs and reversible circuits of greater bounded bandwidth are characterized by the class $\mathbf{NC}^1$. Within the context of bounded width permutation branching programs, we introduced a transformation framework, under which we showed that weakly accepting and monotonically accepting width-4 programs have the same computation power—a weakly accepting program can be transformed into a monotonically accepting one with only a constant-factor increase in size. However, we proved that such weakly accepting programs cannot be transformed into strongly accepting ones. This is in stark contrast to programs of width 5 or more, for which we derived a transformation that transforms every weakly accepting program into a strongly accepting one. Thus, we demonstrated a natural gap between width-4 and width-5 programs; correspondingly, there is also a natural gap between bandwidth-2 and bandwidth-3 reversible circuits.

We considered polylogarithmic bandwidth circuits and defined a natural hierarchy of complexity classes, $\mathbf{RC}^i$. We showed that $\mathbf{RC}^i \subseteq \mathbf{SC}^i \subseteq \mathbf{RC}^{2i}$ and hence, that the union of all $\mathbf{RC}^i$, $\mathbf{RC}$, is equal to $\mathbf{SC}$. Thus, polylogarithmic bandwidth bounded circuits provide another natural framework for studying low-lying complexity classes such as $\mathbf{SC}$.

Lastly, within the context of unbounded bandwidth reversible circuits we derived concrete constructions and upper bounds for several common functions including the incrementor, adder, consen-

sus, and threshold functions. We proved that if a function can be embedded into a permutation that has a polynomial size cyclic representation, then the function has a polynomial size realization; we presented an explicit method for constructing a polynomial size realization of the respective functions. This is in contrast to the space parsimonious reversible construction of Lange et al. [LMT00], which only guarantees an exponential size realization.

There are several tracks along which this research can proceed. Our investigation has yielded new insight into the problem of whether polynomial size width-4 permutation branching programs can compute the conjunction of n variables, but the answer still remains elusive. One approach to this problem is to further investigate various program transformations. If it is possible to transform a weakly accepting program into a strongly accepting one, with a bounded increase in size, then a construction for a polynomial size width-4 program that computes the conjunction follows. Otherwise, this provides further evidence that the conjecture of Barrington [Bar89] is correct.

While the sufficient conditions for polynomial-size realizations are useful, we would also like to ascertain when a function is not polynomially realizable. For example, our construction for the majority function is exponential in the number of variables. The question remains as to whether this bound is tight. In fact the majority function is a good place to begin for characterizing functions that cannot be realized by polynomial-size reversible circuits (without an additional $O(\log n)$ lines).

Bibliography

[AB87] N. Alon and R. B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[ABH+86] M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Pudlák, V. Rödl, E. Szemerédi, and G. Turán. Two lower bounds for branching programs. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 30–38, May 1986.

[ABO84] M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computations. In *Proceedings of the 16th annual ACM Symposium on Theory of Computing*, pages 471–474, 1984.

[ABOIN96] D. Aharonov, M. Ben-Or, R. Impagliazo, and N. Nisan. Limitations of noisy reversible computation, 1996.

[Adl78] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 75–83, October 1978.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 1–9, 1983.

[Ald87] D. Aldous. On the Markov-chain simulation method for uniform combinatorial and simulated annealing. *Prob. Engng. Info. Sci.*, 1:33–46, 1987.

[Ald89] D. Aldous. Lower bounds for covering times for reversible markov chains and random walks on graphs. *Journal of Theoretical Probability*, 2(1):91–100, January 1989.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[And85] A. Andreev. A method for obtaining lower bounds for the complexity of individual monotone functions. *Dokl. Akad. Nauk USSR (Russian)*, 282(5):1033–1037, 1985. English translation in Soviet Math. Dokl. 31(3):530–534, 1985.

[Apo97] T. Apostol. *Modular Functions and Dirichlet Series in Number Theory*. Springer, 2nd edition, 1997.

[Bab91] L. Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proceedings of the 23rd annual ACM Symposium on Theory of Computing*, pages 164–174, 1991.

[Bar85] D. Barrington. Width-3 permutation branching programs. Technical Memo MIT/LCS/TM-293, Massachusetts Institute of Technology, Laboratory for Computer Science, 1985.

[Bar89] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$. *J. Comput. System Sci.*, pages 150–164, 1989.

[BBD+95] A. Barenco, C. Bennett, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Quantum gates and circuits. *Phys. Rev. A.*, 52:3457–3467, 1995.

[BDFP83] A. Borodin, D. Dolev, F. Fich, and W. Paul. Bounds for width two branching programs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 87–93, April 1983.

[Ben73] C. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:198–200, November 1973.

[Ben82] C. Bennett. The thermodynamics of computation–a review. *International Journal of Theoretical Physics*, 21:905–940, 1982.

[Ben88a] C. Bennett. Notes on the history of reversible computation. *IBM Journal of Research and Development*, 32(1), 1988.

[Ben88b] C. Bennett. Notes on the history of reversible computation. *IBM Journal of Research and Development*, 44(1/2):270–277, January 2000 (reprint of Bennett, 1988).

[Ben89] C. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[Ber82] S. Berkowitz. On some relationships between monotone and nonmonotone circuit complexity. Technical report, Department of Computer Science, University of Toronto, Canada, Toronto, Canada, 1982.

[BG81] C. Bennett and J. Gill. Relative to a random oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq \text{co-}\mathbf{NP}^A$ with probability 1. *SIAM Journal on Computing*, 10(1):96–113, February 1981.

[BGL+93] C. Bennett, P. Gacs, M. Li, P. Vitányi, and W. Zurek. Thermodynamics of computation and information distance. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, 1993.

[Bop85] R. Boppana. Amplification of probabilistic Boolean formulas. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 20–29, October 1985.

[Bop89] R. Boppana. Amplification of probabilistic Boolean formulas. *Advances in Computing Research 5: Randomness and Computation*, pages 27–45, 1989.

[BS95] D. Barrington and H. Straubing. Superlinear lower bounds for bounded-width branching programs. *Journal of Computer and System Sciences*, 50(3):374–381, June 1995.

[BST90] D. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, December 1990.

[BT88] D. Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35(4):941–952, October 1988.

[BTV01] H. Buhrman, J. Tromp, and P. Vitányi. Time and space bounds for reversible simulation. In *arXiv:quant-ph/0101133*, 2001.

[Bur11] W. Burnside. *Theory of groups of finite order.* Cambridge University Press, 2nd edition, 1911.

[CFL83] A. Chandra, M. Furst, and R. Lipton. Multi-party protocols. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 94–99, April 1983.

[CG75] D. Coppersmith and E. Grossman. Generators for certain alternating groups with applications to cryptogaphy. *SIAM Journal on Applied Mathematics*, 29(4):624–627, December 1975.

[Cle90] R. Cleve. *Methodologies for Designing Block Ciphers and Cryptographic Protocols.* PhD thesis, University of Toronto, 1990.

[CM87] S. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987.

[Coo74] S. Cook. An observation on time-storage trade-off. *Journal of Computer and System Sciences*, 9(3):308–316, December 1974.

[Coo79] S. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 338–345, May 1979.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[DH92] K. Doerk and T. Hawkes. *Finite Soluble Groups.* Berlin: de Gruyter, 1992.

[DSC93] P. Diaconis and L. Saloff-Coste. Comparison techniques for random walk on finite groups. *Ann. Probab.*, 21:2131–2156, 1993.

[DZ92] M. Dubiner and U. Zwick. Amplification and percolation. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 258–267, October 1992.

[DZ97] M. Dubiner and U. Zwick. Amplification by read-once formulas. *SIAM Journal on Computing*, 26(1):15–38, January 1997.

[Fri91] J. Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11, 1991.

[Fro12] G. Frobenius. Uber matrizen aus nicht negativen elementen. *S.-B. Deutsch. Akad. Wiss. Berlin, Math.-Nat. Kl.*, pages 456–477, 1912.

[FSS81] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science*, pages 260–270, October 1981.

[FT82] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[GM91] Q. Gu and A. Maruoka. Amplification of bounded depth monotone read-once Boolean formulae. *SIAM Journal on Computing*, 20(1):41–55, February 1991.

[Has86] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 6–20, May 1986.

[Has89] J. Hastad. Almost optimal lower bounds for small depth circuits. *ADVCR: Advances in Computing Research*, 5, 1989.

[HN77] T. Hikita and A. Nozaki. A completeness criterion for spectra. *SIAM Journal on Computing*, 6(2):285–297, June 1977.

[HN79] T. Hikita and A. Nozaki. Corrigenda: A completeness criterion for spectra. *SIAM Journal on Computing*, 8(4):656, November 1979.

[HPV75] J. Hopcroft, W. Paul, and L. Valiant. On time versus space and related problems. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, pages 57–64, October 1975.

[HR98] T. Hikita and I. Rosenberg. A completeness criterion for uniformly delayed circuits. *Acta Applicandae Mathematicae*, 52:49–61, 1998.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. (AMS)*, 117:285–306, 1965.

[Joh94] D. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Publisher, 1994.

[Kar53] M. Karnaugh. The map method for synthesis of combinational logic circuits. *AIEE Transactions, Part I Communication and Electronics*, 72:593–599, November 1953.

[Khr71] V. Khrapchenko. Complexity of the realization of a linear function in the class of π-circuits. *Mat. Zametki, (Mathematical Notes of the Academy of Science of the USSR*, 10:21–23, 1971.

[Khr72] V. Khrapchenko. A method of of obtaining lower bounds for the complexity of π-schemes. *Mat. Zametki, (Mathematical Notes of the Academy of Science of the USSR*, 11:474–479, 1972.

[KLNS89] J. Kahn, N. Linial, N. Nisan, and M. Saks. The cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, January 1989.

[Kor65] V. Korobjov. O monotonnykh funktsiyakh algebry logiki. *Prob. Cyb.*, 13:5–28, 1965.

[Kor66] V. Korobjov. Sur le nombres des functions booleennes monotones de n variables. *C.R. Acad. Sc. Paris*, 262:1088–1090, 1966.

[Kor80] A. Korshunov. O chisle monotonnykh bulevykh funktsiĭ. *Problemy Kibernetiki*, 38:5–100, 1980.

[Kri61] R. Krichevskii. Realizations of functions by superpositions. *Prob. Cyb.*, 2:458–477, 1961.

[Kud60a] V. Kudryavcev. Completeness theorem for a class of automata without feedback couplings. *Soviet Math. Doklady*, 1:537–539, 1960.

[Kud60b] V. Kudryavcev. Problems of completeness for automatic machine systems. *Soviet Math. Doklady*, 1:146–149, 1960.

[Lad75] R. Ladner. The circuit value problem is log space complete for P. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 7, 1975.

[Lan61] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[Lec63] Y. Lecerf. Machines de turing reversibles. *C.R. Acad. Sc. Paris*, 257:2597–2600, 1963.

[Lee59] C. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, July 1959.

[LMT97] K. Lange, P. McKenzie, and A. Tapp. Reversible space equals deterministic space. In *Proceedings of the 12th IEEE Conference on Computational Complexity*, pages 45–50, 1997.

[LMT00] K. Lange, P. McKenzie, and A. Tapp. Reversible space equals deterministic space. *Journal of Computer and System Sciences*, 60(2):354–367, 2000.

[Loo65] H. Loomis, Jr. A theory of high-speed clocked logic. *IEEE Trans. Elektron. Comput.*, EC-14:157–172, 1965.

[Lov96] L. Lovász. Random walks on graphs - a survey. In D. Miklos, V. T. Sos, T. Szony, and Janos Bolyai, editors, *Combinatorics, Paul Erdős is Eighty*, volume 2, pages 353–398. Mathmatical Society Budapest, 1996.

[LS90] R. Levine and A. Sherman. A note on Bennett's time-space tradeoff for reversible computation. *SIAM Journal on Computing*, 19(4):673–677, 1990.

[LS97] H. Lefmann and P. Savický. Some typical properties of large and/or Boolean functions. *Random Structures and Algorithms*, 10:337–351, 1997.

[LTV98] M. Li, J. Tromp, and P. Vitányi. Reversible simulation of irreversible computation. *Physica D*, 120:168–176, 1998.

[Lup58] O. Lupanov. A method for synthesizing circuits. *Izv. vysshykh uchebnykh zavedenii, Radiofizika*, 1:120–140, 1958.

[Lup61a] O. Lupanov. Implementing the algebra of logic functions in terms of bounded depth formulas in the basis $\{+, *, -\}$. *Soviet Physics Doklady*, 6:107–108, 1961.

[Lup61b] O. Lupanov. O realizatsiĭ funktsiĭ algebry logiki formulami iz konechnykh klassov (formulami ogranichennoĭ glubiny) v bazise &, $\vee$, $^-$. *Problemy Kibernetiki*, 6:5–14, 1961.

[Lup65] O. Lupanov. Ob odnom podkhode k sintezu upravlyayushchikh sistem. *Problemy Kibernetiki*, 14:31–110, 1965.

[LV96a] M. Li and P. Vitányi. Reversibility and adiabatic computation: Trading time and space for energy. In *Proceedings of the Royal Society of London, Series A*, volume 452 of *A*, pages 769–789, 1996.

[LV96b] M. Li and P. Vitányi. Reversible simulation of irreversible computation. In *Proceedings of the 11th IEEE Computational Complexity Conference*, pages 306–306, 1996. Submitted to Physica D, 1997.

[LV97] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2nd edition, 1997.

[Max71] J. Maxwell. *Theory of Heat*. Longmans, Green and Co., London, 1871.

[McK81] B. McKay. The expected eigenvalue distribution of a large regular graph. *Linear Algebra and its Applications*, 40:203–216, 1981.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.

[MS56] E. Moore and C. Shannon. Reliable circuits using less reliable relays. *Journal of Franklin Institute*, 262(3):191–208, 1956.

[Neč66] E. Nečiporuk. A Boolean function. *Russian Academy of Sciences Doklady. Mathematics*, 7, 1966.

[Ost54] A. M. Ostrowski. On two problems in abstract algebra connected with horner's rule. In O. Taussky-Todd, editor, *Studies in Mathematics and Mechanics Presented to Richard von Mises*, pages 40–48. Academic Press, New York, 1954.

[Per07] O. Perron. Über Matrizen. *Mathematische Annalen*, 64:248–263, 1907.

[PH70] M. Paterson and C. Hewitt. Comparative schematology. In *Proj. MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–128, December 1970.

[Pip76] N. Pippenger. The realization of monotone Boolean functions. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pages 204–210, 1976.

[Pip79] N. Pippenger. On simultaneous resource bounds. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 307–311, October 1979.

[Pip80] N. Pippenger. Pebbling. In *Proceedings of the 5th IBM Japan Symposium on Mathematical Foundations of Computing*, 1980.

[Pud84] P. Pudlák. A lower bound on complexity of branching programs. In *Proceedings of the 11th Symposium on Mathematical Foundations of Computer Science*, volume 176 of *LNCS*, pages 480–489, September 1984.

[Raz85] A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akad. Nauk USSR*, 282:1033–1037, 1985.

[Raz88] A. Razborov. Formulas of bounded depth in basis $(\wedge, \oplus)$ and some combinatorial problems. *Voprosy Kibernetiky, USSR*, pages 149–166, 1988.

[Rez62] V. Reznik. The realization of monotonic functions by means of networks consisting of functional elements. *Soviet Physics Doklady*, 6(7):558–561, 1962.

[Rot76] O. Rothaus. On bent functions. *Journal of Combinatorial Theory*, Series A, 20:300–305, 1976.

[RS42] J. Riordan and C. Shannon. The number of two-terminal series-parallel networks. *J. Math. Phys.*, 21:83–93, 1942.

[RS94] J. Radhakrishnan and K. Subrahmanyam. Directed monotone contact networks for threshold functions. *Information Processing Letters*, 50(4):199–203, May 1994.

[Ruz79] W. Ruzzo. Tree-size bounded alternation. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 352–359, April 1979.

[Sap89] A. Sapozhenko. O chisle antitsepeĭ v mnogosloĭnykh ranzhirovannykh mnozhestvakh. *Diskretnaya Matematika*, 1:110–128, 1989.

[Sav88] P. Savický. Random Boolean formulas representing any Boolean function with asymptotically equal probability (extended abstract). In *Proceedings of the 12th Symposium on Mathematical Foundations of Computer Science*, volume 324 of *lncs*, pages 512–517. Springer, September 1988.

[Sav90] P. Savický. Random Boolean formulas representing any Boolean function with asymptotically equal probability. *Discrete Mathematics*, 83:95–103, 1990.

[Sav94] P. Savický. On the bent Boolean functions that are symmetric. *European Journal of Combinatorics*, pages 407–410, 1994.

[Sav95a] P. Savický. Bent functions and random Boolean formulas. *Discrete Mathematics*, 147:211–234, 1995.

[Sav95b] P. Savický. Improved Boolean formulas for the Ramsey graphs. *Random Structures and Algorithms*, 6:407–415, 1995.

[Sav98] P. Savický. Complexity and probability of some Boolean formulas. *Combinatorics, Probability and Computing*, 7(4):451–463, 1998.

[Sha38] C. Shannon. A symbolic analysis of relay and switching circuits. *AIEE Trans.*, 57:713–723, 1938.

[Sha48] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[Sha49] C. Shannon. The synthesis of two–terminal switching circuits. *Bell System Technical Journal*, 28:59–98, 1949.

[Sip80] M. Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10(3):335–338, March 1980.

[SW49] C. Shannon and W. Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.

[Szi29] L. Szilard. Über die Entropieverminderung in einem thermodynamischen System bei eingriffen intelligenter wesen. *Zeitschrift für Physik*, 53:829–856, 1929.

[Tar88] E. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1), 1988.

[Tof80] T. Toffoli. Reversible computing. In *Automata, Languages and Programming, 7th Colloquium*, volume 85 of *Lecture Notes in Computer Science*, pages 632–644, 1980.

[Val84] L. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5:363–366, 1984.

[vN56] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[Weg79] I. Wegener. Switching functions whose monotone complexity is nearly quadratic. *Theoretical Computer Science*, 9(1):83–97, July 1979.

[Weg82] I. Wegener. Boolean functions whose monotone complexity is of size $n^2 \log n$. *Theoretical Computer Science*, 21(2):213–224, November 1982.

[Weg87] I. Wegener. *The Complexity of Boolean Functions*. Wiley Teubner Series in Computer Science. John Wiley and Sons, New York, 1987.

[Yao83] A. Yao. Lower bounds by probabilistic arguments (extended abstract). In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 420–428, 1983.

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 1–10, October 1985.

[Zur89] W. Zurek. Thermodynamic cost of computation, algorithmic complexity and the information metric. *Nature*, 341:119–124, September 1989.