Digital yet Deliberately Random: Synthesizing Logical Computation on Stochastic Bit Streams

A DISSERTATION

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Weikang Qian

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Doctor of Philosophy

Advisor: Marc D. Riedel

July, 2011

Acknowledgements

First and foremost, I want to thank my advisor Prof. Marc Riedel. It is a honor to be his first Ph.D. student. Throughout my graduate study, Marc has spent numerous amount of time to mentor me in my research. He also gives me a lot of guidance and advice on my career development. I appreciate his financial support which allows me to devote all of my attention to the research. As a student of him, I not only have developed expertise in several areas, but also have learned some important skills in research.

My sincere thank also goes to Prof. David Lilja, Prof. Kia Bazargan, and Prof. Sachin Sapatnekar at the University of Minnesota, for their kind help and mentoring throughout my graduate study.

I am grateful to Peng Li and Dr. Xin Li at the University of Minnesota, Hongchao Zhou and Prof. Jehoshua Bruck at Caltech, and Prof. Ivo Rosenberg at the University of Montreal, for their valuable feedback to my research and their collaboration in the stochastic computing project.

Last but not the least, I would like to thank my father Guoxing Qian and my mother Yafen Xu, for their love and encouragement. Without their support in my education from the very beginning, I would not have earned my Ph.D. degree.

Abstract

Most digital circuits process information that is encoded as zeros and ones deterministically. For example, the arithmetic unit of a modern computer performs calculations on deterministic integer or floating-point values represented in binary radix. However, digital computation need not be deterministic. In this dissertation, we consider an alternative paradigm: digital circuits that compute on stochastic sequences of zeros and ones. Such circuits can implement complex arithmetic operations with very simple hardware. For instance, multiplication can be performed with a single AND gate. Also they are highly tolerant of soft errors (i.e., bit flips). In the first part of the dissertation, we present a general method for synthesizing digital circuitry that computes on stochastic bit streams. Our method can be used to synthesize arbitrary polynomial functions. Through polynomial approximations, it can also be used to synthesize non-polynomial functions. Experiments on polynomial functions and functions used in image processing show that our method produces circuits that are highly tolerant of soft errors. The accuracy degrades gracefully with the error rate. For applications that mandate simple hardware, producing relatively low precision computation very reliably, our method is a winning proposition.

A premise for the stochastic paradigm is the availability of stochastic bit streams with the requisite probabilities. Physical random sources can be exploited to generate such random bit streams. Generally, each source has a fixed bias and so provides bits that have a specific probability of being one versus zero. If many different probability values are required, it can be difficult or expensive to generate all of these directly from physical sources. In the second part of the dissertation, we demonstrate novel techniques for synthesizing combinational logic that transforms a set of *source* probabilities into different *target* probabilities. We consider three scenarios in terms of whether the source probabilities are *specified* and whether they can be *duplicated*. We present solutions to all of these three scenarios.

In the final part of the dissertation, we consider *optimizing* the circuits that compute on stochastic bit streams. We focus on a fundamental problem pertaining to generating probabilities: how to synthesize optimal two-level logic circuits to generate arbitrary probability values from unbiased input probability values of 0.5? This motivates a novel logic synthesis problem: find a Boolean function with exactly a given number of minterms and having a sum-of-product expression with the minimum number of products. A crucial step towards solving the problem is to determine whether there exists a set of cubes to satisfy a given *intersection pattern* of these cubes and, if it exists, to synthesize a set of cubes. We show a necessary and sufficient condition for the existence of a set of cubes to satisfy a given intersection pattern. We also show that the synthesis problem can be reduced to the problem of finding a non-negative solution of a set of linear equalities and inequalities.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Humans are accustomed to counting in a positional number system – decimal radix. Nearly all computer systems operate on another positional number system – binary radix. From the standpoint of *representation*, such positional systems are compact: given a radix b, one can represent b^n distinct numbers with n digits. Each choice of the digits $d_i \in \{0, \ldots, b-1\}$, $i = 0, \ldots, n-1$, results in a different number N in $[0, \ldots, b^n - 1]$:

$$N = \sum_{i=0}^{n-1} b^i d_i.$$

However, from the standpoint of *computation*, positional systems impose a burden: for each operation such as addition or multiplication, the signal must be "decoded," with each digit weighted according to its position. The result must be "re-encoded" back in positional form. Any student who has designed a binary multiplier in a course on logic design can appreciate all the complexity that goes into wiring up such an operation.

Consider instead digital computation that is based on a *stochastic representation* of data: each real-valued number x ($0 \leq x \leq 1$) is represented by a sequence of random bits, each of which has probability x of being one and probability $1 - x$ of being zero. These bits can either be serially streaming on a single wire or in parallel on a bundle of wires. When serially streaming, the signals are probabilistic in *time*, as illustrated in Figure 1.1(a); when in parallel, they are probabilistic in *space*, as illustrated in Figure 1.1(b). Throughout this dissertation, we frame the discussion in terms of serial bit streams. However, our approach is equally applicable to parallel wire bundles. Indeed, we have advocated this sort of stochastic representation for technologies such as nanowire crossbar arrays [1].

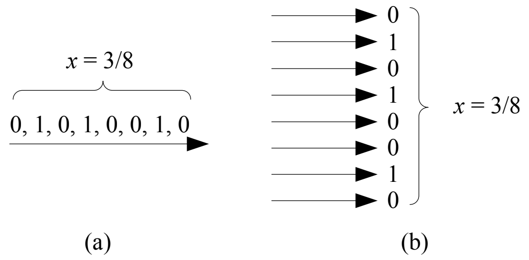


Figure 1.1: Stochastic representation: (a) A stochastic bit stream; (b) A stochastic wire bundle. A real value x in the unit interval $[0, 1]$ is represented as a bit stream or a bundle. For each bit in the bit stream or the bundle, the probability that it is one is x.

Consider the problem of designing digital circuits that operate on stochastic bit streams. We focus on combinational circuits, that is to say, memoryless digital circuits built with logic gates such as AND, OR, and NOT. For such circuits, suppose that we supply stochastic bit streams as the inputs; we will observe stochastic bit streams at the outputs. Accordingly, combinational circuits can be viewed as constructs that accept real-valued probabilities as inputs and compute real-valued probabilities as outputs. An illustration of this is shown in Figure 1.2. The circuit, consisting of an AND gate and an OR gate, accepts ones and zeros and produces ones and zeros, as any digital circuit does. If we set the input bits x_1, x_2 and x_3 to be one randomly and independently with specific probabilities, then we will get an output y that is one with a specific probability. For instance, given input probabilities $x_1 = 1/2$, $x_2 = 1$, and $x_3 = 1/4$, the circuit in Figure 1.2 produces an output y with probability 5/8.[1] The figure illustrates computations with bit lengths of 8.

[1] When we say "probability" without further qualification, we mean the probability of obtaining a one.

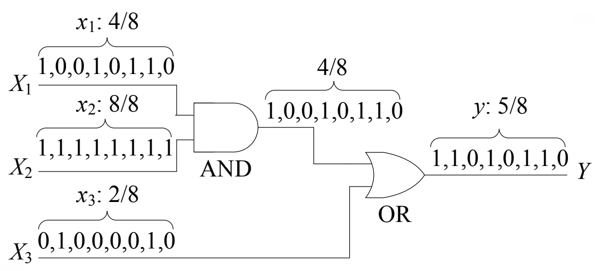


Figure 1.2: An example of logical computation on stochastic bit streams, implementing the arithmetic function $y = x_1x_2 + x_3 - x_1x_2x_3$. We see that, with inputs $x_1 = 1/2$, $x_2 = 1$ and $x_3 = 1/4$, the output is 5/8, as expected.

Compared to a binary radix representation, a stochastic representation is not very compact. With M bits, a binary radix representation can represent 2^M distinct numbers. To represent real numbers with a resolution of 2^{-M}, i.e., numbers of the form $\frac{a}{2^M}$ for integers a between 0 and 2^M, a stochastic representation requires a stream of 2^M bits. The two representations are at opposite ends of the spectrum: conventional binary radix is a maximally compressed, positional encoding; a stochastic representation is an uncompressed, uniform encoding.

A stochastic representation, although not very compact, has an advantage over binary radix in terms of error tolerance. Suppose that the environment is noisy: bit flips occur and these afflict all the bits with equal probability. Compare the two representations for a fractional number of the form $\frac{a}{2^M}$ for integers a between 0 and 2^M. With a binary radix representation, in the worst case, the most significant bit gets flipped, resulting in a change of $\frac{2^{M-1}}{2^M} = \frac{1}{2}$. In contrast, with a stochastic representation, all the bits in a stream of length 2^M have equal weight. Thus, a single bit flip always results in a change of $\frac{1}{2^M}$, which is small in comparison.

With the stochastic representation, noise does not introduce more randomness. The bit streams are random to begin with, biased to specific probability values. Rather, noise distorts the bias, producing streams with different probabilities than intended. To illustrate this, consider a stochastic bit stream X that encodes a value p. With a bit flip rate of ϵ, the probability of each bit of the noise-injected bit stream to be one is

$$p' = (1-\epsilon)P(X=1) + \epsilon P(X=0) = (1-\epsilon)p + \epsilon(1-p) = \epsilon + (1-2\epsilon)p. \tag{1.1}$$

Thus, with bit flip occurring, a number p in the stochastic representation is biased to a number $\epsilon + (1-2\epsilon)p$, a change of $(1-2p)\epsilon$ in the value. Such a change is bounded by ϵ.

Figure 1.3 visually demonstrates the fault tolerance of the stochastic approach. In the figure, we compare the fault tolerance of the stochastic implementation to that of the conventional implementation of a image processing application, the *gamma correction function* (See Example 6 for a detailed definition.). We illustrated the fault tolerance of our stochastic implementation by randomly flipping a given percentage of input bits and evaluating the output. For example, a 2% error rate means that we randomly flip 2% of the input bits. We observed a dramatic improvement in fault tolerance, particularly when the magnitude of errors is analyzed: with a 10% soft error injection rate, the conventional circuit produces outputs that are more than 20% off over 37% the time. The result: nearly unreadable images. In contrast, our stochastic circuit *never* produces pixel values with errors larger than 20%.

A stochastic representation also has the advantage over binary radix in the amount of hardware needed for arithmetic computation. Consider multiplication. Figure 1.4 shows a conventional design for a 3-bit carry-save multiplier, operating on binary radix-encoded numbers. It consists of 9 AND gates, 3 half adders and 3 full adders, for a total of 30 gates.[2]

In contrast, with a stochastic representation, multiplication can be implemented with much less hardware: we only need one AND gate. Figure 1.5 illustrates the multiplication of values that are represented by stochastic bit streams. Assuming that the two input stochastic bit streams A and B are independent, the number represented

[2] A half adder can be implemented with one XOR gate and one AND gate. A full adder can be implemented with two XOR gates, two AND gates, and one OR gate.

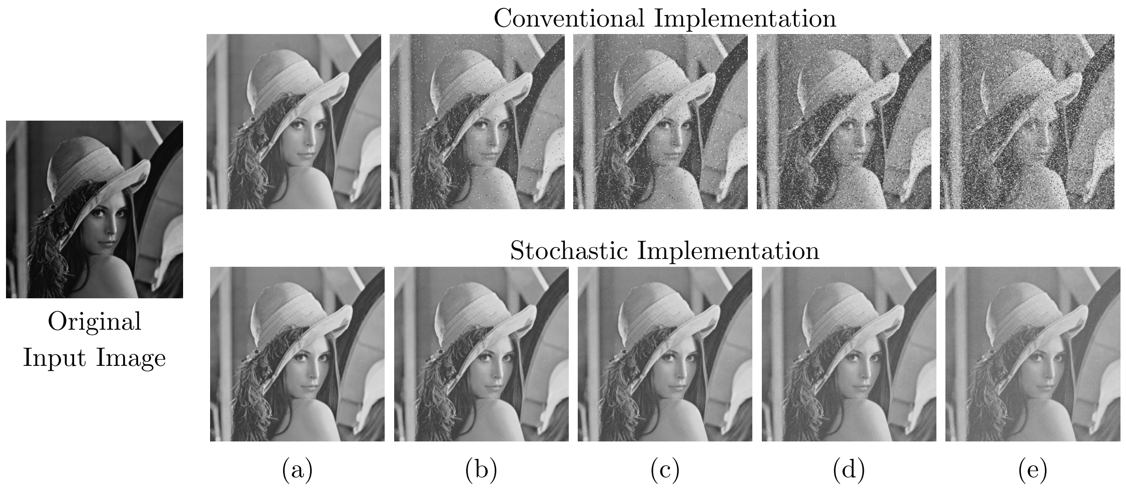


Figure 1.3: Comparison of the fault tolerance of the stochastic implementation to that of the conventional implementation of the gamma correction function. The images in the top row are generated by a conventional implementation. The images in the bottom row are generated by our stochastic implementation. Input error ratios are (a) 0%; (b) 1%; (c) 2%; (d) 5%; (e) 10%.

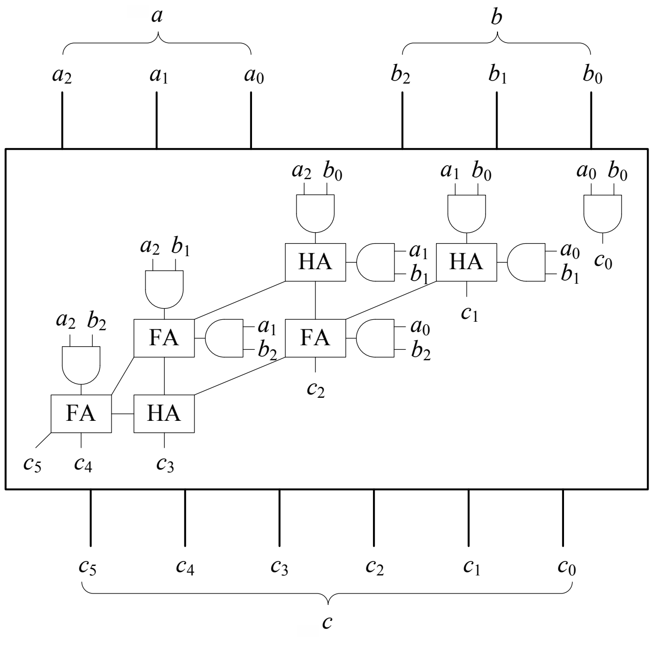


Figure 1.4: Multiplication on a conventional binary representation: a carry-save multiplier, operating on 3-bit binary radix encoded inputs A and B. “FA” refers to a full adder and “HA” refers to a half adder.

by the output stochastic bit stream C is

$$c = P(C = 1) = P(A = 1 \text{ and } B = 1) = P(A = 1)P(B = 1) = a \cdot b. \tag{1.2}$$

So the AND gate multiplies the two values represented by the stochastic bit streams. In the figure, with bit streams of length 8, the values have a resolution of 1/8.

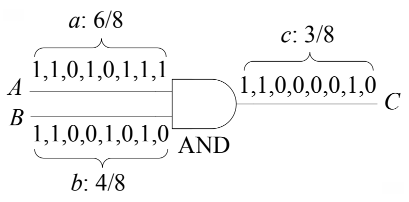


Figure 1.5: Multiplication on stochastic bit streams with an AND gate. Here the inputs are 6/8 and 4/8. The output is $6/8 \times 4/8 = 3/8$, as expected.

Why is multiplication so simple with a stochastic representation and so complex with a conventional positional representation? Although compact, a positional representation imposes a computational burden for arithmetic: for each operation we must, in essence, "decode" the operands, weighting the higher order bits more and the lower order bits less; then we must "re-encode" the result in weighted form. Since the stochastic representation is uniform, no decoding and no re-encoding are required to operate on the values.

We can perform operations other than multiplication with the stochastic representation. Consider addition. It is not feasible to add two probability values directly; this could result in a value greater than one, which cannot be represented as a probability value. However, we can perform *scaled* addition. Figure 1.6 shows a scaled adder operating on real numbers in the stochastic representation. It consists of a multiplexer (MUX), a digital construct that selects one of its two input values to be the output value, based on a third "selecting" input value. For the multiplexer shown in Figure 1.6, S is the selecting input. When $S = 1$, the output $C = A$. Otherwise, when $S = 0$, the output $C = B$. The Boolean function implemented by the multiplexer is

$C = (A \wedge S) \vee (B \wedge \neg S)$.[3] With the assumption that the three input stochastic bit streams A, B, and S are independent, the number represented by the output stochastic bit stream C is

$$\begin{aligned} c = P(C = 1) &= P(S = 1 \text{ and } A = 1) + P(S = 0 \text{ and } B = 1) \\ &= P(S = 1)P(A = 1) + P(S = 0)P(B = 1) = s \cdot a + (1 - s) \cdot b. \end{aligned} \tag{1.3}$$

Thus, with this stochastic representation, the computation performed by a multiplexer is the scaled addition of the two input values a and b, with a scaling factor of s for a and $1 - s$ for b.

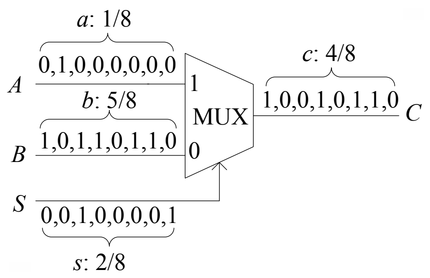


Figure 1.6: Scaled addition on stochastic bit streams, with a multiplexer (MUX). Here the inputs are $1/8, 5/8$, and $2/8$. The output is $2/8 \times 1/8 + (1 - 2/8) \times 5/8 = 4/8$, as expected.

1.1 Overview

The task of *analyzing* combinational circuitry operating on stochastic bit streams is well understood [2]. For instance, it can be shown that, given an input x, an inverter (i.e., a NOT gate) implements the operation $1 - x$. Given inputs x and y, an OR gate implements the operation $x + y - xy$. Analyzing the circuit in Figure 1.2, we see that it implements the function $x_1x_2 + x_3 - x_1x_2x_3$. Aspects such as signal correlations of reconvergent paths must be taken into account. Algorithmic details for such analysis

[3] When discussing Boolean functions, we will use $\wedge$, $\vee$, and $\neg$ to represent logical AND, OR, and negation, respectively. We adopt this convention since we use $+$ and $\cdot$ to represent *arithmetic* addition and multiplication, respectively.

were first fleshed out by the testing community [3]. They have also found mainstream application for tasks such as timing and power analysis [4, 5].

In Chapter 3, we explore the more challenging task of *synthesizing* logical computation on stochastic bit streams that implements the functionality that we want. We have developed a general method for synthesizing arbitrary univariate polynomial functions on stochastic bit streams. A necessary condition is that the target polynomial maps the unit interval onto the unit interval. Our major contribution is to show that this condition is also sufficient: we provide a constructive method for implementing any polynomial that satisfies this condition. Our method is based on some novel mathematics for manipulating polynomials in a special form called a Bernstein polynomial. We introduce the definition and the related properties of Bernstein polynomials in Chapter 2.

A premise for the stochastic paradigm is that we have access to input stochastic bit streams with desired probabilities. Physical sources of randomness can be exploited to generate such bit streams [6]. Generally, each source has a fixed bias and so provides bits that have a specific probability of being one versus zero. For schemes that generate stochastic bit streams from physical sources, a significant limitation is the cost of generating different probability values. For instance, if each probability value is determined by a specific voltage level, different voltage levels are required to generate different probability values. For an application that requires many different values, many voltage regulators are required; this might be prohibitively costly in terms of area as well as power consumption [7].

In Chapter 4, we propose a strategy to mitigate this issue: we demonstrate a method for synthesizing combinational logic that transforms a set of *source* probabilities into different *target* probabilities. We consider three scenarios in terms of whether the source probabilities are *specified* and whether they can be *duplicated*.

In the case that the source probabilities are not specified and can be duplicated, we provide a specific choice, the set $\{0.4, 0.5\}$; we show how to synthesize logic that transforms probabilities from this set into arbitrary *decimal* probabilities. Figure 1.7

shows a circuit synthesized by our algorithm to realize the decimal output probability 0.119 from the input probabilities 0.4 and 0.5. The circuit consists of AND gates and inverters: each AND gate performs a multiplication of its inputs and each inverter performs a one-minus operation of its input.

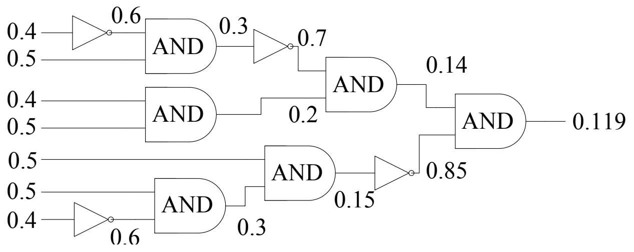


Figure 1.7: A circuit synthesized by our algorithm to realize the decimal output probability 0.119 from the input probabilities 0.4 and 0.5.

In the case that the source probabilities are specified and cannot be duplicated, we provide two methods for synthesizing logic to transform them into target probabilities. The first method is based on linear 0-1 programming and gives an optimal approximation to the targe probability. The second method is a greedy constructive method and gives a suboptimal result.

In the case that the source probabilities are not specified, but once chosen cannot be duplicated, we provide an optimal choice of the set of source probabilities.

Area is an important concern in digital circuit design. By reducing the silicon area, the manufacturing cost of the circuit is reduced and so is the power dissipation. Circuits operating on stochastic bit streams will also benefit from small area. In Chapter 5, we further consider the problem of optimizing the area of circuits that compute on stochastic bit streams. We start with a fundamental problem pertaining to generating probabilities: synthesizing a circuit to generate an arbitrary given probability value from a set of independent inputs of an unbiased probability value of 0.5. We focus on synthesizing two-level logic circuit. This motivates a novel problem in logic synthesis:

find a Boolean function with exactly a given number of minterms and having a sum-of-product expression with the minimum number of products. We call this problem *arithmetic two-level minimization problem.*

A crucial step towards solving the arithmetic two-level-minimization problem is to determine whether there exists a set of cubes to satisfy a given *intersection pattern* of these cubes and, if it exists, to synthesize a set of cubes. We call this problem λ-*cube intersection problem.* Here an intersection pattern of a set of cubes refers to a set of numbers, each of which specifies the number of the minterms covered by the intersection of one of the subsets of the set of cubes. In Chapter 5, we provide a rigorous mathematic treatment to the λ-cube intersection problem and derive a necessary and sufficient condition for the existence of a set of cubes to satisfy the given intersection pattern. The problem reduces to checking whether a set of linear equalities and inequalities has a non-negative integer solution.

Chapter 2

Bernstein Polynomials

In this chapter, we introduce a specific type of polynomial that we use, namely Bernstein polynomials [8].

Definition 1

A **Bernstein polynomial** *of degree n, denoted as $B_n(t)$, is a polynomial expressed in the following form [9]:*

$$\sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t), \tag{2.1}$$

where each $\beta_{k,n}$, $k = 0, 1, \ldots, n$, is a real number and

$$b_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}.^{1} \tag{2.2}$$

The coefficients $\beta_{k,n}$ are called **Bernstein coefficients** *and the polynomials $b_{0,n}(t), b_{1,n}(t), \ldots, b_{n,n}(t)$ are called* **Bernstein basis polynomials** *of degree n.* □

2.1 Properties of Bernstein Polynomials

We list some pertinent properties of Bernstein polynomials.

1. The *positivity* property:

[1] Here $\binom{n}{k}$ denotes the binomial coefficient "n choose k."

For all $k = 0, 1, \ldots, n$ and all t in $[0, 1]$, we have

$$b_{k,n}(t) \geq 0. \tag{2.3}$$

2. The *partition of unity* property:

   The binomial expansion of the left-hand side of the equality $(t + (1 - t))^n = 1$ shows that the sum of all Bernstein basis polynomials of degree n is the constant 1, i.e.,

$$\sum_{k=0}^{n} b_{k,n}(t) = 1. \tag{2.4}$$

3. Converting power-form coefficients to Bernstein coefficients:

   The set of Bernstein basis polynomials $b_{0,n}(t), b_{1,n}(t), \ldots, b_{n,n}(t)$ forms a basis of the vector space of polynomials of real coefficients and degree no more than n [10]. Each power basis function t^j can be uniquely expressed as a linear combination of the $n + 1$ Bernstein basis polynomials:

$$t^j = \sum_{k=0}^{n} \sigma_{jk} b_{k,n}(t), \tag{2.5}$$

   for $j = 0, 1, \ldots, n$. To determine the entries of the transformation matrix σ, we write

$$t^j = t^j (t + (1 - t))^{n-j}$$

   and perform a binomial expansion on the right hand side. This gives

$$t^j = \sum_{k=j}^{n} \frac{\binom{k}{j}}{\binom{n}{j}} b_{k,n}(t),$$

   for $j = 0, 1, \ldots, n$. Therefore, we have

$$\sigma_{jk} = \begin{cases} \binom{k}{j}\binom{n}{j}^{-1}, & \text{for } j \leq k \\ 0, & \text{for } j > k. \end{cases} \tag{2.6}$$

Suppose that a power-form polynomial of degree no more than n is

$$g(t) = \sum_{k=0}^{n} a_{k,n} t^k \tag{2.7}$$

and the Bernstein polynomial of degree n of g is

$$g(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t). \tag{2.8}$$

Substituting Equations (2.5) and (2.6) into Equation (2.7) and comparing the Bernstein coefficients, we have

$$\beta_{k,n} = \sum_{j=0}^{n} a_{j,n}\sigma_{jk} = \sum_{j=0}^{k} \binom{k}{j}\binom{n}{j}^{-1} a_{j,n}. \tag{2.9}$$

Equation (2.9) provide a means for obtaining Bernstein coefficients from power-form coefficients.

4. Degree elevation:

Based on Equation (2.2), we have that for all $k = 0, 1, \ldots, m$,

$$\begin{aligned}&\frac{1}{\binom{m+1}{k}} b_{k,m+1}(t) + \frac{1}{\binom{m+1}{k+1}} b_{k+1,m+1}(t)\\ =&t^k(1-t)^{m+1-k} + t^{k+1}(1-t)^{m-k}\\ =&t^k(1-t)^{m-k} = \frac{1}{\binom{m}{k}} b_{k,m}(t),\end{aligned}$$

or

$$\begin{aligned} b_{k,m}(t) &= \frac{\binom{m}{k}}{\binom{m+1}{k}} b_{k,m+1}(t) + \frac{\binom{m}{k}}{\binom{m+1}{k+1}} b_{k+1,m+1}(t)\\ &= \frac{m+1-k}{m+1} b_{k,m+1}(t) + \frac{k+1}{m+1} b_{k+1,m+1}(t).\end{aligned} \tag{2.10}$$

Given a power-form polynomial g of degree n, for any $m \geq n$, g can be uniquely converted into a Bernstein polynomial of degree m. Suppose that the Bernstein polynomials of degree m and degree $m+1$ of g are $\sum_{k=0}^{m} \beta_{k,m} b_{k,m}(t)$ and

$\sum_{k=0}^{m+1} \beta_{k,m+1} b_{k,m+1}(t)$, respectively. We have

$$\sum_{k=0}^{m} \beta_{k,m} b_{k,m}(t) = \sum_{k=0}^{m+1} \beta_{k,m+1} b_{k,m+1}(t). \tag{2.11}$$

Substituting Equation (2.10) into the left-hand side of Equation (2.11) and comparing the Bernstein coefficients, we have

$$\beta_{k,m+1} = \begin{cases} \beta_{0,m}, & \text{for } k = 0 \\ \frac{k}{m+1}\beta_{k-1,m} + \left(1 - \frac{k}{m+1}\right)\beta_{k,m}, & \text{for } 1 \le k \le m \\ \beta_{m,m}, & \text{for } k = m+1. \end{cases} \tag{2.12}$$

Equation (2.12) provides a means for obtaining the coefficients of the Bernstein polynomial of degree $m+1$ of g from the coefficients of the Bernstein polynomial of degree m of g. We will call this procedure *degree elevation.*

2.2 Uniform Approximation and Bernstein Polynomials with Coefficients in the Unit Interval

In this section, we present two of our major mathematical findings on Bernstein polynomials. The first result pertains to uniform approximation with Bernstein polynomials. We show that, given a power-form polynomial g, we can obtain a Bernstein polynomial of degree m with coefficients that are as close as desired to the corresponding values of g evaluated at the points $0, \frac{1}{m}, \frac{2}{m}, \ldots, 1$, provided that m is sufficiently large. This result is formally stated by the following theorem.

Theorem 1

Let g be a polynomial of degree $n \ge 0$. For any $\epsilon > 0$, there exists a positive integer $M \ge n$ such that for all integers $m \ge M$ and $k = 0, 1, \ldots, m$, we have

$$\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| < \epsilon,$$

where $\beta_{0,m}, \beta_{1,m}, \ldots, \beta_{m,m}$ *satisfy* $g(t) = \sum_{k=0}^{m} \beta_{k,m} b_{k,m}(t)$. $\square$

Please see Appendix A for the proof of the above theorem.

The second result pertains to a special type of Bernstein polynomials: those with coefficients that are all in the unit interval. We are interested in this type of Bernstein polynomials since we can show that such Bernstein polynomials can be implemented by logical computation on stochastic bit streams (See Chapter 3 for the details.).

Definition 2

Define U *to be the set of Bernstein polynomials with coefficients that are all in the unit interval* $[0, 1]$*:*

$$U = \left\{ p(t) \mid \exists\ n \geq 1, 0 \leq \beta_{0,n}, \beta_{1,n}, \ldots, \beta_{n,n} \leq 1,\ \textit{such that}\ p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) \right\}. \ \square$$

The question we are interested in is: which (power-form) polynomials can be converted into Bernstein polynomials in U?

Definition 3

Define the set V *to be the set of polynomials which are either identically equal to* 0 *or equal to* 1*, or map the open interval* $(0, 1)$ *into* $(0, 1)$ *and the points* 0 *and* 1 *into the closed interval* $[0, 1]$*, i.e.,*

$$\begin{aligned} V = \{p(t) \mid\ & p(t) \equiv 0,\ \textit{or}\ p(t) \equiv 1, \\ & \textit{or}\ 0 < p(t) < 1, \forall t \in (0, 1)\ \textit{and}\ 0 \leq p(0), p(1) \leq 1\}. \ \square \end{aligned}$$

We prove that the set U and the set V are equivalent, thus giving a clear characterization of the set U.

Theorem 2

$$V = U. \qquad \square$$

The proof of the above theorem utilizes Theorem 1. Please see Appendix B for the proof.

We end this chapter with two examples illustrating Theorem 2. In what follows, we will refer to a Bernstein polynomial of degree n converted from a polynomial g as "the Bernstein polynomial of degree n of g". When we say that a polynomial is of degree n, we mean that the power-form of the polynomial is of degree n.

Example 1

Consider the polynomial $g(t) = \frac{5}{8} - \frac{15}{8}t + \frac{9}{4}t^2$. *It maps the open interval* $(0,1)$ *into* $(0,1)$ *with* $g(0) = \frac{5}{8}$ *and* $g(1) = 1$. *Thus,* g *is in the set* V. *Based on Theorem 2, we have that* g *is in the set* U. *We verify this by considering Bernstein polynomials of increasing degree.*

- *The Bernstein polynomial of degree 2 of* g *is*

$$g(t) = \frac{5}{8} \cdot b_{0,2}(t) + \left(-\frac{5}{16}\right) \cdot b_{1,2}(t) + 1 \cdot b_{2,2}(t).$$

  Note that the coefficient $\beta_{1,2} = -\frac{5}{16} < 0$.

- *The Bernstein polynomial of degree 3 of* g *is*

$$g(t) = \frac{5}{8} \cdot b_{0,3}(t) + 0 \cdot b_{1,3}(t) + \frac{1}{8} \cdot b_{2,3}(t) + 1 \cdot b_{3,3}(t).$$

  Note that all the coefficients are in $[0,1]$.

Since the Bernstein polynomial of degree 3 of g *satisfies Definition 2, we conclude that* g *is in the set* U. $\square$

Example 2

Consider the polynomial $g(t) = \frac{1}{4} - t + t^2$. *Since* $g(0.5) = 0$, *thus* g *is not in the set* V. *Based on Theorem 2, we have that* g *is not in the set* U. *We verify this. By contraposition, suppose that there exist* $n \geq 1$ *and* $0 \leq \beta_{0,n}, \beta_{1,n}, \ldots, \beta_{n,n} \leq 1$ *such that*

$$g(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t).$$

Since $g(0.5) = 0$, *therefore,* $\sum_{k=0}^{n} \beta_{k,n} b_{k,n}(0.5) = 0$. *Note that for all* $k = 0, 1, \ldots, n$, $b_{k,n}(0.5) > 0$. *Thus, we have that for all* $k = 0, 1, \ldots, n$, $\beta_{k,n} = 0$. *Therefore,* $g(t) \equiv 0$, *which contradicts the original assumption about* g. *Thus,* g *is not in the set* U. $\square$

Chapter 3

Synthesizing Arithmetic Functions

In Chapter 1, we demonstrate that logical computation on stochastic bit streams has several advantages such as fault-tolerance and simple hardware design. Given these advantages, we are interested in how we can broadly utilize logical computation on stochastic bit streams. We consider combinational circuits. Since both its inputs and outputs are treated as probabilities, a combinational circuit operating on stochastic bit streams implements an arithmetic function. The task of *analyzing* the output function is well understood [2]. In this chapter, we focus on the *synthesis* aspect: given an arbitrary arithmetic function, how can we design a combinational circuit operating on stochastic bit streams to implement that function?

3.1 Analysis of Logical Computation on Stochastic Bit Streams

In the introduction, we showed that an AND gate implements multiplication on stochastic bit streams; a multiplexer implements scaled addition. In general, what sort of mathematical function does logical computation on stochastic bit streams implement? In this section, we will show that it implements a multivariate polynomial with integer coefficients. The degree of each variable is at most one, i.e., there are no terms with variables raised to the power of two, three or higher.

To see this, suppose that we have combinational logic that realizes the Boolean function $Y = f(X_1, X_2, \ldots, X_n)$ and that for $i = 1, \ldots, n$, the input X_i is a stochastic bit stream where each bit has probability x_i of being one. Equivalently, each input X_i can be viewed as a random Boolean variable that has probability x_i of being one, i.e., $P(X_i = 1) = x_i$. Then the probability of X_i being zero is $P(X_i = 0) = 1 - x_i$. With the inputs being random Boolean variables, the output of the combinational logic is also a random Boolean variable. Assume that the output Y has probability y of being one, i.e., $P(Y = 1) = y$. The computation implements a function describing the relation between the output probability y and the input probabilities $x_1, x_2, \ldots, x_n$. Note that y is the sum of the probability of occurrence of all combinations of input values for which the Boolean function evaluates to 1. That is,

$$y = P(Y = 1) = \sum_{\substack{(a_1,\ldots,a_n)\in\{0,1\}^n: \\ f(a_1,\ldots,a_n)=1}} P(X_1 = a_1, X_2 = a_2, \ldots, X_n = a_n).$$

With the assumption that the input random variables $X_1, X_2, \ldots, X_n$ are independent, we further have

$$y = \sum_{\substack{(a_1,\ldots,a_n)\in\{0,1\}^n: \\ f(a_1,\ldots,a_n)=1}} \left(\prod_{k=1}^{n} P(X_k = a_k) \right). \tag{3.1}$$

Since $P(X_i = a_i)$ is either x_i or $1 - x_i$, depending on the value of a_i in the given combination, it is easily seen that y is a multivariate polynomial on the arguments $x_1, x_2, \ldots, x_n$. Moreover, if we expand Equation (3.1) into a power form, each product term has an integer coefficient. In all the product terms, the degree of each variable is at most one.

Thus, we have the following theorem describing the general form of a function implemented by computation on stochastic bit streams.

Theorem 3

Logical computation on stochastic bit streams implements a multivariate polynomial of the form

$$y = F(x_1, x_2, \ldots, x_n) = \sum_{i_1=0}^{1} \cdots \sum_{i_n=0}^{1} \left(\alpha_{i_1 \ldots i_n} \prod_{k=1}^{n} x_k^{i_k} \right), \tag{3.2}$$

where the $\alpha_{i_1 \ldots i_n}$'s are integer coefficients. $\square$

Example 3

Suppose that a combinational circuit implements the Boolean function $Y = (X_1 \vee X_2) \wedge X_3$. Its truth table is shown in Table 3.1. Consider the computation performed by this combinational circuit on stochastic bit streams. Based on the truth table, we have

$$\begin{aligned} y &= P(Y = 1) \\ &= P(X_1 = 0, X_2 = 1, X_3 = 1) + P(X_1 = 1, X_2 = 0, X_3 = 1) \\ &+ P(X_1 = 1, X_2 = 1, X_3 = 1) \\ &= (1 - x_1)x_2x_3 + x_1(1 - x_2)x_3 + x_1x_2x_3 \\ &= x_1x_3 + x_2x_3 - x_1x_2x_3. \end{aligned} \tag{3.3}$$

This function is an integer-coefficient multivariate polynomial on the variables x_1, x_2, and x_3. In all the product terms, the degree of each variable is at most one. $\square$

3.2 Stochastically Computable Polynomials

Computation on stochastic bit streams generally implements a special type of multivariate polynomial on input arguments, as was shown by Theorem 3. If we associate some of the x_i's of the polynomial $F(x_1, x_2, \ldots, x_n)$ in Equation (3.2) with real constants in the unit interval and the others with a common variable t, then the function F becomes a real-coefficient *univariate* polynomial $g(t)$. With different choices of the original Boolean function f and different settings of the probabilities of the x_i's, we get

Table 3.1: A truth table for the Boolean function $Y = (X_1 \vee X_2) \wedge X_3$.

X_1	X_2	X_3	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

different polynomials $g(t)$. We call a univariate polynomial $g(t)$ obtained in this way *stochastically computable.*

Definition 4

A univariate polynomial $g(t)$ is **stochastically computable** *if it can be realized by a combinational circuit computing on stochastic bit streams, with some input streams representing the variable probability t and the other input streams representing constant probabilities.* $\square$

Example 4

Consider the combinational circuit in Example 3. The multivariate polynomial it computes on stochastic bit streams is $y = x_1x_3 + x_2x_3 - x_1x_2x_3$. If we set the probabilities of the input bit streams as $x_1 = x_2 = t$, and $x_3 = 0.8$, then the circuit implements the polynomial $g(t) = 1.6t - 0.8t^2$. If we set the probabilities of the input bit streams as $x_1 = x_2 = x_3 = t$, then the circuit implements the polynomial $g(t) = 2t^2 - t^3$. Therefore, both the polynomial $1.6t - 0.8t^2$ and the polynomial $2t^2 - t^3$ are stochastically computable. $\square$

In the rest of this chapter, we focus on univariate polynomial. When we say "a polynomial," we mean a univariate polynomial. For convenience, we give the following definition.

Definition 5

Define the set W to be the set of stochastically computable polynomial. $\square$

The first question that arises is: what kind of polynomial is stochastically computable? It is equivalently to ask what polynomials are in the set W. In this chapter, we will demonstrate that $W = V$, where V is defined in Definition 3. First, we will show $W \subseteq V$.

Theorem 4

$$W \subseteq V. \qquad \square$$

Proof. The constant polynomial 0 and 1 can be trivially implemented by logical computation on stochastic bit streams. Thus, these two polynomials are in the set W. From the definition of the set V, they are also in the set V.

Now consider any polynomial $g \in W$ such that $g \not\equiv 0$ and $g \not\equiv 1$. We will show that for all $0 < t < 1$, $0 < g(t) < 1$ and $0 \leq g(0), g(1) \leq 1$. Thus, by the definition of V, we have $g \in V$.

Since g is stochastically computable, then given any $0 \leq t \leq 1$, $g(t)$ must evaluate to a probability value. In other words, for all $0 \leq t \leq 1$, we have $0 \leq g(t) \leq 1$. We only need to prove that for all $0 < t < 1$, $g(t) \neq 0$ and $g(t) \neq 1$. We prove this by the way of contraposition. Suppose that these exists a $0 < t^* < 1$ such that $g(t^*) = 0$ or 1.

There exists a combinational circuit with the least number of inputs to implement the polynomial g on stochastic bit streams. Consider this combinational circuit. Let its Boolean function be $f(X_1, X_2, \ldots, X_n)$. Let the multivariate polynomial that the circuit computes on stochastic bit streams be $F(x_1, x_2, \ldots, x_n)$. Without loss of generality, we can assume that we set $x_1, x_2, \ldots, x_m (m \leq n)$ to be real constants $c_1, c_2, \ldots, c_m$ and $x_{m+1}, x_{m+2}, \ldots, x_n$ to be a variable t. In other words, $g(t) = F(c_1, \ldots, c_m, t, \ldots, t)$.

We claim that $c_1, \ldots, c_m \in (0, 1)$. Indeed, if, say c_1 is either 0 or 1, then the input X_1 of the combinational circuit is a deterministic zero or a deterministic one. Thus, we can simplify the original combinational circuit by removing one input. This contradicts our assumption that the combinational circuit is the one with the least number of inputs to implement the polynomial g.

From Equation (3.1), we have

$$g(t^*) = \sum_{\substack{(a_1,\ldots,a_n)\in\{0,1\}^n:\\ f(a_1,\ldots,a_n)=1}} \left(\prod_{k=1}^{n} P(X_k = a_k)\right), \tag{3.4}$$

where

$$P(X_k = 1) = \begin{cases} c_k, & \text{for } 1 \le k \le m \\ t^*, & \text{for } m < k \le n, \end{cases}$$

and

$$P(X_k = 0) = \begin{cases} 1 - c_k, & \text{for } 1 \le k \le m \\ 1 - t^*, & \text{for } m < k \le n. \end{cases}$$

Since $c_1, \ldots, c_m \in (0,1)$ and $t^* \in (0,1)$, we have that for all $(a_1, \ldots, a_n) \in \{0,1\}^n$,

$$\prod_{k=1}^{n} P(X_k = a_k) > 0.$$

We distinguish two cases of $g(t^*)$.

1. The case where $g(t^*) = 0$. Then the Boolean function f must be a constant 0. Otherwise, there exists a n-tuple $(a_1^*, \ldots, a_n^*) \in \{0,1\}^n$ such that $f(a_1^*, \ldots, a_n^*) = 1$. Consequently, the right-hand side of Equation (3.4) is larger than zero, contradicting that $g(t^*) = 0$. Since the Boolean function f is a constant 0, then the polynomial g is a constant polynomial of 0, which contradicts our initial assumption that $g \not\equiv 0$.

2. The case where $g(t^*) = 1$. Then the Boolean function f must be a constant 1. Otherwise, there exists a n-tuple $(a_1^*, \ldots, a_n^*) \in \{0,1\}^n$ such that $f(a_1^*, \ldots, a_n^*) = 0$.

Then we have

$$g(t^*) = \sum_{\substack{(a_1,\ldots,a_n)\in\{0,1\}^n:\\ f(a_1,\ldots,a_n)=1}} \left(\prod_{k=1}^{n} P(X_k = a_k)\right)$$
$$< \sum_{(a_1,\ldots,a_n)\in\{0,1\}^n} \left(\prod_{k=1}^{n} P(X_k = a_k)\right) = 1,$$

contradicting that $g(t^*) = 1$. Since the Boolean function f is a constant 1, then the polynomial g is a constant polynomial of 1, which contradicts our initial assumption that $g \not\equiv 1$.

Thus, we have proved that for all $0 < t < 1$, $g(t) \neq 0$ and $g(t) \neq 1$. In conclusion, for any polynomial $g \in W$, we have shown that $g \in V$. Therefore, $W \subseteq V$. □

Remark: Theorem 4 essentially states a necessary condition for a polynomial g to be stochastically computable: g is a constant polynomial 0, a constant polynomial 1, or a polynomial that maps the open interval $(0, 1)$ into itself and maps the points 0 and 1 into the closed interval $[0, 1]$.

3.3 Synthesizing Bernstein Polynomials with Coefficients in the Unit Interval

In this section, we will show that $U \subseteq W$. In other words, any Bernstein polynomial that has all the coefficients in the unit interval is stochastically computable [11].

Consider an arbitrary Bernstein polynomial with all the coefficients in the unit interval

$$B_n(t) = \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t), \tag{3.5}$$

where for all $0 \leq i \leq n$, $\beta_{i,n} \in [0, 1]$. We can implement the Bernstein polynomial with the construct shown in Figure 3.1.

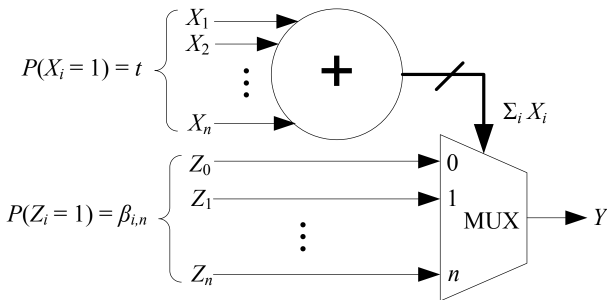


Figure 3.1: Combinational logic that implements a Bernstein polynomial with all coefficients in the unit interval. In order to implement the Bernstein polynomial $B_n(t) = \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t)$, we set the inputs $X_1, \ldots, X_n$ to be independent stochastic bit streams, each representing the probability t, and the inputs $Z_0, \ldots, Z_n$ to be independent stochastic bit streams with probabilities equal to the Bernstein coefficients $\beta_{0,n}, \ldots, \beta_{n,n}$, respectively.

The block labeled "+" in Figure 3.1 has n inputs $X_1, \ldots, X_n$ and $\lceil \log_2(n+1) \rceil$ outputs. It consists of combinational logic that computes the weight of the inputs, that is to say, it counts the number of ones in the n Boolean inputs $X_1, \ldots, X_n$, producing a *binary radix encoding* of this count. We will call this an n-bit Boolean "weight counter." The multiplexer (MUX) shown in the figure has "data" inputs $Z_0, \ldots, Z_n$ and the $\lceil \log_2(n+1) \rceil$ outputs of the weight counter as the selecting inputs. If the binary radix encoding of the outputs of the weight counter is k $(0 \leq k \leq n)$, then the output Y of the multiplexer is set to Z_k.

Figure 3.2 gives a simple design for an 8-bit Boolean weight counter based on a tree of adders. The eight inputs are grouped into 4 pairs and each pair is fed into a 1-bit adder, which gives a 2-bit sum as the output. The 4 sets of outputs of the 1-bit adders are further grouped into 2 pairs and each pair is fed into a 2-bit adder, which gives a 3-bit sum as the output. Finally, the pair of outputs of the 2-bit adders are fed into a 3-bit adder, which gives a 4-bit sum as the output, equal to the ones among the eight inputs. An n-bit Boolean weight counter can be implemented in a similar way.

In order to implement the Bernstein polynomial $B_n(t) = \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t)$, we set the inputs $X_1, \ldots, X_n$ to be independent stochastic bit streams representing variable

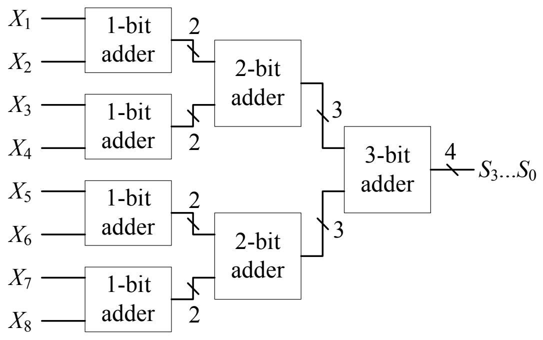


Figure 3.2: The implementation of an 8-bit Boolean weight counter.

probability t. Equivalently, $X_1, \ldots, X_n$ can be viewed as independent random Boolean variables that have the same probability t of being one. The probability that the count of the number of ones among the X_i's is k ($0 \leq k \leq n$) is given by the binomial distribution:

$$P\left(\sum_{i=1}^{n} X_i = k\right) = \binom{n}{k} t^k (1-t)^{n-k} = b_{k,n}(t). \tag{3.6}$$

We set the inputs $Z_0, \ldots, Z_n$ to be independent stochastic bit streams with probabilities equal to the Bernstein coefficients $\beta_{0,n}, \ldots, \beta_{n,n}$, respectively. Notice that we can represent $\beta_{i,n}$ with stochastic bit streams because we assume that $0 \leq \beta_{i,n} \leq 1$. Equivalently, we can view $Z_0, \ldots, Z_n$ as $n+1$ independent random Boolean variables that are one with probabilities $\beta_{0,n}, \ldots, \beta_{n,n}$, respectively.

The probability that the output Y is one is

$$y = P(Y=1) = \sum_{k=0}^{n} \left(P\left(Y = 1 | \sum_{i=1}^{n} X_i = k\right) P\left(\sum_{i=1}^{n} X_i = k\right)\right). \tag{3.7}$$

Since the multiplexer sets Y equal to Z_k, when $\sum_{i=1}^{n} X_i = k$, we have

$$P\left(Y = 1 | \sum_{i=1}^{n} X_i = k\right) = P(Z_k = 1) = \beta_{k,n}. \tag{3.8}$$

Thus, from Equations (3.5), (3.6), (3.7), and (3.8), we have

$$y = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) = B_n(t). \tag{3.9}$$

We conclude that the circuit in Figure 3.1 implements the given Bernstein polynomial with all coefficients in the unit interval. Since the given Bernstein polynomial is an arbitrary one in the set U. We have the following result.

Theorem 5

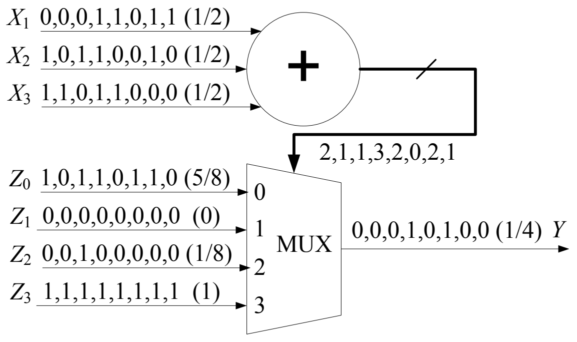
$$U \subseteq W. \qquad \square$$

Example 5

Figure 3.3 shows a circuit that implements the Bernstein polynomial

$$g(t) = \frac{5}{8} \cdot b_{0,3}(t) + 0 \cdot b_{1,3}(t) + \frac{1}{8} \cdot b_{2,3}(t) + 1 \cdot b_{3,3}(t),$$

converted from the power-form polynomial $g(t)$ in Example 1. The function is evaluated at $t = 1/2$. The stochastic bit streams X_1, X_2 and X_3 are independent, each representing the probability $t = 1/2$. The stochastic bit streams $Z_0, \ldots, Z_3$ represent the probability values $\frac{5}{8}$, 0, $\frac{1}{8}$, and 1, respectively. As expected, the computation produces the correct output value: $g(1/2) = 1/4$. $\square$



X_1 0,0,0,1,1,0,1,1 (1/2)
X_2 1,0,1,1,0,0,1,0 (1/2)
X_3 1,1,0,1,1,0,0,0 (1/2)
+
2,1,1,3,2,0,2,1
Z_0 1,0,1,1,0,1,1,0 (5/8)
Z_1 0,0,0,0,0,0,0,0 (0)
Z_2 0,0,1,0,0,0,0,0 (1/8)
Z_3 1,1,1,1,1,1,1,1 (1)
0
1
2
3
MUX
0,0,0,1,0,1,0,0 (1/4) Y

Figure 3.3: Computation on stochastic bit streams that implements the Bernstein polynomial $g(t) = \frac{5}{8} \cdot b_{0,3}(t) + 0 \cdot b_{1,3}(t) + \frac{1}{8} \cdot b_{2,3}(t) + 1 \cdot b_{3,3}(t)$ at $t = 1/2$. The stochastic streams X_1, X_2, and X_3 are independent, each with bits that have probability $t = 1/2$. The bits of the stochastic streams Z_0, Z_1, Z_2, and Z_3 have probabilities that correspond to the Bernstein coefficients.

3.4 Synthesizing Power-Form Polynomials

In real applications, we encounter polynomials in power form. There are two questions we are interested in:

1. What type of power-form polynomial is stochastically computable?
2. How do we implement this type of polynomial by logical computation on stochastic bit streams?

In this section, we give the answers to the above two questions.

Theorems 2, 4, and 5 establish relations among the sets U, V, and W: $V = U$, $W \subseteq V$, and $U \subseteq W$. Combining these three relations, we immediately get

Corollary 1

$$W = V. \qquad \square$$

The above corollary essentially answers the first question: a power-form polynomial g is stochastically computable if and only if g is a constant polynomial 0, a constant polynomial 1, or a polynomial that maps the open interval $(0, 1)$ into itself and maps the points 0 and 1 into the closed interval $[0, 1]$.

In order to implement a power-form polynomial g in the set V by logical computation on stochastic bit streams, we can first convert it into a Bernstein polynomial with all the coefficients in the unit interval. This is guaranteed by the relation $V = U$. However, we should note here that the degree of the equivalent Bernstein polynomial with all the coefficients in the unit interval may be greater than the degree of the original polynomial. Example 1 shows one instance. After we obtain the Bernstein polynomial with all the coefficients in the unit interval, we can implement it by the circuit construct described in Section 3.3.

In summary, given a power-form polynomial $g(t) = \sum_{i=0}^{n} a_{i,n} t^i$ that is in the set V, we can synthesize it by the following steps:

1. Let $m = n$. Obtain $\beta_{0,m}, \beta_{1,m}, \ldots, \beta_{m,m}$ from $a_{0,n}, a_{1,n}, \ldots, a_{n,n}$ by Equation (2.9).
2. Check to see if $0 \leq \beta_{i,m} \leq 1$, for all $i = 0, 1, \ldots, m$. If so, go to step 4.
3. Let $m = m+1$. Calculate $\beta_{0,m}, \beta_{1,m}, \ldots, \beta_{m,m}$ from $\beta_{0,m-1}, \beta_{1,m-1}, \ldots, \beta_{m-1,m-1}$ based on Equation (2.12). Go to step 2.
4. Implement the Bernstein polynomial
$$B_m(t) = \sum_{i=0}^{m} \beta_{i,m} b_{i,m}(t).$$
with the generalized multiplexing construct in Figure 3.1.

The synthesis flow is summarized in Figure 3.4.

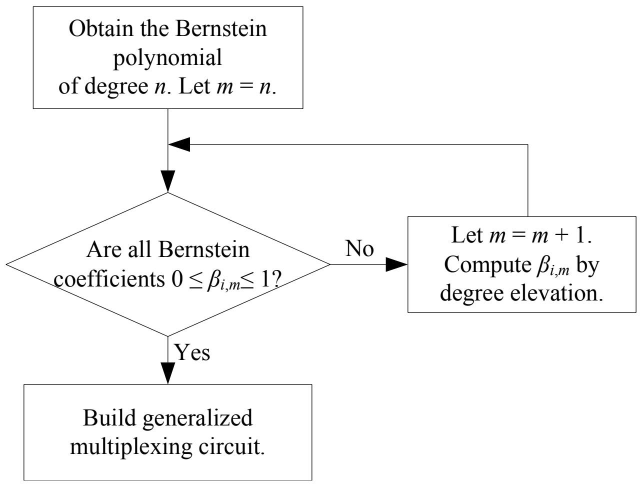

Obtain the Bernstein polynomial of degree n. Let $m = n$.
Are all Bernstein coefficients $0 \leq \beta_{i,m} \leq 1$?
No
Let $m = m + 1$. Compute $\beta_{i,m}$ by degree elevation.
Yes
Build generalized multiplexing circuit.

Figure 3.4: The synthesis flow to implement a power-form polynomial in the set V.

3.5 Synthesizing Non-Polynomial Functions

In real applications, of course, we often encounter non-polynomial functions, such as trigonometric functions. In this section, we present a method for synthesizing logical computation on stochastic bit streams that implements arbitrary functions. Our strategy is to approximate them by Bernstein polynomials with coefficients in the unit interval. We formulate the problem as follows:

Given $g(t)$, a continuous function on the unit interval, and n, the degree of a Bernstein polynomial, find real numbers $\beta_{i,n}$, $i = 0, \ldots, n$, that minimize

$$\int_0^1 (g(t) - \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t))^2 \, \mathrm{d}t, \tag{3.10}$$

subject to

$$0 \le \beta_{i,n} \le 1, \text{ for all } i = 0, 1, \ldots, n. \tag{3.11}$$

Here we try to find the optimal Bernstein polynomial approximation by minimizing an objective function, Equation (3.10), that measures the approximation error. This is the square of the L^2 norm on the difference between the original function $g(t)$ and the Bernstein polynomial $B_n(t) = \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t)$. The integral is on the unit interval because t, representing a probability value, is always in the unit interval. The constraints in Equation (3.11) guarantee that the Bernstein coefficients are all in the unit interval so that the function can be implemented by the generalized multiplexing construct in Figure 3.1.

If we expand (3.10), then an equivalent objective function is

$$f(\boldsymbol{\beta}) = \frac{1}{2} \boldsymbol{\beta}^T \boldsymbol{H} \boldsymbol{\beta} + \boldsymbol{c}^T \boldsymbol{\beta}, \tag{3.12}$$

where

$$\begin{aligned}
\boldsymbol{\beta} &= [\beta_{0,n}, \ldots, \beta_{n,n}]^T, \\
\boldsymbol{c} &= [-\int_0^1 g(t) b_{0,n}(t) \, \mathrm{d}t, \ldots, -\int_0^1 g(t) b_{n,n}(t) \, \mathrm{d}t]^T, \\
\boldsymbol{H} &= \begin{bmatrix} \int_0^1 b_{0,n}(t) b_{0,n}(t) \, \mathrm{d}t & \ldots & \int_0^1 b_{0,n}(t) b_{n,n}(t) \, \mathrm{d}t \\ \int_0^1 b_{1,n}(t) b_{0,n}(t) \, \mathrm{d}t & \ldots & \int_0^1 b_{1,n}(t) b_{n,n}(t) \, \mathrm{d}t \\ \vdots & \ddots & \vdots \\ \int_0^1 b_{n,n}(t) b_{0,n}(t) \, \mathrm{d}t & \ldots & \int_0^1 b_{n,n}(t) b_{n,n}(t) \, \mathrm{d}t \end{bmatrix}.
\end{aligned}$$

This optimization problem is, in fact, a constrained quadratic programming problem. Its solution can be obtained using standard techniques.

Example 6

Gamma Correction. *The gamma correction function is a nonlinear operation used to code and decode luminance and tri-stimulus values in video and still-image systems. It is defined by a power-law expression*

$$V_{out} = V_{in}^{\gamma},$$

where V_{in} is normalized between zero and one [12]. We apply a value of $\gamma = 0.45$, which is the value used in most TV cameras.

Consider the non-polynomial function

$$g(t) = t^{0.45}.$$

We approximate this function by a Bernstein polynomial of degree 6. By solving the constrained quadratic optimization problem, we obtain the Bernstein coefficients as:

$$\beta_{0,6} = 0.0955, \beta_{1,6} = 0.7207, \beta_{2,6} = 0.3476, \beta_{3,6} = 0.9988,$$
$$\beta_{4,6} = 0.7017, \beta_{5,6} = 0.9695, \beta_{6,6} = 0.9939. \qquad \square$$

In a strict mathematical sense, logical computation on stochastic bit streams can only implement functions that map the unit interval into the unit interval. However, with scaling, it can implement functions that map any *finite* interval into any *finite* interval. For example, the functions used in grayscale image processing are defined on the interval $[0, 255]$ with the same output range. If we want to implement such a function $y = g(t)$, we can instead implement the function $y = h(t) = \frac{1}{256}g(256t)$. Note that the new function $h(t)$ is defined in the unit interval and its output is also in the unit interval.

3.6 The Reconfigurable Architecture Based on Stochastic Computing

In this section, we present a **Re**configurable architecture based on **S**tochastic **C**omputing: the **ReSC architecture**. As illustrated in Figure 3.5, it is composed of three parts: the *Randomizer unit* generates stochastic bit streams; the *ReSC unit* processes these bit streams; and the *De-Randomizer unit* converts the resulting bit streams into output values. The architecture is reconfigurable in the sense that it can be used to compute different functions by setting appropriate values of the constant registers.

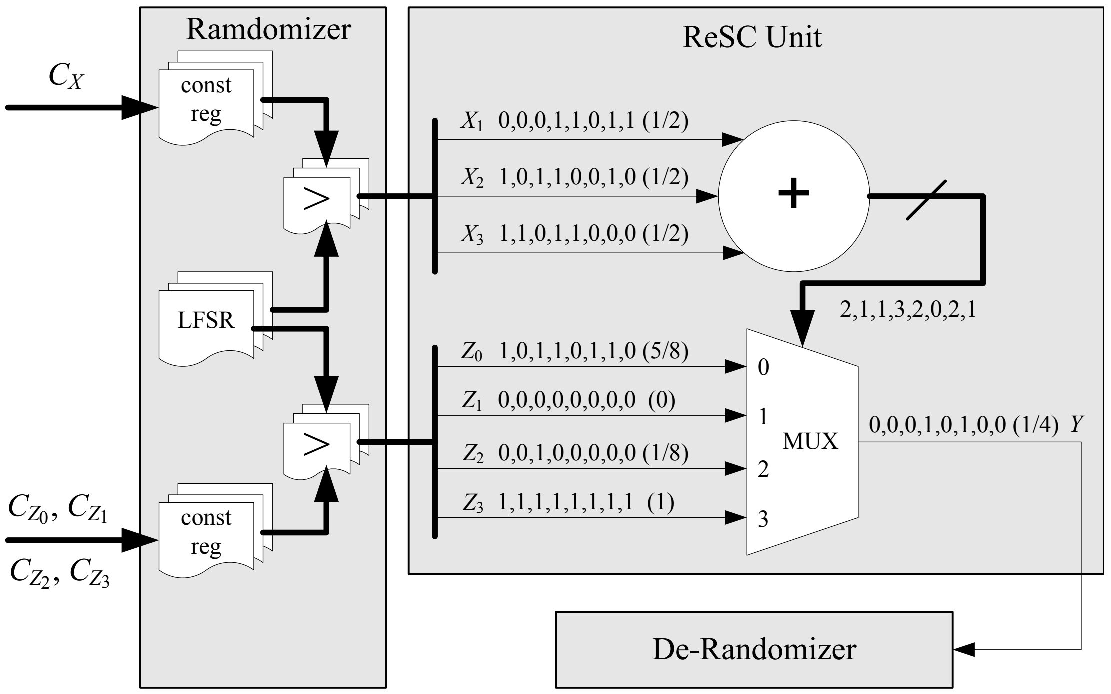
3.6.1 The ReSC Unit

The ReSC unit is the kernel of the architecture. It is the generalized multiplexing circuit described in Section 3.3, which implements a Bernstein polynomial with coefficients in the unit interval. As described in Section 3.5, we can use it to approximate arbitrary arithmetic functions.

The probability t of the independent stochastic bit streams X_i is controlled by the binary number C_X in a constant register, as illustrated in Figure 3.5. The constant register is a part of the Randomizer unit, discussed below. Similarly, stochastic bit streams $Z_0, \ldots, Z_n$ representing a specified set of coefficients can be produced by configuring the binary numbers C_{Z_i}'s in the constant registers.

3.6.2 The Randomizer Unit

The Randomizer unit is shown in Figure 3.6. It consists of a linear feedback shift register (LFSR), a constant number register, and a comparator. The LFSR produces a pseudorandom number R in each clock cycle. If R is strictly less than the number C stored in the constant number register, then the comparator generates a one; otherwise, it generates a zero.



Ramdomizer
ReSC Unit
C_X
const reg
LFSR
const reg
C_{Z_0}, C_{Z_1}
C_{Z_2}, C_{Z_3}
X_1 0,0,0,1,1,0,1,1 (1/2)
X_2 1,0,1,1,0,0,1,0 (1/2)
X_3 1,1,0,1,1,0,0,0 (1/2)
+
2,1,1,3,2,0,2,1
Z_0 1,0,1,1,0,1,1,0 (5/8)
Z_1 0,0,0,0,0,0,0,0 (0)
Z_2 0,0,1,0,0,0,0,0 (1/8)
Z_3 1,1,1,1,1,1,1,1 (1)
MUX
0,0,0,1,0,1,0,0 (1/4) Y
De-Randomizer

Figure 3.5: A reconfigurable architecture based on stochastic computing. Here the ReSC unit implements the target function $y = \frac{5}{8} - \frac{15}{8}t + \frac{9}{4}t^2$ at $t = 1/2$.

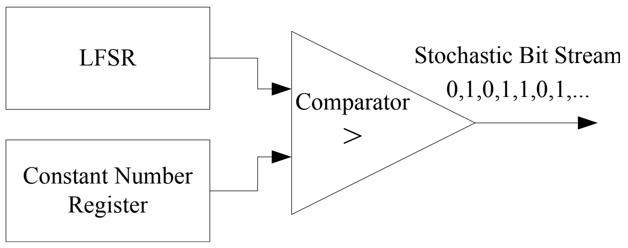


Figure 3.6: The Randomizer unit.

Assume that the LFSR has L bits. Accordingly, it generates repeating pseudorandom numbers with a period of $2^L - 1$. We choose L so that $2^L - 1 \geq N$, where N is the length of the input random bit streams. This guarantees good randomness of the input bit streams. The set of random numbers that can be generated by such an LFSR is $\{1, 2, \ldots, 2^L - 1\}$ and the probability that R equals a specific k in the set is

$$P(R = k) = \frac{1}{2^L - 1}. \tag{3.13}$$

Given a constant integer $1 \leq C \leq 2^L$, the comparator generates a one with probability

$$P(R < C) = \sum_{k=1}^{C-1} P(R = k) = \frac{C-1}{2^L - 1}. \tag{3.14}$$

Thus, the set of probability values that can be generated by the Randomizer unit is

$$S = \{0, \frac{1}{2^L - 1}, \ldots, 1\}. \tag{3.15}$$

Given an arbitrary value $0 \leq p \leq 1$, we round it to the closest number p' in S. Hence, C is determined by p as

$$C = \text{round}(p(2^L - 1)) + 1, \tag{3.16}$$

where the function round(x) gives the nearest integer to x. The value p is quantized to

$$p^* = \frac{\text{round}(p(2^L - 1))}{2^L - 1} \tag{3.17}$$

In our stochastic implementation, we require different input random bit streams to be independent. Therefore, LFSRs for generating different input random bit streams are configured to have different feedback functions.

3.6.3 The De-Randomizer Unit

The De-Randomizer unit translates the result of the stochastic computation, expressed as a stochastic bit stream, back to a deterministic value using a counter. We set the length of the stochastic bit stream to be a power of 2, i.e., $N = 2^M$, where M is an integer. We choose the number of bits of the counter to be $M + 1$, so that we can count all possible numbers of ones in the stream: from 0 to 2^M. We treat the output of the counter as a binary decimal number $d = (c_M.c_{M-1} \ldots c_0)_2$, where $c_0, c_1, \ldots, c_M$ are the $M + 1$ output bits of the counter.

Let the N bits of the output stochastic bit stream Y be $Y(1), \ldots, Y(N)$. Suppose that each bit has probability y of being one. Then the mean value of the counter result d is

$$E[d] = E\left[\frac{(c_M \ldots c_0)_2}{2^M}\right] = E\left[\frac{1}{N}\sum_{\tau=1}^{N} Y(\tau)\right] = \frac{1}{N}\sum_{\tau=1}^{N} E[Y(\tau)] = y, \tag{3.18}$$

which is the value represented by the stochastic bit stream Y.

Compared to the kernel, the Randomizer and De-Randomizer units are expensive in terms of the hardware resources required. Indeed, they dominate the area cost of the architecture. We note that in many applications that involve analog to digital (A/D) and digital to analog (D/A) conversion, a part of the A/D and D/A converters function as the Randomizer and De-Randomizer units. For instance in sensors and embedded systems, the inputs are obtained from physical measurements in analog form, so as real-valued numbers. In the analog to digital conversion process, these real-valued numbers are converted to binary radix. However, many A/D converters, such as sigma-delta converters, naturally produce pulse streams of ones and zeros as an intermediate form [13]. Such converters could easily be adapted to produce stochastic bit streams, exploiting white noise for the encoding. This could potentially be much less costly than a full conversion to binary radix, as that entails thresholding and sampling. Similarly, at the outputs, digital to analog conversion could produce analog values directly from stochastic bit streams.

3.7 Error Analysis of the ReSC Architecture

By its very nature, logical computation on stochastic bit streams introduces uncertainty into the computation. There are three sources of errors.

1. **The error due to the Bernstein approximation**: Since we use a Bernstein polynomial with coefficients in the unit interval to approximate a function $g(t)$, there is an approximation error

   $$e_1 = \left| g(t) - \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t) \right|. \tag{3.19}$$

   We could use the L^2-norm to measure the average error as

   $$\begin{aligned} e_{1\text{avg}} &= \left(\frac{1}{1-0} \int_0^1 (g(t) - \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t))^2 \,\mathrm{d}t \right)^{0.5} \\ &= \left(\int_0^1 (g(t) - \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t))^2 \,\mathrm{d}t \right)^{0.5} \end{aligned} \tag{3.20}$$

   The average approximation error e_{1avg} only depends on the original function $g(t)$ and the degree of the Bernstein polynomial; e_{1avg} decreases as n increases. For all of the functions that we tested, a Bernstein approximation of degree of 6 was sufficient to reduce e_{1avg} to be below 10^{-3}.

2. **The quantization error**:

   As shown in Section 3.6.2, given an arbitrary value $0 \leq p \leq 1$, we round it to the closest number p^* in $S = \{0, \frac{1}{2^L-1}, \ldots, 1\}$ and generate the corresponding bit stream. Thus, the quantization error for p is

   $$|p - p^*| \leq \frac{1}{2(2^L - 1)}, \tag{3.21}$$

   where L is the number of bits of the LFSR.

Due to the effect of quantization, we will compute $\sum_{i=0}^{n} \beta_{i,n}^* b_{i,n}(t^*)$ instead of the Bernstein approximation $\sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t)$, where $\beta_{i,n}^*$ and t^* are the closest numbers to $\beta_{i,n}$ and t, respectively, in the set S. Thus, the quantization error is

$$e_2 = \left| \sum_{i=0}^{n} \beta_{i,n}^* b_{i,n}(t^*) - \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t) \right| \tag{3.22}$$

Define $\Delta\beta_{i,n} = \beta_{i,n}^* - \beta_{i,n}$ and $\Delta t = t^* - t$. Then, using a first order approximation, the error due to quantization is

$$\begin{aligned} e_2 &\approx \left| \sum_{i=0}^{n} b_{i,n}(t)\Delta\beta_{i,n} + \sum_{i=0}^{n} \beta_{i,n} \frac{\mathrm{d}b_{i,n}(t)}{\mathrm{d}t} \Delta t \right| \\ &= \left| \sum_{i=0}^{n} b_{i,n}(t)\Delta\beta_{i,n} + n \sum_{i=0}^{n-1} (\beta_{i+1,n} - \beta_{i,n}) b_{i,n-1}(t) \Delta t \right| \end{aligned}$$

Notice that since $0 \le \beta_{i,n} \le 1$, we have $|\beta_{i+1,n} - \beta_{i,n}| \le 1$. Combining this with the fact that $\sum_{i=0}^{n} b_{i,n}(t) = 1$ and $|\Delta\beta_{i,n}|, |\Delta t| \le \frac{1}{2(2^L-1)}$, we have

$$e_2 \le \frac{1}{2(2^L-1)} \left| \sum_{i=0}^{n} b_{i,n}(t) \right| + \frac{n}{2(2^L-1)} \left| \sum_{i=0}^{n-1} b_{i,n-1}(t) \right| = \frac{n+1}{2(2^L-1)}. \tag{3.23}$$

Thus, the quantization error is inversely proportional to 2^L. We can mitigate this error by increasing the number of bits L of the LFSR.

3. **The error due to random fluctuations**: Due to the Bernstein approximation and the quantization effect, the probability of each bit in the output bit stream $Y(\tau)$ $(\tau = 1, 2, \ldots, N)$ to be one is $p^* = \sum_{i=0}^{n} \beta_{i,n}^* b_{i,n}(t^*)$. The De-Randomizer unit returns the result

$$y = \frac{1}{N} \sum_{\tau=1}^{N} Y(\tau). \tag{3.24}$$

It is easily seen that $E[y] = p^*$. However, the realization of y is not, in general, exactly equal to p^*. The error can be measured by the variation as

$$\begin{aligned} Var[y] &= Var[\frac{1}{N} \sum_{\tau=1}^{N} Y(\tau)] = \frac{1}{N^2} \sum_{\tau=1}^{N} Var[Y(\tau)] \\ &= \frac{p^*(1-p^*)}{N}. \end{aligned} \tag{3.25}$$

Since $Var[y] = E[(y - E[y])^2] = E[(y - p^*)^2]$, the error due to random fluctuations is

$$e_3 = |y - p^*| \approx \sqrt{\frac{p^*(1-p^*)}{N}}. \tag{3.26}$$

Thus, the error due to random fluctuations is inversely proportional to $\sqrt{N}$. Increasing the length of the bit stream will decrease the error.

The overall error is bounded by the sum of the above three error components:

$$\begin{aligned} e = |g(t) - y| &\le \left| g(t) - \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t) \right| \\ &+ \left| \sum_{i=0}^{n} \beta_{i,n} b_{i,n}(t) - \sum_{i=0}^{n} \beta^*_{i,n} b_{i,n}(t^*) \right| + \left| \sum_{i=0}^{n} \beta^*_{i,n} b_{i,n}(t^*) - y \right| \\ &= e_1 + e_2 + e_3. \end{aligned} \tag{3.27}$$

Note that we choose the number of bits L of the LFSRs to satisfy $2^L - 1 \ge N$ in order to get non-repeating random bit streams. Therefore, we have

$$\frac{1}{2^L} < \frac{1}{N} \ll \frac{1}{\sqrt{N}}$$

Combining the above equation with Equations (3.23) and (3.26), we can see that in our implementation, the error due to random fluctuations will dominate the quantization error. Therefore, the overall error e is approximately bounded by the sum of the error e_1 and e_3, i.e.,

$$e \le e_1 + e_3.$$

We use the gamma correction function $g(t) = t^{0.45}$ introduced in Example 6 to demonstrate the three error components described above.

1. **The error due to the Bernstein approximation.** Figure 3.7 plots the error due to the Bernstein approximation versus the degree of the approximation. The error is measured by Equation (3.20). It shows that the error decreases as the degree of the Bernstein approximation increases. For a choice of degree $n = 6$, the error is approximately $4 \cdot 10^{-3}$.

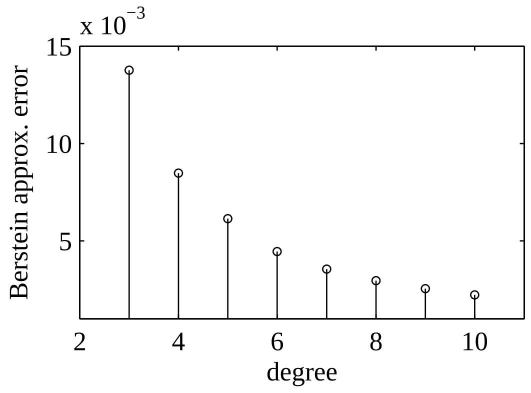


Figure 3.7: The Bernstein approximation error versus the degree of the Bernstein approximation.

2. **The quantization error.** Figure 3.8 plots the quantization error versus the number of bits L of the LFSR. In the figure, the x-axis is $1/2^L$, where the range of L is from 5 to 11. For different values of L, $\beta^*_{i,n}$ and t^* in Equation (3.22) change according to Equation (3.17). The quantization error is measured by Equation (3.22) with the Bernstein polynomial chosen as the degree 6 Bernstein polynomial approximation of the gamma correction function. For each value of L, we evaluate the quantization error on 11 sample points $x = 0, 0.1, \ldots, 0.9, 1$. The mean, the mean plus the standard deviation, and the mean minus the standard deviation of the errors are plotted by a circle, a downward-pointing triangle, and a upward-pointing triangle, respectively.

   Clearly, the means of the quantization error are located near a line, which means that the quantization error is inversely proportional to 2^L. Increasing L will decrease the quantization error.

3. **The error due to random fluctuations.** Figure 3.9 plots the error due to random fluctuations versus the length N of the stochastic bit stream. In the figure, the x-axis is $1/\sqrt{N}$, where N is chosen to be 2^m, with $m = 7, 8, \ldots, 13$. The error is measured as the average of 50 Monte Carlo simulations of the difference between

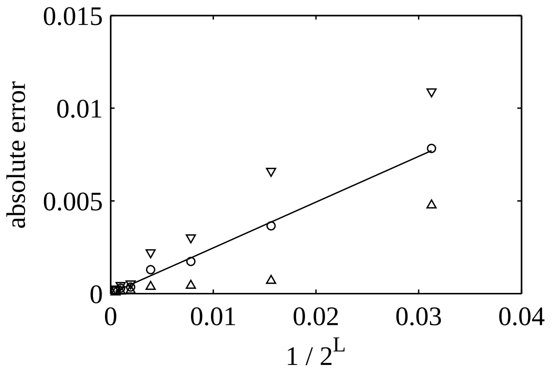


Figure 3.8: The quantization error versus $1/2^L$, where L is the number of bits of the LFSR. The circles, the downward-pointing triangles, and the upward-pointing triangles represent the means, the means plus the standard deviations, and the means minus the standard deviations of the errors on the sample points $x = 0, 0.1, \ldots, 0.9, 1$, respectively.

the stochastic computation result and the quantized implementation of the degree 6 Bernstein polynomial approximation of the gamma correction function. To add the quantization effect, we choose an LFSR of 10 bits. For each N, we evaluate the error on 11 sample points $x = 0, 0.1, \ldots, 0.9, 1$. The mean, the mean plus the standard deviation, and the mean minus the standard deviation of the errors are plotted by a circle, a downward-pointing triangle, and a upward-pointing triangle, respectively.

The figure clearly shows that the means of the error due to random fluctuations are located near a straight line. Thus, it confirms the fact that the error due to random fluctuations is inversely proportional to $\sqrt{N}$. The error component could be decreased by increasing the length of the stochastic bit stream.

3.8 Experimental Results

In this section, we present experimental results. We performed two sets of experiments: the first set of experiments was on synthesizing polynomials and the second set on synthesizing some common functions encountered in image processing. In both sets

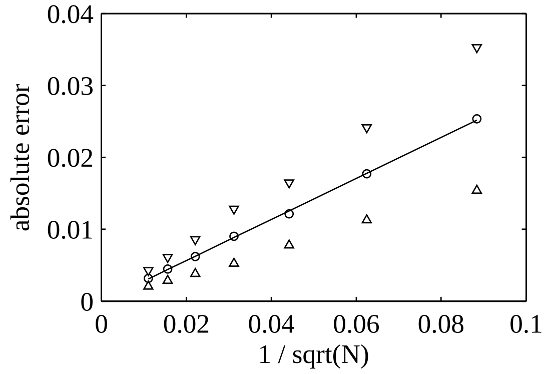


Figure 3.9: The error due to random fluctuations versus $1/\sqrt{N}$, where N is the length of the stochastic bit stream. The circles, the downward-pointing triangles, and the upward-pointing triangles represent the means, the means plus the standard deviations, and the means minus the standard deviations of the errors on sample points $x = 0, 0.1, \ldots, 0.9, 1$, respectively.

of experiments, we first compared the hardware cost of the conventional implementations to that of the stochastic implementations. Then, we compared the performance of these two implementations on noisy input data.

The stochastic implementations (ReSC architecture) were written in Verilog, and then synthesized, placed, and routed using Xilinx ISE 9.1.03i. A Xilinx Virtex-II Pro XC2VP30-7-FF896 FPGA was chosen as the FPGA platform. The coefficients of the ReSC unit for each test module were obtained from Matlab code. The conventional implementations of the same test modules were manually translated into Verilog codes and synthesized on the same FPGA platform.

3.8.1 Hardware Comparison for Implementing Polynomials

For a conventional implementation of a polynomial, we assume that the input consists of M bits, encoding values in binary radix. The resolution of the computation in binary radix is 2^{-M}. A polynomial

$$g(t) = \sum_{i=0}^{n} a_{i,n} t^i$$

can be factorized as

$$g(t) = a_{0,n} + t(a_{1,n} + t(a_{2,n} + \cdots + t(a_{n-1,n} + ta_{n,n}))).$$

With such a factorization, we can evaluate the polynomial in n iterations. Each iteration consists of an addition and a multiplication. Hence, for such an iterative calculation, the hardware consists of an adder and a multiplier.

Two different designs were applied for the conventional implementation. The first one is a combinational implementation: n adders and n multipliers are used to compute the polynomial in one cycle. The second one is a sequential implementation. The circuit is composed of an adder and a multiplier. All the coefficients of the polynomial are sequentially fed into the circuit and the result is registered for the next stage. We need n cycles to get the result when using the sequential implementation.

We built two types of stochastic implementations. The "Core" type implementation only includes the ReSC unit. The "Full" type implementation includes the Randomizer unit, the ReSC unit, and the De-Randomizer unit. In order to get the same resolution as the conventional implementation, the length of the stochastic bit streams should be $N = 2^M$. Therefore, we need 2^M *cycles* to get the result when using a serial stochastic implementation.

In Table 3.2, we compare the area of the conventional implementations to that of the stochastic implementations for polynomial degree $n = 3, 4, 5, 6$ and $M = 7, 8, 9, 10, 12$. The area is measured in terms of the number of look-up tables (LUTs) used in FPGA. The ratio columns of the table show the ratios of the area of the conventional sequential implementation and the area of the stochastic implementation to that of the conventional combinational implementation. The stochastic "Core" type implementation has on average a 97.2% reduction of hardware usage compared to the conventional combinational implementation. The stochastic "Full" type implementation, which further includes the Randomizer unit and the De-Randomizer unit, needs 14 times hardware usage than the "Core" type implementation. However, it still has on average a 60%

reduction of hardware cost compared with the conventional combinational implementation, and is comparable to the conventional sequential implementation.

3.8.2 Comparison of Performance on Noisy Input Data for Implementing Polynomials

We compared the performance of conventional versus stochastic implementations of polynomials when the input data is corrupted with noise. Suppose that the input data of a conventional implementation has $M = 10$ bits. In order to have the same resolution, the bit stream of a stochastic implementation should have $2^M = 1024$ bits. We chose the error ratio ϵ of the input data to be 0, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, and 0.1, as measured by the fraction of random bit flips that occur.

To measure the impact of the noise, we performed two sets of experiments. In the first, we chose the 6-th order Maclaurin polynomial approximation of 11 elementary functions as our implementation target. We list these 11 functions in Table 3.3, together with the degree of their 6-th order Maclaurin polynomials. Such Maclaurin approximations are commonly used in numerical evaluation of non-polynomial functions.

All of these Maclaurin polynomials evaluate to non-negative values for $0 \leq t \leq 1$. However, for some of these, the maximal evaluation on $[0, 1]$ is greater than 1. Thus, we scaled these polynomials by the reciprocal of their maximal value; this is a necessary condition for a stochastic implementation. The scaling factors that we used are listed in Table 3.3.

We evaluated each Maclaurin polynomial on 13 points: 0.2, 0.25, 0.3, ..., 0.8. For each error ratio ϵ, each Maclaurin polynomial, and each evaluation point, we simulated both the stochastic and the conventional implementations 1000 times. We averaged the relative errors over all simulations. Finally, for each error ratio ϵ, we averaged the relative errors over all polynomials and all evaluation points.

Table 3.2: Comparison of the area (in terms of LUTs) of the conventional implementation to that of the stochastic implementation of polynomials with different degrees and resolutions.

Degree	Length	Conventional			Stochastic			
n	M	Combinational	Sequential		Core		Full	
		Area	Area	Ratio (%)	Area	Ratio (%)	Area	Ratio (%)
		α	β	$100 \cdot \beta/\alpha$	γ	$100 \cdot \gamma/\alpha$	δ	$100 \cdot \delta/\alpha$
3	7	110	69	62.7%	4	3.6%	62	56.4%
3	8	137	81	59.1%		2.9%	69	50.4%
3	9	167	94	56.3%		2.4%	76	45.5%
3	10	200	108	54.0%		2.0%	83	41.5%
3	11	236	123	52.1%		1.7%	93	39.4%
3	12	275	139	50.5%		1.5%	93	33.8%
4	7	146	77	52.7%	6	4.1%	73	50.0%
4	8	180	90	50.0%		3.3%	81	45.0%
4	9	220	104	47.3%		2.7%	88	40.0%
4	10	265	119	44.9%		2.3%	97	36.6%
4	11	312	135	43.3%		1.9%	111	35.6%
4	12	370	152	41.1%		1.6%	110	29.7%
5	7	182	89	48.9%	8	4.4%	92	50.5%
5	8	225	99	44.0%		3.6%	101	44.9%
5	9	275	115	41.8%		2.9%	109	39.6%
5	10	331	131	39.6%		2.4%	117	35.3%
5	11	390	148	37.9%		2.1%	127	32.6%
5	12	461	166	36.0%		1.7%	134	29.1%
6	7	219	85	38.8%	11	5.0%	105	47.9%
6	8	275	99	36.0%		4.0%	118	42.9%
6	9	333	114	34.2%		3.3%	126	37.8%
6	10	396	131	33.1%		2.8%	136	34.3%
6	11	471	148	31.4%		2.3%	154	32.7%
6	12	556	166	29.9%		2.0%	158	28.4%
Average				44.4%		2.8%		40.0%

Table 3.3: Sixth-order Maclaurin polynomial approximation of elementary functions.

function	degree of Maclaurin polynomials	scaling factor
$\sin(x)$	5	1
$\tan(x)$	5	0.6818
$\arcsin(x)$	5	0.8054
$\arctan(x)$	5	1
$\sinh(x)$	5	0.8511
$\tanh(x)$	5	1
$\mathrm{arcsinh}(x)$	5	1
$\cos(x)$	6	1
$\cosh(x)$	6	0.6481
$\exp(x)$	6	0.3679
$\ln(x+1)$	6	1

In the second set of experiments, we randomly chose 100 Bernstein polynomials of degree 6 with coefficients in the unit interval. With this specification, we were guaranteed that the polynomials can be implemented stochastically. We evaluated each on 10 points: $0, 1/9, 2/9, \ldots, 1$. We compiled similar statistics to that in the first set of experiments.

Table 3.4 shows the average relative error of the stochastic implementation and the conventional implementation versus different error ratios ϵ for both sets of experiments. We plot the data for the experiments on Maclaurin polynomials in Figure 3.10 to give a clear comparison.

When $\epsilon = 0$, meaning that no noise is injected into the input data, the conventional implementation computes without any error. However, due to the inherent variance, the stochastic implementation produces a small relative error. However, with noise, the relative error of the conventional implementation blows up dramatically as ϵ increases. Even for small values, the stochastic implementation performs much better.

Table 3.4: Relative error for the stochastic and conventional implementations of polynomial computation versus the error ratio ϵ in the input data.

	Maclaurin polynomial		Randomly chosen polynomials	
error ratio ϵ	relative error of stochastic (%)	relative error of conventional (%)	relative error of stochastic (%)	relative error of conventional (%)
0.0	2.63	0.00	2.92	0.00
0.001	2.62	0.68	3.06	11.1
0.002	2.64	1.41	3.27	21.3
0.005	2.73	3.36	4.25	53.9
0.01	3.01	6.75	6.05	106
0.02	3.89	12.8	9.93	208
0.05	7.54	28.9	21.4	494
0.1	13.8	51.2	39.2	948

This demonstrates that the stochastic representation is much more tolerant of noise than the conventional binary radix representation.

In the data presented in Table 3.4, note that the relative error of the randomly chosen polynomials computed by the conventional implementation is much larger than that of the Maclaurin polynomials computed by the conventional implementation. The explanation for this is that the randomly chosen polynomials have much larger power-form coefficients. Bit flips on these coefficients dramatically change their values.

3.8.3 Hardware Comparison for Implementing Image Processing Applications

We demonstrated the effectiveness of our method on a collection of image processing benchmarks. We chose ten test cases [14, 15, 16]. These can be classified into three categories: `Gamma`, `RGB`$\rightarrow$`XYZ`, `XYZ`$\rightarrow$`RGB`, `XYZ`$\rightarrow$`CIE-L*ab`, and `CIE-L*ab`$\rightarrow$`XYZ` are popular color-space converter functions in image processing; `Geometric` and `Rotation` are

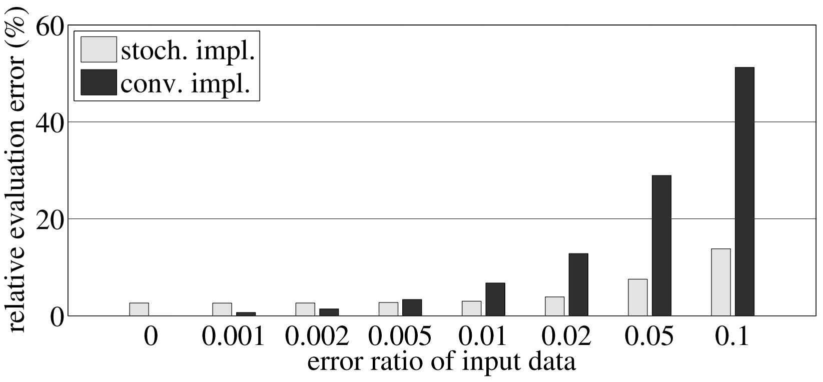


Figure 3.10: A plot of the relative error for the stochastic and the conventional implementations of Maclaurin polynomial computation versus the error ratio ϵ in the input data.

geometric models for processing two-dimensional figures; and `Example01` to `Example03` are operations used to generate 3D image data sets.

Table 3.5 compares the hardware usage of the conventional implementations to that of the stochastic implementations. Similar to the experiments on implementing polynomials, we built two types of stochastic implementations: the "Core" and the "Full" type implementations. On average, the stochastic "Core" type implementations achieve a 89% reduction of LUT usage. If the peripheral Randomizer unit and De-Randomizer unit are included, then the stochastic implementation achieves a 40% reduction of hardware usage.

3.8.4 Comparison of Performance on Noisy Input Data for Implementing Image Processing Applications

In this section, we compared the performance of the conventional implementation to that of the stochastic implementation on noisy input data. We performed experiments injecting soft errors. This consists of flipping a given percentage of the input bits of the circuit and evaluating the output. For example, if 2% noise is injected, this implies that 2% of the total number of input bits of the circuit are chosen randomly and flipped.

Table 3.5: Comparison of the hardware usage (in terms of LUTs) of the conventional implementation to that of the stochastic implementation.

Module	Conventional Cost	Stochastic			
		Core		Full	
		Cost	Ratio (%)	Cost	Ratio (%)
	α	β	$100 \cdot \beta/\alpha$	γ	$100 \cdot \gamma/\alpha$
Gamma	96	16	16.7	124	129.2
RGB→XYZ	524	64	12.2	301	57.4
XYZ→RGB	627	66	10.5	301	48.0
XYZ→CIE-L*ab	295	58	19.7	250	84.7
CIE-L*ab→XYZ	554	54	9.7	258	46.6
Geometric	831	32	3.9	299	36.0
Rotation	737	30	4.1	257	34.9
Example01	474	46	9.7	378	79.7
Example02	1065	109	10.2	378	35.5
Example03	702	89	12.7	318	45.3
Average	590	56	10.9	286	59.7

We evaluated the output in terms of the average error in pixel values. Table 3.6 shows the results for three different injected noise ratios for both the stochastic implementations and the conventional implementations of the color-space converter functions. The average output error of the conventional implementation is about twice that of the stochastic implementation.

The stochastic approach produces dramatic results when the magnitude of the errors is analyzed. In Table 3.7, we list the percentages of output pixels that have errors greater than 20% for three different input noise ratios. With a 10% soft error injection rate, the conventional approach produces outputs that are more than 20% off over 37% the time. In contrast, the stochastic implementation *never* produces pixel values with errors greater than 20%. Figure 1.3 visually shows what a difference this makes.

Table 3.6: The average output error of the stochastic implementations compared to the conventional implementations of the color-space converter functions.

Module	Injected Error					
	1%		2%		10%	
	Conv.	Stoch.	Conv.	Stoch.	Conv.	Stoch.
`Gamma`	0.7	0.9	1.5	1.6	6.8	7.5
`RGB`$\longrightarrow$`XYZ`	2.7	0.8	5.3	1.4	22.4	6.2
`XYZ`$\longrightarrow$`RGB`	3.2	1.2	5.9	2.3	21.6	8.2
`XYZ`$\longrightarrow$`CIE-L*ab`	2.1	0.8	3.4	1.4	11.7	7.3
`CIE-L*ab`$\longrightarrow$`XYZ`	0.6	0.8	1.2	1.5	7.4	7.3
Average	2.2	0.9	4.0	1.7	15.8	7.3

Table 3.7: The percentage of pixels with errors greater than 20% for the conventional implementations and the stochastic implementations of the color-space converter functions.

Module	Conventional Injected Error			Stochastic Injected Error
	1%	2%	10%	1%, 2%, 10%
`Gamma`	1.4	3.8	13.4	0.0
`RGB`$\longrightarrow$`XYZ`	2.2	4.4	20.7	0.0
`XYZ`$\longrightarrow$`RGB`	11.7	20.0	63.8	0.0
`XYZ`$\longrightarrow$`CIE-L*ab`	6.1	11.6	43.7	0.0
`CIE-L*ab`$\longrightarrow$`XYZ`	2.0	4.0	20.7	0.0
Average	5.0	10.0	37.2	0.0

3.9 Related Work

A sequence of early papers established the concept of logical computation on stochastic bit streams [17, 18]. These papers discussed basic operations such as multiplication and addition. Later papers delved into more complex operations, including exponential functions and square roots [19, 20]. In [21], the authors discuss the implementation of basic arithmetic operations as well as complex ones, including hyperbolic functions, with stochastic bit streams. They also discuss different forms of stochastic representation, including a "bipolar" representation for negative values. Much of the interest in computing with stochastic bit streams stems from the field of neural networks, where the concept is known as "pulsed" or "pulse-coded" computation [22, 23].

In fact, the general concept of stochastic computing dates back even earlier, to work by J. von Neumann in the 1950's [24]. He applied probabilistic logic to the study of thresholding and multiplexing operations on bundles of wires with stochastic signals. As he eloquently states in the introduction to his seminal paper, "Error is viewed not as an extraneous and misdirected or misdirecting accident, but as an essential part of the [design]." We find this view, that randomness and noise are integral to computation, to be compelling in the modern era of nanoscale electronics.

We point to two recent research efforts that embrace randomness in circuit and system design. In [6], the authors propose a construct that they call probabilistic CMOS (PCMOS) that generates random bits from intrinsic sources of noise. In [7], PCMOS switches are applied to form a probabilistic system-on-a-chip (PSOC); this system provides intrinsic randomness to the application layer, so that it can be exploited by probabilistic algorithms. In [25] and [26], the authors propose a methodology for designing stochastic processors, that is to say, processors that can tolerate computational errors caused by hardware uncertainties. They strive for a favorable trade-off between reliability and power consumption.

Chapter 4

Synthesizing Combinational Logic to Generate Probabilities

A premise for logical computation on stochastic bit streams is the availability of random bit streams with the requisite probabilities. Such streams can either be generated from physical random sources or with pseudo-random constructs such as LFSR. We have shown how to generate stochastic bit streams from LFSR in Section 3.6.2. Figure 4.1 illustrates the general process. In each clock cycle, a random source generates a value R obeying a certain probability density function $f(R)$. A comparator compares the value R with a constant value C: it outputs a one if $R < C$ and a zero otherwise. The output of the comparator is a stream of random bits that have probability

$$p = \int_{-\infty}^{C} f(R)\,\mathrm{d}R \tag{4.1}$$

of being one.

Generating stochastic bit streams entails significant cost in terms of hardware resources. If the system employs pseudo-random number generators such as LFSRs, most of the cost is incurred in the pseudo-random source itself. The constant value can be generated relatively cheaply using a simple register [27].

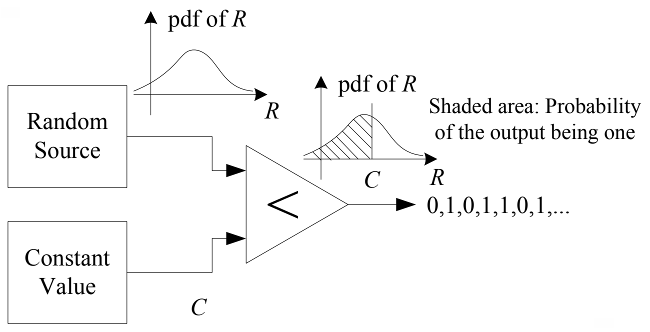


Figure 4.1: Generating stochastic bit streams from random or pseudo-random sources.

If the system exploits a physical mechanism, the random source may be cheap but the constant value may be expensive to implement. For example, in [6], the authors describe a scheme for exploiting the intrinsic thermal noise of nanoscale CMOS devices as the random source. This is inexpensive to do. However, in their approach, the constant value C corresponds to a supply voltage. Providing different supply voltages is comparatively expensive. If the application requires many stochastic bit streams with different probabilities, many constant values are required. The cost of generating these directly might be prohibitive.

In this chapter, we present a synthesis strategy to mitigate this issue: we describe a method for synthesizing combinational logic to transform a set of stochastic bit streams representing a limited number of probabilities into stochastic bit streams representing other target probabilities.

4.1 Description of the Problem

It is convenient to treat stochastic bit streams mathematically as random Boolean variables. For what follows, we consider combinational logic that has random Boolean variables as inputs. When we say “a probability,” we mean the probability of a random Boolean variable being one. When we say “a circuit,” we mean a combinational circuit built with logic gates.

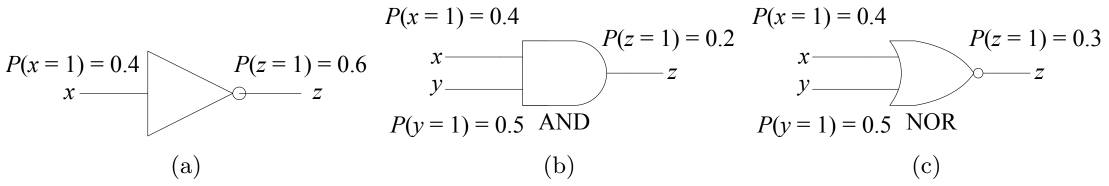


Figure 4.2: An illustration of transforming a set of source probabilities into new probabilities with logic gates. (a): An inverter implementing $P(Z = 1) = 1 - P(X = 1)$. (b): An AND gate implementing $P(Z = 1) = P(X = 1) \cdot P(Y = 1)$. (c): A NOR gate implementing $P(Z = 1) = (1 - P(X = 1)) \cdot (1 - P(Y = 1))$.

Example 7

Suppose that we have a set of source probabilities $S = \{0.4, 0.5\}$. As illustrated in Figure 4.2, we can transform this set into new probabilities:

1. *Given an input x with probability 0.4, an inverter will have an output z with probability 0.6 since*

$$P(z = 1) = P(x = 0) = 1 - P(x = 1). \tag{4.2}$$

2. *Given inputs x and y with independent probabilities 0.4 and 0.5, an AND gate will have an output z with probability 0.2 since*

$$P(z = 1) = P(x = 1, y = 1) = P(x = 1)P(y = 1). \tag{4.3}$$

3. *Given inputs x and y with independent probabilities 0.4 and 0.5, a NOR gate will have an output z with probability 0.3 since*

$$\begin{aligned} P(z = 1) &= P(x = 0, y = 0) = P(x = 0)P(y = 0) \\ &= (1 - P(x = 1))(1 - P(y = 1)). \end{aligned}$$

Thus, using combinational logic, we obtain the set of probabilities $\{0.2, 0.3, 0.6\}$ from the set $\{0.4, 0.5\}$. $\Box$

Motivated by this example, we consider the problem of how to synthesize combinational logic to transform a set of source probabilities $S = \{p_1, p_2, \ldots, p_n\}$ into a target probability q. We assume that the probabilistic sources are all independent. We consider three scenarios:

1. **Scenario One:**

   Suppose that we are generating stochastic bit streams from physical random sources and that we have the flexibility to construct a number of constant value generators. This gives us the freedom to choose a set of source probabilities S. The cost of the random sources is negligible but the cost of generating the constant values for the comparators is considerable. Accordingly, we seek to minimize the cardinality of the set S. Note that we can produce multiple independent copies of each source probability in S cheaply, since each copy uses the same constant value. Thus, we assume that each probability in the set S can be used an arbitrary number of times. (We say that the probability can be *duplicated*.) The problem is to find a small set S and to demonstrate how to synthesize logic that transforms the values from this set into an arbitrary target probability q.

2. **Scenario Two:**

   Suppose that we are given a collection of stochastic bit stream generators that produce a fixed set S of source probabilities. We cannot adjust the probabilities in S nor can we duplicate them; each source probability can only be used once. (Although we cannot duplicate the values, the set S can be a *multiset*, i.e., one that could contain multiple elements of the same value.) The problem is how to synthesize logic that has input probabilities taken from this fixed set S and produces an output probability q.

3. **Scenario Three:**

   Suppose that we are generating stochastic bit streams with pseudo-random constructs such as LFSRs and we have full freedom to design the system. In this case, the cost of building the random sources is considerable. Suppose that we establish a budget of n random sources. Thus, the number of input stochastic bit streams is limited to n. Since it is relatively cheap to generate different constant values, we are able to choose n arbitrary source probabilities. The problem is to find a set S of n probabilities such that we can synthesize logic that transforms values from this set into an arbitrary probability q. Again, the set S can be a multiset. Since each probability in the source set S corresponds to an individual random source, each element of the set S can be used as an input probability at most once.

To summarize, we consider scenarios that differ in respect to:

1. Whether the set S is specified or not.
2. Whether the probabilities from S can be duplicated or not.

Specifically, in Scenario One, the set S is not specified and the probabilities from S can be duplicated. In Scenario Two, the set S is specified and the probabilities from S cannot be duplicated. In Scenario Three, the set S is not specified and the probabilities from S cannot be duplicated.

4.2 Scenario One: Set S is not Specified and the Elements Can Be Duplicated

In this scenario, we assume that the set S of probabilities is not specified. Once the set has been determined, each element of the set can be used as an input probability an arbitrary number of times. The inputs are all assumed to be independent. As discussed in the introduction, we seek a set S of small size.

4.2.1 Generating Decimal Probabilities

In this section, we consider the case where the target probabilities are represented as *decimal* numbers. The problem is to find a small set S of source probabilities that can be transformed into an arbitrary target decimal probability. We provide a set S consisting of two elements.

Theorem 6

With circuits consisting of fanin-two AND gates and inverters, we can transform the set of source probabilities $\{0.4, 0.5\}$ *into an arbitrary decimal probability.* $\square$

Proof. First, we note that an inverter with a probabilistic input gives an output probability equal to one minus the input probability, as was shown in Equation (4.2). An AND gate with two independent inputs performs a multiplication of the input probabilities, as was shown in Equation (4.3). Thus, we need to prove: with the two operations $1-x$ and $x \cdot y$, we can transform the values from the set $\{0.4, 0.5\}$ into arbitrary decimal fractions. We prove this statement by induction on the number of digits n after the decimal point.

Base case:

1. $n = 0$. The values 0 and 1 correspond to deterministic inputs of zero and one, respectively.

2. $n = 1$. We can generate 0.1, 0.2, and 0.3 as follows:

$$0.1 = 0.4 \times 0.5 \times 0.5,$$
$$0.2 = 0.4 \times 0.5,$$
$$0.3 = (1 - 0.4) \times 0.5.$$

   Since we can generate the decimal fractions $0.1, 0.2, 0.3$, and 0.4, we can generate $0.6, 0.7, 0.8$, and 0.9 with an extra $1 - x$ operation. Together with the source value 0.5, we can transform the pair of values 0.4 and 0.5 into any decimal fraction with one digit after the decimal point.

Inductive step:

Assume that the statement holds for all $m \leq (n-1)$. Consider an arbitrary decimal fraction z with n digits after the decimal point. Let $u = 10^n \cdot z$. Here u is an integer.

Consider the following four cases.

1. The case where $0 \leq z \leq 0.2$.

   (a) The integer u is divisible by 2. Let $w = 5z$. Then $0 \leq w \leq 1$ and $w = (u/2) \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = 0.4 \times 0.5 \times w$.

   (b) The integer u is not divisible by 2 and $0 \leq z \leq 0.1$. Let $w = 10z$. Then $0 \leq w \leq 1$ and $w = u \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = 0.4 \times 0.5 \times 0.5 \times w$.

   (c) The integer u is not divisible by 2 and $0.1 < z \leq 0.2$. Let $w = 2 - 10z$. Then $0 \leq w < 1$ and $w = 2 - u \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - 0.5 \times w) \times 0.4 \times 0.5$.

2. The case where $0.2 < z \leq 0.4$.

   (a) The integer u is divisible by 4. Let $w = 2.5z$. Then $0 < w \leq 1$ and $w = (u/4) \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = 0.4 \times w$.

   (b) The integer u is not divisible by 4 but is divisible by 2. Let $w = 2 - 5z$. Then $0 \leq w < 1$ and $w = 2 - (u/2) \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - 0.5 \times w) \times 0.4$.

(c) The integer u is not divisible by 2 and $0.2 < u \leq 0.3$. Let $w = 10z - 2$. Then $0 < w \leq 1$ and $w = u \cdot 10^{-n+1} - 2$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - (1 - 0.5 \times w) \times 0.5) \times 0.4$.

(d) The integer u is not divisible by 2 and $0.3 < u \leq 0.4$. Let $w = 4 - 10z$. Then $0 \leq w < 1$ and $w = 4 - u \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - 0.5 \times 0.5 \times w) \times 0.4$.

3. The case where $0.4 < z \leq 0.5$. Let $w = 1 - 2z$. Then $0 \leq w < 0.2$ and w falls into case 1. Thus, we can generate w. It follows that z can be generated as $z = 0.5 \times (1 - w)$.

4. The case where $0.5 < z \leq 1$. Let $w = 1 - z$. Then $0 \leq w < 0.5$ and w falls into one of the above three cases. Thus, we can generate w. It follows that z can be generated as $z = 1 - w$.

For all of the above cases, we proved that we can transform the pair of values 0.4 and 0.5 into z with the two operations $1 - x$ and $x \cdot y$. Thus, we proved the statement for all $m \leq n$. By induction, the statement holds for all integers n. $\square$

Based on the proof above, we derive an algorithm to synthesize a circuit that transforms the probabilities from the set $\{0.4, 0.5\}$ into an arbitrary decimal probability z. This is shown in Algorithm 1.

The function GetDigits(z) in Algorithm 1 returns the number of digits after the decimal point of z. The algorithm iterates until z has at most one digit after the decimal point. During each iteration, it calls the function ReduceDigit(ckt, z). This

Algorithm 1 Synthesize a circuit consisting of AND gates and inverters that transforms the probabilities from the set $\{0.4, 0.5\}$ into a target decimal probability.

```
1: {Given an arbitrary decimal probability $0 \leq z \leq 1$.}
2: Initialize $ckt$;
3: while GetDigits($z$) $> 1$ do
4:     $(ckt, z) \Leftarrow$ ReduceDigit$(ckt, z)$;
5: end while
6: $ckt \Leftarrow$ AddBaseCkt$(ckt, z)$; {Base case: $z$ has at most one digit after the decimal
   point.}
7: return $ckt$;
```

function, shown in Algorithm 2, converts z into a number w with one less digit after the decimal point than z. It is implemented based on the inductive step in the proof of Theorem 6. Finally, the algorithm calls the function AddBaseCkt(ckt, z) to add logic gates to realize a number z with at most one digit after the decimal point; this corresponds to the base case of the proof.

The function ReduceDigit(ckt, z) in Algorithm 2 builds the circuit from the output back to the inputs. During its construction, the circuit always has a single dangling input. Initially, the circuit is just a wire connecting an input to the output. The function AddInverter(ckt) attaches an inverter to the dangling input creating a new dangling input. The function AddAND(ckt, p) attaches a fanin-two AND gate to the dangling input; one of the AND gate's inputs is the new dangling input; the other is set to a random source of probability p. In Algorithm 2, Lines 3–5 correspond to Case 4 in the proof; Lines 6–9 correspond to Case 3; Lines 10–19 correspond to Case 1; and Lines 20–34 correspond to Case 2.

The area complexity of the synthesized circuit is linear in the number of digits after the target value's decimal point, since at most 3 AND gates and 3 inverters are needed to generate a value with n digits after the decimal point from a value with $(n-1)$ digits

Algorithm 2 ReduceDigit(ckt, z)

```
{Given a partial circuit $ckt$ and an arbitrary decimal probability $0 \leq z \leq 1$.}
$n \Leftarrow$ GetDigits($z$);
**if** $z > 0.5$ **then** {Case 4}
   $z \Leftarrow 1 - z$; AddInverter($ckt$);
**end if**
**if** $0.4 < z \leq 0.5$ **then** {Case 3}
   $z \Leftarrow z/0.5$; AddAND($ckt, 0.5$);
   $z \Leftarrow 1 - z$; AddInverter($ckt$);
**end if**
**if** $z \leq 0.2$ **then** {Case 1}
   $z \Leftarrow z/0.4$; AddAND($ckt, 0.4$);
   $z \Leftarrow z/0.5$; AddAND($ckt, 0.5$);
   **if** GetDigits($z$) $< n$ **then**
      **go to** END;
   **end if**
   **if** $z > 0.5$ **then**
      $z \Leftarrow 1 - z$; AddInverter($ckt$);
   **end if**
   $z = z/0.5$; AddAND($ckt, 0.5$);
**else** {Case 2: $0.2 < z \leq 0.4$}
   $z \Leftarrow z/0.4$; AddAND($ckt, 0.4$);
   **if** GetDigits($z$) $< n$ **then**
      **go to** END;
   **end if**
   $z \Leftarrow 1 - z$; AddInverter($ckt$);
   $z \Leftarrow z/0.5$; AddAND($ckt, 0.5$);
   **if** GetDigits($z$) $< n$ **then**
      **go to** END;
   **end if**
   **if** $z > 0.5$ **then**
      $z \Leftarrow 1 - z$; AddInverter($ckt$);
   **end if**
   $z = z/0.5$; AddAND($ckt, 0.5$);
**end if**
END: **return** $ckt, z$;
```

after the decimal point.[1] The number of AND gates in the synthesized circuit is at most $3n$.

Example 8

We show how to generate the probability value 0.757*. Based on Algorithm 1, we can derive a sequence of operations that transform* 0.757 *to* 0.7*:*

$$0.757 \overset{1-}{\Longrightarrow} 0.243 \overset{/0.4}{\Longrightarrow} 0.6075 \overset{1-}{\Longrightarrow} 0.3925 \overset{/0.5}{\Longrightarrow} 0.785 \overset{1-}{\Longrightarrow} 0.215 \overset{/0.5}{\Longrightarrow} 0.43,$$

$$0.43 \overset{/0.5}{\Longrightarrow} 0.86 \overset{1-}{\Longrightarrow} 0.14 \overset{/0.4}{\Longrightarrow} 0.35 \overset{/0.5}{\Longrightarrow} 0.7.$$

Since 0.7 *can be realized as* $0.7 = 1 - (1 - 0.4) \times 0.5$*, we obtain the circuit shown in Figure 4.3. (Note that here we use a black dot to represent an inverter.)* □

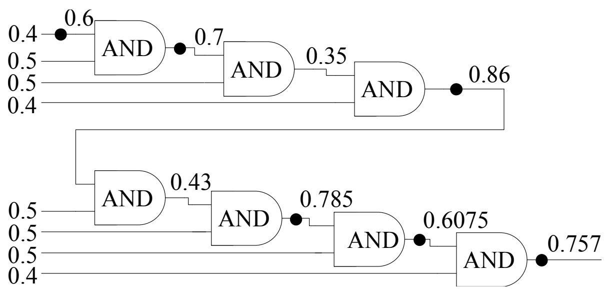


Figure 4.3: A circuit transforming the set of source probabilities $\{0.4, 0.5\}$ into a decimal output probability of 0.757.

Remarks:

1. One may question the usefulness of synthesizing a circuit that generates arbitrary *decimal* fractions. In [28], Wilhelm and Bruck proposed a scheme for synthesizing switching circuits that generate arbitrary *binary* probabilities. A switching circuit consists of relays that are either open or closed; the circuit computes a logical value of one if there exists a closed path through the circuit. By mapping every switch connected in series to an AND gate and every switch connected in parallel to an

[1] In Case 3, z is transformed into $w = 1 - 2z$ where w falls in Case 1(a). Thus, we actually need only 3 AND gates and 1 inverter for Case 3. For the other cases, it is not hard to see that we need at most 3 AND gates and 3 inverters.

OR gate, we can easily derive a combinational circuit that generates an arbitrary binary probability. Since any decimal fractional value can be approximated by a binary fractional value, we can build combinational circuits implementing decimal probabilities this way. However, the circuits synthesized by our procedure are less costly in terms of area.

To see this, consider a decimal fraction q with n digits. The circuit that Algorithm 1 synthesizes to generate q has at most $3n$ AND gates. For the approximation error of the binary fraction for q to be below $1/10^n$, the number of digits m of the binary fraction should be greater than $n \log_2 10$. In [28], it is proved that the minimal number of probabilistic switches needed to generate a binary fraction of m digits is m. Assuming that we build an equivalent combinational circuit consisting of AND gates and inverters, we need $m-1$ AND gates to implement the binary fraction.[2] Thus, the combinational logic realizing the binary approximation needs more than $n \log_2 10 \approx 3.32n$ AND gates. This is more than the number of AND gates in the circuit synthesized by our procedure.

2. In many applications, we need to generate many different target probabilities. To make these target probabilities *independent*, we can generate each of them from a different collection of input probabilities. It is inexpensive to generate different collections of input probabilities taking values from the source set, since we can generate independent copies of each probability in the source set cheaply.

4.2.2 Reducing the Depth

The circuits produced by Algorithm 1 have a linear topology (i.e., each gate adds to the depth of the circuit). For practical purposes, we want circuits with shallower depth. In this section, we explore two kinds of optimizations for reducing the depth.

[2] Of course, an OR gate can be converted into an AND gate with inverters at both the inputs and the output.

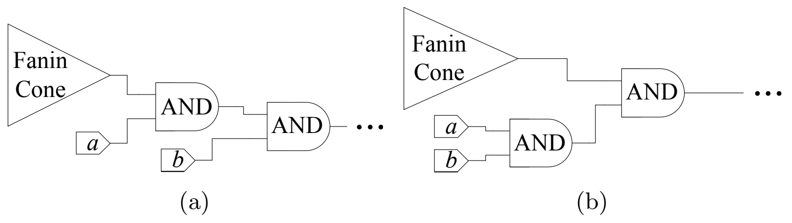


Figure 4.4: An illustration of balancing to reduce the depth of the circuit. Here a and b are primary inputs. (a): The circuit before balancing. (b): The circuit after balancing.

The first kind of optimization is at the logic level. The circuit synthesized by Algorithm 1 is composed of inverters and AND gates. We can reduce its depth by properly repositioning certain AND gates, as illustrated in Figure 4.4. We refer to such optimization as *balancing*.

The second kind of optimization is at a higher level, based on the factorization of the decimal fraction. We use the following example to illustrate the basic idea.

Example 9

Suppose we want to generate the decimal probability value 0.49.

Method based on Algorithm 1: We can derive the following transformation sequence:

$$0.49 \overset{/0.5}{\Longrightarrow} 0.98 \overset{1-}{\Longrightarrow} 0.02 \overset{/0.4}{\Longrightarrow} 0.05 \overset{/0.5}{\Longrightarrow} 0.1.$$

The synthesized circuit is shown in Figure 4.5(a). Notice that the circuit is balanced; it has five AND gates and a depth of four.[3]

Method based on factorization: Notice that $0.49 = 0.7 \times 0.7$. *Thus, we can generate the probability 0.7 twice and feed these values into an AND gate. The synthesized circuit is shown in Figure 4.5(b). Compared to the circuit in Figure 4.5(a), both the number of AND gates and the depth of the circuit are reduced.* $\square$

[3] *When counting depth, we ignore inverters.*

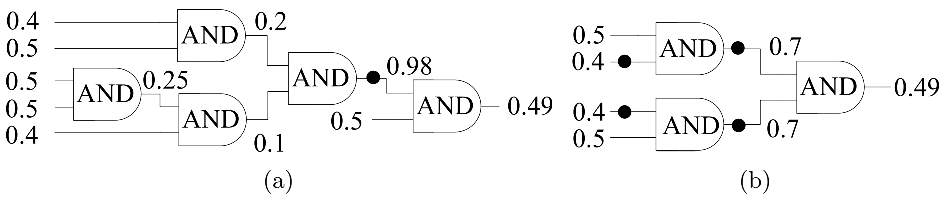


Figure 4.5: Synthesizing combinational logic to generate the probability 0.49. (a): The circuit synthesized through Algorithm 1. (b): The circuit synthesized based on factorization.

Algorithm 3 shows the procedure that synthesizes the circuit based on the factorization of the decimal fraction. The factorization is actually carried out on the numerator. A crucial function is PairCmp(a_l, a_r, b_l, b_r), which compares the integer factor pair (a_l, a_r) with the pair (b_l, b_r) and returns a positive (negative) value if the pair (a_l, a_r) is better (worse) than the pair (b_l, b_r). Algorithm 4 shows how the function PairCmp(a_l, a_r, b_l, b_r) is implemented.

The quality of a factor pair (a_l, a_r) should reflect the depth of the circuit that generates the original probability based on that factorization. For this purpose, we define a function EstDepth(x) to estimate the depth of the circuit that generates the decimal fraction with a numerator x. If $1 \leq x \leq 9$, the corresponding fraction is $x/10$. EstDepth(x) is set as the depth of the circuit that generates the fraction $x/10$, which is

$$\text{EstDepth}(x) = \begin{cases} 0, & x = 4, 5, 6, \\ 1, & x = 2, 3, 7, 8, \\ 2, & x = 1, 9. \end{cases}$$

When $x \geq 10$, we use a simple heuristic to estimate the depth: we let EstDepth$(x) = \lceil \log_{10}(x) \rceil + 1$. The intuition behind this is that the depth of the circuit is a monotonically increasing function of the number of digits of x. The estimated depth of the circuit that generates the original fraction based on the factor pair (a_l, a_r) is

$$\max\{\text{EstDepth}(a_l), \text{EstDepth}(a_r)\} + 1. \tag{4.4}$$

Algorithm 3 ProbFactor(ckt, z)

```
1: {Given a partial circuit $ckt$ and an arbitrary decimal probability $0 \leq z \leq 1$.}
2: $n \Leftarrow$ GetDigits($z$);
3: if $n \leq 1$ then
4:    $ckt \Leftarrow$ AddBaseCkt($ckt, z$);
5:    return $ckt$;
6: end if
7: $u \Leftarrow 10^n z$; $(u_l, u_r) \Leftarrow (1, u)$; {$u$ is the numerator of the fraction $z$}
8: for each factor pair $(a, b)$ of $u$ do
9:    if PairCmp($u_l, u_r, a, b$) $< 0$ then
10:       $(u_l, u_r) \Leftarrow (a, b)$; {Choose the best factor pair for $z$}
11:    end if
12: end for
13: $w \Leftarrow 10^n - u$; $(w_l, w_r) \Leftarrow (1, w)$;
14: for each factor pair $(a, b)$ of $w$ do
15:    if PairCmp($w_l, w_r, a, b$) $< 0$ then
16:       $(w_l, w_r) \Leftarrow (a, b)$; {Choose the best factor pair for $1 - z$}
17:    end if
18: end for
19: if PairCmp($u_l, u_r, w_l, w_r$) $< 0$ then
20:    $(u_l, u_r) \Leftarrow (w_l, w_r)$; $z \Leftarrow w/10^n$;
21:    AddInverter($ckt$);
22: end if
23: if IsTrivialPair($u_l, u_r$) then {$u_l = 1$ or $u_r = 1$}
24:    $(ckt, z) \Leftarrow$ ReduceDigit($ckt, z$);
25:    $ckt \Leftarrow$ ProbFactor($ckt, z$);
26:    return $ckt$;
27: end if
28: $n_l \Leftarrow \lceil \log_{10}(u_l) \rceil$; $n_r \Leftarrow \lceil \log_{10}(u_r) \rceil$;
29: if $n_l + n_r > n$ then {Unable to factor $z$ into two decimal fractions in the unit
    interval}
30:    $(ckt, z) \Leftarrow$ ReduceDigit($ckt, z$);
31:    $ckt \Leftarrow$ ProbFactor($ckt, z$);
32:    return $ckt$;
33: end if
34: $z_l \Leftarrow u_l/10^{n_l}$; $z_r \Leftarrow u_r/10^{n_r}$;
35: $ckt_l \Leftarrow$ ProbFactor($ckt_l, z_l$);
36: $ckt_r \Leftarrow$ ProbFactor($ckt_r, z_r$);
37: Connect the input of $ckt$ to an AND gate with two inputs as $ckt_l$ and $ckt_r$;
38: if $n_l + n_r < n$ then
39:    AddExtraLogic($ckt, n - n_l - n_r$);
40: end if
41: return $ckt$;
```

The function PairCmp(a_l, a_r, b_l, b_r) essentially compares the quality of pair (a_l, a_r) and pair (b_l, b_r) based on Equation (4.4). Further details are given in Algorithm 4.

Algorithm 4 PairCmp(a_l, a_r, b_l, b_r)

```
1: {Given two integer factor pairs $(a_l, a_r)$ and $(b_l, b_r)$}
2: $c_l \Leftarrow$ EstDepth$(a_l)$; $c_r \Leftarrow$ EstDepth$(a_r)$;
3: $d_l \Leftarrow$ EstDepth$(b_l)$; $d_r \Leftarrow$ EstDepth$(b_r)$;
4: Order$(c_l, c_r)$; {Order $c_l$ and $c_r$, so that $c_l \leq c_r$}
5: Order$(d_l, d_r)$; {Order $d_l$ and $d_r$, so that $d_l \leq d_r$}
6: **if** $c_r < d_r$ **then** {The circuit w.r.t. the first pair has smaller depth}
7:   **return** 1;
8: **else if** $c_r > d_r$ **then** {The circuit w.r.t. the first pair has larger depth}
9:   **return** -1;
10: **else**
11:   **if** $c_l < d_l$ **then** {The circuit w.r.t. the first pair has fewer ANDs}
12:     **return** 1;
13:   **else if** $c_l > d_l$ **then** {The circuit w.r.t. the first pair has more ANDs}
14:     **return** -1;
15:   **else**
16:     **return** 0;
17:   **end if**
18: **end if**
```

In Algorithm 3, Lines 2–6 correspond to the trivial fractions. If the fraction z is non-trivial, Lines 7–12 choose the best factor pair (u_l, u_r) of u, where u is the numerator of the fraction z. Lines 13–18 choose the best factor pair (w_l, w_r) of w, where w is the numerator of the fraction $1 - z$. Finally, Lines 19–22 choose the better factor pair of (u_l, u_r) and (w_l, w_r). Here, we consider the factorization on both z and $1 - z$, since in some cases the latter might be better than the former. An example is $z = 0.37$. Note that $1 - z = 0.63 = 0.7 \times 0.9$; this has a better factor pair than z itself.

After obtaining the best factor pair, we check whether we can use it. Lines 23–27 check whether the factor pair (u_l, u_r) is trivial; a factor pair is considered trivial if $u_l = 1$ or $u_r = 1$. If the best factor pair is trivial, we call the function ReduceDigit(ckt, z) in Algorithm 2 to transform z into a new value with one less digit after the decimal point. Then we perform factorization on the new value.

If the best factor pair is non-trivial, Lines 28–33 continue to check whether the factor pair can be transformed into two decimal fractions in the unit interval. Let n_l be the number of digits of the integer u_l and n_r be the number of digits of the integer u_r. If $n_l + n_r > n$, where n is the number of digits after the decimal point of z, then it is impossible to use the factor pair (u_l, u_r) to factorize z. For example, consider $z = 0.143$. Although we could factorize 143 as 11×13, we cannot use the factor pair $(11, 13)$ since the factorization 0.11×1.3 and the factorization 1.1×0.13 both contain a fraction larger than 1; a probability value can never be larger than 1. If it is impossible to use the best factor pair (u_l, u_r) to factorize z, we call the function ReduceDigit(ckt, z) in Algorithm 2 to transform z into a new value with one less digit after the decimal point. Then we perform factorization on the new value.

Finally, if it is possible to use the best factor pair, Lines 34–37 synthesize two circuits for fractions $u_l/10^{n_l}$ and $u_r/10^{n_r}$, respectively, and then combine these two circuits with an AND gate. Lines 38–40 check whether $n > n_l + n_r$. If this is the case, we have

$$z = u/10^n = u_l/10^{n_l} \cdot u_r/10^{n_r} \cdot 0.1^{n-n_l-n_r}.$$

We need to add an extra AND gate with one input probability as $0.1^{n-n_l-n_r}$ and the other input probability as $u_l/10^{n_l} \cdot u_r/10^{n_r}$. The extra logic is added through the function AddExtraLogic(ckt, m).

4.2.3 Experimental Results

In this section, we empirically validate the effectiveness of the synthesis scheme that is presented in the previous section. For logic-level optimization, we used the "balance" command of the synthesis tool ABC [29]. We find that it is very effective in reducing the depth of tree-style circuits.[4]

Table 4.1 compares the quality of the circuits generated by three different schemes. The first scheme, called "Basic," is based on Algorithm 1. It generates a linear-style

[4] We find that the other synthesis commands of ABC such as "rewrite" do not affect the depth or the number of AND gates of a tree-style AND-inverter graph.

Table 4.1: A comparison of the basic synthesis scheme, the basic synthesis scheme with balancing, and the factorization-based synthesis scheme with balancing.

	Basic		Basic+Balance		Factor+Balance			
#Digits n	#AND	Depth	#AND a_1	Depth d_1	#AND a_2	Depth d_2	#AND Imprv.(%) $100\frac{a_1-a_2}{a_1}$	Depth Imprv.(%) $100\frac{d_1-d_2}{d_1}$
2	3.67	3.67	3.67	2.98	3.22	2.62	12.1	11.9
3	6.54	6.54	6.54	4.54	5.91	3.97	9.65	12.5
4	9.47	9.47	9.47	6.04	8.57	4.86	9.45	19.4
5	12.43	12.43	12.43	7.52	11.28	5.60	9.21	25.6
6	15.40	15.40	15.40	9.01	13.96	6.17	9.36	31.5
7	18.39	18.39	18.39	10.50	16.66	6.72	9.42	35.9
8	21.38	21.38	21.38	11.99	19.34	7.16	9.55	40.3
9	24.37	24.37	24.37	13.49	22.05	7.62	9.54	43.6
10	27.37	27.37	27.37	14.98	24.74	7.98	9.61	46.7
11	30.36	30.36	30.36	16.49	27.44	8.36	9.61	49.3
12	33.35	33.35	33.35	17.98	30.13	8.66	9.65	51.8

circuit. The second scheme, called "Basic+Balance," combines Algorithm 1 and the logic-level balancing algorithm. The third scheme, called "Factor+Balance," combines Algorithm 3 and the logic-level balancing algorithm. We performed experiments on a set of target decimal probabilities that have n digits after the decimal point and average the results. Table 4.1 shows the results for n ranging from 2 to 12. When $n \leq 5$, we synthesized circuits for all possible decimal probabilities with n digits after the decimal point. When $n \geq 6$, we randomly chose 100,000 decimal probabilities with n digits after the decimal point as the synthesis targets. We show the average number of AND gates and the average depth.

Compared to the "Basic+Balance" scheme, the "Factor+Balance" scheme reduces the average number of AND gates by 10% and the average depth by more than 10%, for all n. The percentage of reduction of the average depth increases with increasing n. For $n = 12$, the average depth of the circuits is reduced by more than 50%. Both the

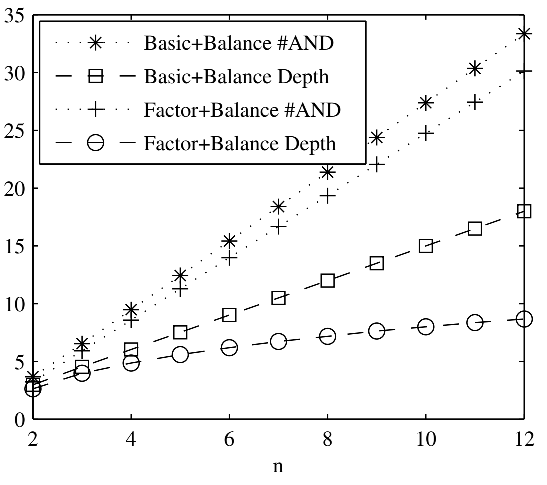


Figure 4.6: Average number of AND gates and depth of the circuits versus n.

"Basic+Balance" and the "Factor+Balance" synthesis schemes have only millisecond-order CPU runtimes.

In Figure 4.6, we plot the average number of AND gates and the average depth of the circuits versus n for the "Basic+Balance" and "Factor+Balance" schemes. The figure shows that the "Factor+Balance" scheme is clearly superior. The average number of AND gates in the circuits synthesized by both schemes increases linearly with n. The average depth of the circuits synthesized by the "Basic+Balance" scheme also increases linearly with n. In contrast, the average depth of the circuits synthesized by the "Factor+Balance" scheme increases logarithmically with n.

4.2.4 Generate Decimal Fractions with a Single Source Probability

Theorem 6 shows that there exists a pair of probabilities that can be used to generate arbitrary decimal fractions. A stronger question is whether we can further reduce the size of the set down to one, i.e., whether there exists a real number $0 \leq p \leq 1$ such that any decimal fractions can be generated from p with combinational logic.

The first result is that there is no *rational* number p such that an arbitrary decimal fraction can be generated from that p through combinational logic. To prove this, we first need the following lemma.

Lemma 1

If the probability $\frac{1}{2}$ can be generated from a rational probability p through combinational logic, then $p = \frac{1}{2}$. $\square$

Proof. Obviously, $0 < p < 1$. Thus, we can assume that

$$p = \frac{a}{b}, \tag{4.5}$$

where both a and b are positive integers, satisfying that $a < b$ and $(a, b) = 1$.

Moreover, we can assume that $a \geq b - a$. Otherwise, suppose that $a < b - a$. Since we can generate $\frac{1}{2}$ from p, we can also generate $\frac{1}{2}$ from $p^* = 1 - p$ by using an inverter to convert p^* into p. Note that $p^* = \frac{a^*}{b^*}$, where $a^* = b - a$ and $b^* = b$, satisfying that $a^* > b^* - a^*$. Thus, we can assume that $a \geq b - a$.

Suppose that the Boolean function of the combinational logic that generates $\frac{1}{2}$ from p is $f(x_1, \ldots, x_n)$. For $k = 0, 1, \ldots, n$, define

$$A_k = \{(x_1, \ldots, x_n) | (x_1, \ldots, x_n) \in \{0,1\}^n, f(x_1, \ldots, x_n) = 1, \text{ and } \sum_{i=1}^{n} x_i = k\}.$$

(i.e., A_k consists of n-tuples over $\{0, 1\}$ that have exactly k ones and let the function f evaluate to one.) For $k = 0, 1, \ldots, n$, define l_k to be the cardinality of the set A_k. Note that $0 \leq l_k \leq \binom{n}{k}$.

Since each input of the combinational logic has probability p of being 1, we have

$$\frac{1}{2} = \sum_{k=0}^{n} l_k (1-p)^{n-k} p^k. \tag{4.6}$$

Let $c = b - a$. Based on Equation (4.5), we can rewrite Equation (4.6) as

$$b^n = 2 \sum_{k=0}^{n} l_k a^k c^{n-k}. \tag{4.7}$$

From Equation (4.7), we can show that $a = 1$. By contraposition, suppose that $a > 1$. Since $0 \leq l_0 \leq \binom{n}{0} = 1$, l_0 is either 0 or 1. If $l_0 = 0$, then from Equation (4.7), we have

$$b^n = 2\sum_{k=1}^{n} l_k a^k c^{n-k} = 2a\sum_{k=1}^{n} l_k a^{k-1} c^{n-k}.$$

Thus, $a|b^n$. Since $(a, b) = 1$, the only possibility is that $a = 1$ which is contradictory to our hypothesis that $a > 1$. Therefore, we have $l_0 = 1$. Together with the binomial expansion $b^n = \sum_{k=0}^{n} \binom{n}{k} a^k c^{n-k}$, we can rewrite Equation (4.7) as

$$c^n + \sum_{k=1}^{n} \binom{n}{k} a^k c^{n-k} = 2c^n + 2\sum_{k=1}^{n} l_k a^k c^{n-k}.$$

or

$$c^n = a\sum_{k=1}^{n} \left(\binom{n}{k} - 2l_k\right) a^{k-1} c^{n-k}. \tag{4.8}$$

Thus, $a|c^n$. Since $(a, b) = 1$ and $c = b - a$, we have $(a, c) = 1$. Thus, the only possibility is that $a = 1$, which is contradictory to our hypothesis that $a > 1$.

Therefore, we proved that $a = 1$. Together with the assumption that $b - a \leq a < b$, we get $b = 2$. Thus, p can only be $\frac{1}{2}$. $\square$

Now, we can prove the original statement.

Theorem 7

There is no rational number p such that an arbitrary decimal fraction can be generated from that p with combinational logic. $\square$

PROOF. We prove the above statement by the way of contraposition. Suppose that there exists a rational number p such that an arbitrary decimal fraction can be generated from it through combinational logic.

Since an arbitrary decimal fraction can be generated from p, $0.5 = \frac{1}{2}$ can be generated. Thus, based on Lemma 1, we have $p = \frac{1}{2}$.

Note that $0.2 = \frac{1}{5}$ is also a decimal number. Thus, there exists a combinational circuit which can generate the decimal fraction $\frac{1}{5}$ from $p = \frac{1}{2}$.

Suppose that the Boolean function of the combinational circuit that generates $\frac{1}{5}$ from $\frac{1}{2}$ is $f(x_1, \ldots, x_n)$. For $k = 0, 1, \ldots, n$, define

$$A_k = \{(x_1, \ldots, x_n) | (x_1, \ldots, x_n) \in \{0,1\}^n, f(x_1, \ldots, x_n) = 1, \text{ and } \sum_{i=1}^{n} x_i = k\}.$$

(i.e., A_k consists of n-tuples over $\{0,1\}$ that have exactly k ones and let the function f evaluate to one.) For $k = 0, 1, \ldots, n$, define l_k to be the cardinality of the set A_k.

Since each input of the combinational circuit has probability $\frac{1}{2}$ of being 1, we have

$$\frac{1}{5} = \sum_{k=0}^{n} l_k \left(1 - \frac{1}{2}\right)^{n-k} \left(\frac{1}{2}\right)^{k},$$

or

$$2^n = 5 \sum_{k=0}^{n} l_k,$$

which is impossible since the right-hand side is a multiple of 5. Therefore, we proved the statement in the theorem. □

Thus, based on Theorem 7, we have the conclusion that if there exists a p such that an arbitrary decimal fraction can be generated from p through combinational logic, the number p must be irrational.

On the one hand, we note that if such a value p exists, then 0.4 and 0.5 can be generated from it. On the other hand, due to Theorem 6, if p can generate 0.4 and 0.5, then p can generate arbitrary decimal numbers. The following lemma shows that such a value p that could generate 0.4 and 0.5 does, in fact, exist.

Lemma 2

The polynomial $g(t) = 10t - 20t^2 + 20t^3 - 10t^4 - 1$ has a real root $0 < p < 0.5$. This value p can generate both 0.4 and 0.5 through combinational logic. □

Proof. First, note that $g(0) = -1 < 0$ and that $g(0.5) = 0.875 > 0$. Based on the continuity of the function $g(t)$, there exists a $0 < p < 0.5$ such that $g(p) = 0$. Let polynomial $h(t) = \frac{1}{10}(g(t) + 1) = t - 2t^2 + 2t^3 - t^4$. Then, $h(p) = 0.1$.

Note that the Boolean function

$$f_1(x_1, x_2, x_3, x_4, x_5) = (x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5)$$

has 30 minterms, $m_1, m_2, \ldots, m_{30}$. It is not hard to verify that with $P(x_i = 1) = p$ for $i = 1, 2, 3, 4, 5$, the output probability of f_1 is

$$\begin{aligned} p_1 &= 5(1-p)^4 p + 10(1-p)^3 p^2 + 10(1-p)^2 p^3 + 5(1-p)p^4 \\ &= 5h(p) = 0.5. \end{aligned}$$

Thus, the probability value 0.5 can be generated. Now consider the Boolean function

$$\begin{aligned} f_2(x_1, x_2, x_3, x_4, x_5) = &(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee \neg x_5) \\ &\wedge (\neg x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4 \vee \neg x_5). \end{aligned}$$

It has 24 minterms, $m_2, m_4, m_5, \ldots, m_8, m_{10}$, $m_{12}, m_{13}, \ldots, m_{24}, m_{26}, m_{28}, m_{29}, m_{30}$. It is not hard to verify that with $P(x_i = 1) = p$ for $i = 1, 2, 3, 4, 5$, the output probability of f_2 is

$$\begin{aligned} p_2 &= 4(1-p)^4 p + 8(1-p)^3 p^2 + 8(1-p)^2 p^3 + 4(1-p)p^4 \\ &= 4h(p) = 0.4. \end{aligned}$$

Thus, the probability value 0.4 can be generated. □

Based on Theorem 6 and Lemma 2, we have the following theorem.

Theorem 8

With the set $S = \{p\}$, where p is the root of the polynomial $g(t) = 10t - 20t^2 + 20t^3 - 10t^4 - 1$ in the unit interval, we can generate arbitrary decimal fractions with combinational logic. □

4.2.5 Generating Base-n Fractional Probabilities

In this section, we generalize the result of Section 4.2.4. We show that for any integer $n \geq 2$, there exists a real number $0 \leq r \leq 1$ that can be transformed into an arbitrary base-n fractional probability $\frac{m}{n^d}$ with combinational logic.

First, we show that we can transform a set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into an arbitrary base-n fractional probability $\frac{m}{n^d}$.

Theorem 9

Let $n \geq 2$ be an integer. For any integers $d \geq 1$ and $0 \leq m \leq n^d$, we can transform the set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into a base-n fractional probability $\frac{m}{n^d}$ with a circuit having $2d-1$ inputs. □

Proof. We prove the above claim by induction on d.

Base case: When $d = 1$, we can obtain each base-n fractional probability $\frac{m}{n}$ $(0 \leq m \leq n)$ directly from an input since the input probability set is $\{\frac{1}{n}, \ldots, \frac{n-1}{n}\}$ and the probabilities 0 and 1 correspond to deterministic values of zero and one, respectively.

Inductive step: Assume the claim holds for $d-1$. Now consider any integer $0 \leq m \leq n^d$. We can write m as $m = an^{d-1} + b$ with an integer $0 \leq a < n$ and an integer $0 \leq b \leq n^{d-1}$.

Consider a multiplexer with data input x_1 and x_2, selecting input s, and output y, as shown in Figure 4.7. The Boolean function of the multiplexer is:

$$y = (x_1 \wedge s) \vee (x_2 \wedge \neg s).$$

By the induction hypothesis, we can transform the set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into the probability $\frac{b}{n^{d-1}}$ with a circuit Q that has $2d-3$ inputs. In order to generate the output probability $\frac{m}{n^d}$, we let the inputs x_1 and x_2 of the multiplexer have probability $\frac{a+1}{n}$ and $\frac{a}{n}$, respectively, and we connect the input s to the output of a circuit Q that generates the probability $\frac{b}{n^{d-1}}$, as shown in Figure 4.7. Note that the inputs to x_1 and x_2 are either probabilistic inputs with a value from the set $\{\frac{1}{n}, \ldots, \frac{n-1}{n}\}$, or deterministic inputs of zero or one. With the primary inputs of the entire circuit being independent, all the inputs of the multiplexer are also independent. The probability

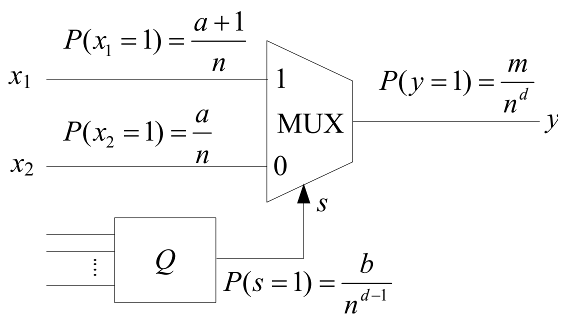

$P(x_1 = 1) = \frac{a+1}{n}$
x_1
1
MUX
0
$P(y=1) = \frac{m}{n^d}$
y
$P(x_2 = 1) = \frac{a}{n}$
x_2
s
Q
$P(s=1) = \frac{b}{n^{d-1}}$

Figure 4.7: The circuit generating the base-n fractional probability $\frac{m}{n^d}$, where m is written as $m = an^{d-1} + b$ with $0 \le a < n$ and $0 \le b \le n^{n-1}$. The circuit Q in the figure generates the base-n fractional probability $\frac{b}{n^{d-1}}$.

that y is one is

$$\begin{aligned}
P(y = 1) &= P(x_1 = 1, s = 1) + P(x_2 = 1, s = 0) \\
&= P(x_1 = 1)P(s = 1) + P(x_2 = 1)P(s = 0) \\
&= \frac{a+1}{n}\frac{b}{n^{d-1}} + \frac{a}{n}\left(1 - \frac{b}{n^{d-1}}\right) \\
&= \frac{an^{d-1} + b}{n^d} = \frac{m}{n^d}.
\end{aligned}$$

Therefore, we can transform the set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into the probability $\frac{m}{n^d}$ with a circuit that has $2d - 3 + 2 = 2d - 1$ inputs. Thus, the claim holds for d. By induction, the claim holds for all $d \ge 1$. $\Box$

Remarks:

1. An equivalent result to Theorem 9 can be found in the work of Jeavons *et al.* [30]. There it is couched in information theoretic language in terms of concurrent operations on random binary sequences.

2. Our proof of Theorem 9 is constructive. It shows that we can synthesize a chain of $d - 1$ multiplexers to generate a base-n fractional probability $\frac{m}{n^d}$.

3. If some of the inputs to the chain of multiplexers are deterministic zeros or ones, we can further simplify the circuit. In such cases, the number of inputs of the entire circuit and the area of the circuit can be further reduced.

Next, we prove a theorem about the existence of a single real value that can be transformed into any value in a given set of rational probabilities through combinational logic.

Theorem 10

For any finite set of rational probabilities $R = \{p_1, p_2, \dots, p_M\}$, there exists a real number $0 < r < 1$ that can be transformed into probabilities in the set R through combinational logic. $\square$

Proof. We only need to prove that the statement is true under the condition that for all $1 \le i \le M$, $0 \le p_i \le 0.5$. In fact, given a general set of probabilities $R = \{p_1, p_2, \dots, p_M\}$, we can derive a new set of probabilities $R^* = \{p_1^*, p_2^*, \dots, p_M^*\}$, such that for all $1 \le i \le M$,

$$p_i^* = \begin{cases} p_i, & \text{if } p_i \le 0.5, \\ 1 - p_i, & \text{if } p_i > 0.5. \end{cases}$$

Then, for all $1 \le i \le M$, the element p_i^* of R^* satisfies that $0 \le p_i^* \le 0.5$. Once we prove that there exists a real number $0 < r < 1$ which can be transformed into any of the probabilities in the set R^*, then any probability in the original set R can also be generated from this value r: to generate $p_i = p_i^*$, we use the same circuit that generates the probability p_i^*; to generate $p_i = 1 - p_i^*$, we append an inverter to the output.

Therefore, we assume that for all $1 \le i \le M$, $0 \le p_i \le 0.5$. Further, without loss of generality, we can assume that $0 \le p_1 < \cdots < p_M \le 0.5$. Since probability 0 can be realized trivially by a deterministic value of zero, we assume that $p_1 > 0$. Since $p_1, \dots, p_M$ are rational probabilities, there exist positive integers $a_1, \dots, a_M$ and b such that for all $1 \le i \le M$, $p_i = \frac{a_i}{b}$. Since $0 < p_1 < \cdots < p_M \le 0.5$, we have $0 < a_1 < \cdots < a_M \le \frac{b}{2}$.

First, it is not hard to see that there exists a positive integer h such that $2^{h-1} > a_M h + 1$. For $k = 1, \ldots, h$, let $c_k = \left\lfloor \frac{\binom{h}{k}}{a_M} \right\rfloor$, where $\lfloor x \rfloor$ represents the largest integer less than or equal to x.

We will prove

$$a_M \sum_{k=1}^{h} c_k > 2^{h-1}. \tag{4.9}$$

In fact,

$$\begin{aligned} 2^h - a_M \sum_{k=1}^{h} c_k &= \sum_{k=0}^{h} \binom{h}{k} - \sum_{k=1}^{h} \left\lfloor \frac{\binom{h}{k}}{a_M} \right\rfloor a_M \\ &= 1 + \sum_{k=1}^{h} \left(\frac{\binom{h}{k}}{a_M} - \left\lfloor \frac{\binom{h}{k}}{a_M} \right\rfloor \right) a_M. \end{aligned}$$

Since $x - \lfloor x \rfloor < 1$, we have

$$2^h - a_M \sum_{k=1}^{h} c_k < 1 + \sum_{k=1}^{h} a_M = a_M h + 1 < 2^{h-1},$$

or

$$a_M \sum_{k=1}^{h} c_k > 2^{h-1}.$$

Now consider the polynomial $f(x) = \sum_{k=1}^{h} c_k x^k (1-x)^{h-k}$. Note that $f(0) = 0$ and $f(0.5) = \frac{1}{2^h} \sum_{k=1}^{h} c_k$. Based on Equation (4.9) and the fact that $a_M \le \frac{b}{2}$, we have

$$f(0.5) > \frac{1}{2a_M} \ge \frac{1}{b}.$$

Thus, $f(0) = 0 < \frac{1}{b} < f(0.5)$. Based on the continuity of the polynomial f, there exists a real number $0 < r < 0.5 < 1$ such that $f(r) = \frac{1}{b}$.

For all $i = 1, \ldots, M$, set $l_{i,0} = 0$. For all $i = 1, \ldots, M$ and all $k = 1, 2, \ldots, h$, set $l_{i,k} = a_i c_k$. Since for all $k = 1, \ldots, h$, c_k is an integer and $0 \le c_k \le \frac{\binom{h}{k}}{a_M}$, then for all $i = 1, \ldots, M$ and all $k = 1, 2, \ldots, h$, $l_{i,k}$ is an integer and $0 \le l_{i,k} = a_i c_k \le a_M c_k \le \binom{h}{k}$.

For $k = 0, 1, \ldots, h$, let $A_k = \{(a_1, a_2, \ldots, a_h) \in \{0,1\}^h : \sum_{i=1}^{h} a_i = k\}$ (i.e., A_k consists of h-tuples over $\{0,1\}$ having exactly k ones.). For any $1 \leq i \leq M$, consider a circuit with h inputs realizing a Boolean function that takes exactly $l_{i,k}$ values 1 on each A_k ($k = 0, 1, \ldots, h$). If we set all the input probabilities to be r, then the output probability is

$$\begin{aligned} p_o &= \sum_{k=0}^{h} l_{i,k} r^k (1-r)^{h-k} = \sum_{k=1}^{h} a_i c_k r^k (1-r)^{h-k} \\ &= a_i f(r) = \frac{a_i}{b}. \end{aligned}$$

Thus, we can transform r into any number in the set $\{p_1, \ldots, p_M\}$ through combinational logic. □

Theorems 9 and 10 lead to the following corollary.

Corollary 2

Given an integer $n \geq 2$, there exists a real number $0 < r < 1$ which can be transformed into any base-n fractional probability $\frac{m}{n^d}$ (d and m are integers with $d \geq 1$ and $0 \leq m \leq n^d$) through combinational logic. □

Proof. Based on Theorem 10, there exists a real number $0 < r < 1$ which can be transformed into any probability in the set $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$. Further, based on Theorem 9, the statement in the corollary holds. □

4.3 Scenario Two: Set S is Specified and the Elements Cannot Be Duplicated.

The problem considered in this scenario is: given a set $S = \{p_1, p_2, \ldots, p_n\}$ and a target probability q, construct a circuit that, given inputs with probabilities from S, produces an output with probability q. Each element of S can be used as an input probability no more than once.

4.3.1 An Optimal Solution

In this section, we show an optimal solution to the problem based on linear 0-1 programming. With the assumption that the probabilities cannot be duplicated, we are building a circuit with n inputs, the i-th input of which has probability p_i. (If a probability is not used, then the corresponding input is just a dummy.)

Our method is based on a truth table for n variables. Each row of the truth table is annotated with the probability that the corresponding input combination occurs. Assume that the n variables are $x_1, x_2, \ldots, x_n$ and x_i has probability p_i. Then, the probability that the input combination $x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n$ ($a_i \in \{0, 1\}$, for $i = 1, \ldots, n$) occurs is

$$P(x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n) = \prod_{i=1}^{n} P(x_i = a_i).$$

A truth table for a two-input XOR gate is shown in Table 5.1. The fourth column is the probability that each input combination occurs. Here $P(x = 1) = p_x$ and $P(y = 1) = p_y$.

Table 4.2: A truth table for a two-input XOR gate.

x	y	z	Probability
0	0	0	$(1 - p_x)(1 - p_y)$
0	1	1	$(1 - p_x)p_y$
1	0	1	$p_x(1 - p_y)$
1	1	0	$p_x p_y$

The output probability is the sum of the probabilities of input combinations that produce an output of one. Assume that the probability of the i-th input combination, corresponding to minterm m_i, is r_i ($0 \le i \le 2^n - 1$) and that the output of the circuit corresponding to the i-th input combination is z_i ($z_i \in \{0,1\}, 0 \le i \le 2^n - 1$). Then, the output probability is

$$p_o = \sum_{i=0}^{2^n-1} z_i r_i. \tag{4.10}$$

For the example in Table 5.1, the output probability is

$$p_o = r_1 + r_2 = (1 - p_x)p_y + p_x(1 - p_y).$$

Thus, constructing a circuit with output probability q is equivalent to determining the z_i's such that Equation (4.10) evaluates to q. In the general case, depending on the values of p_i and q, it is possible that q cannot be exactly realized by any circuit. The problem then is to determine the z_i's such that the difference between the value of Equation (4.10) and q is minimized. We can formulate this as the following optimization problem:

$$\text{Find } z_i \text{ that minimizes } \left|\textstyle\sum_{i=0}^{2^n-1} z_i r_i - q\right| \tag{4.11}$$

$$\text{such that } z_i \in \{0,1\} \text{ for } i = 0, 1, \ldots, 2^n - 1. \tag{4.12}$$

The solution of this optimization problem can be derived by first separating it into two subproblems:

Problem 1

Find z_i that minimizes $obj_1 = \sum_{i=0}^{2^n-1} r_i z_i - q$, such that $\sum_{i=0}^{2^n-1} r_i z_i - q \ge 0$ and $z_i \in \{0,1\}$ for $i = 0, 1, \ldots, 2^n - 1$.

Problem 2

Find z_i that minimizes $obj_2 = q - \sum_{i=0}^{2^n-1} r_i z_i$ such that $q - \sum_{i=0}^{2^n-1} r_i z_i \ge 0$ and $z_i \in \{0,1\}$ for $i = 0, 1, \ldots, 2^n - 1$.

Problems 1 and 2 are linear 0-1 programming problems that can be solved using standard techniques. Suppose that the minimum solution to Problem 1 is $(z_0^*, z_1^*, \ldots, z_{2^n-1}^*)$ with $\mathrm{obj}_1 = \mathrm{obj}_1^*$ and the minimum solution to Problem 2 is $(z_0^{**}, z_1^{**}, \ldots, z_{2^n-1}^{**})$ with

$\text{obj}_2 = \text{obj}_2^*$. Then the solution to the original problem is the set of z_i's corresponding to $\min\{\text{obj}_1^*, \text{obj}_2^*\}$.

If the solution to the above optimization problem has $z_i = 1$, then the Boolean function should contain the minterm m_i; otherwise, it should not. A circuit implementing the solution can be readily synthesized.[5]

4.3.2 A Suboptimal Solution

The above solution is simple and optimal; it works well when n is small. However, when n is large, there are two difficulties with the implementation that might make it impractical. First, the solution is based on linear 0-1 programming, which is NP-hard. Therefore, the computational complexity will become significant. Secondly, if an application-specific integrated circuit (ASIC) is designed to implement the solution of the optimization problem, the circuit may need as many as $O(2^n)$ gates in the worst case. This may be too costly for large n.

In this section, we provide a greedy algorithm that yields suboptimal results. However, the difference between the output probability of the circuit that it synthesizes and the target probability q is bounded. The algorithm has good performance both in terms of its run-time and the size of the resulting circuit.

The idea of the greedy algorithm is that we construct a group of $n + 2$ circuits $C_0, C_1, \ldots, C_{n+1}$ such that the circuit C_k $(0 \le k \le n)$ has k probabilistic inputs and one deterministic input of either zero or one and the circuit C_{n+1} has n probabilistic inputs and two deterministic inputs of either zero or one. For all $0 \le k \le n$, the circuit C_{k+1} is constructed from C_k by replacing one input of C_k with a two-input gate.

The circuit C_0 is constructed by connecting a single input x_0 directly to the output. The input x_0 is either a deterministic value of zero or one. Thus, the probability p_{i_0} of the input x_0 being a one is either 0 or 1. The choice of setting p_{i_0} to 0 or 1 depends on

[5] In particular, a field-programmable gate array (FPGA) can be configured for the task. For an FPGA with n-input lookup tables, the i-th configuration bit of the table would be set to z_i, for $i = 0, 1, \ldots, 2^n - 1$.

which one is closer to the value q: If $q < 1 - q$, we set p_{i_0} to 0; otherwise, we set it to 1. In order for the circuit C_0 to realize the exact probability q, there is an ideal value $p^*_{i_0}$ that should replace the value p_{i_0}. It is not hard to see that $p^*_{i_0} = q$.

Now we assume that the Boolean function of the circuit C_k $(0 \leq k \leq n-1)$ is $f_k(x_0, x_1, \ldots, x_k)$ and the input probabilities are $P(x_0 = 1) = p_{i_0}, P(x_1 = 1) = p_{i_1}, \ldots, P(x_k = 1) = p_{i_k}$, where $p_{i_0} \in \{0, 1\}$ and $p_{i_1}, \ldots, p_{i_k} \in S$. Let $p^*_{i_k}$ be an ideal value such that if we replace p_{i_k} by $p^*_{i_k}$ and keep the remaining input probabilities unchanged then the output probability of C_k is exactly equal to q.

Our idea for constructing the circuit C_{k+1} is to replace the input x_k of the circuit C_k with a single gate with inputs x_k and x_{k+1}. Thus, the Boolean function of the circuit C_{k+1} is

$$f_{k+1}(x_0, \ldots, x_{k+1}) = f_k(x_0, \ldots, x_{k-1}, g_{k+1}(x_k, x_{k+1})),$$

where $g_{k+1}(x_k, x_{k+1})$ is a Boolean function on two variables. We keep the probabilities of the inputs $x_0, x_1, \ldots, x_k$ the same as those of the circuit C_k. We choose the probability of the input x_{k+1} from the remaining choices of the set S such that the output probability of the newly added single gate is the closest to $p^*_{i_k}$. Assume that the probability of the input x_{k+1} is $p_{i_{k+1}}$. In order to construct the circuit C_{k+2} in the same way, we also calculate an ideal probability $p^*_{i_{k+1}}$ such that if we replace $p_{i_{k+1}}$ by $p^*_{i_{k+1}}$ and keep the remaining input probabilities unchanged then the output probability of the circuit C_{k+1} is exactly equal to q.

To make things easy, we only consider AND gates and OR gates as the choices for the newly added gate. The choice depends on whether $p^*_{i_k} > p_{i_k}$. When $p^*_{i_k} > p_{i_k}$, we choose an OR gate to replace the input x_k of the circuit C_k. The first input of the OR gate connects to x_k and the second to x_{k+1} or to the negation of x_{k+1}. The probability of the input x_k is kept as p_{i_k}. The probability of the input x_{k+1} is chosen from the set $S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. Thus, the first input probability of the OR gate is p_{i_k} and the second is chosen from the set

$$S_{k+1} = \{p | p = p_j \text{ or } 1 - p_j, p_j \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}\}.$$

For an OR gate with two input probabilities a and b, its output probability is

$$a + b - ab = a + (1 - a)b.$$

The second input probability of the OR gate is chosen as p in the set S_{k+1} such that the output probability of the OR gate $p_{i_k} + (1 - p_{i_k})p$ is the closest to $p^*_{i_k}$. Equivalently, p is the value in the set S_{k+1} that is the closest to the value $\dfrac{p^*_{i_k} - p_{i_k}}{1 - p_{i_k}}$. We have two cases for p:

1. The case where $p = p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the OR gate to be x_{k+1} and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ should set the output probability of the OR gate to be $p^*_{i_k}$, so it satisfies that
$$p_{i_k} + (1 - p_{i_k})p^*_{i_{k+1}} = p^*_{i_k}, \tag{4.13}$$
or
$$p^*_{i_{k+1}} = \frac{p^*_{i_k} - p_{i_k}}{1 - p_{i_k}}.$$

2. The case where $p = 1 - p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the OR gate to be $\neg x_{k+1}$ and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ should set the output probability of the OR gate to be $p^*_{i_k}$, so it satisfies that
$$p_{i_k} + (1 - p_{i_k})(1 - p^*_{i_{k+1}}) = p^*_{i_k}, \tag{4.14}$$
or
$$p^*_{i_{k+1}} = \frac{1 - p^*_{i_k}}{1 - p_{i_k}}.$$

When $p^*_{i_k} \leq p_{i_k}$, we choose an AND gate to replace the input x_k of the circuit C_k. The first input of the AND gate connects to x_k and the second connects to x_{k+1} or to the negation of x_{k+1}. The probability of the input x_k is kept as p_{i_k}. The probability of the input x_{k+1} is chosen from the set $S\backslash\{p_{i_1}, \ldots, p_{i_k}\}$. Similar to the case where $p^*_{i_k} > p_{i_k}$, the second input probability of the AND gate is chosen as a value p in the set S_{k+1} such that the value $p \cdot p_{i_k}$ is the closest to $p^*_{i_k}$. Equivalently, p is the value in the set S_{k+1} that is the closest to the value $\dfrac{p^*_{i_k}}{p_{i_k}}$. We have two cases for p:

1. The case where $p = p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S\backslash\{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the AND gate to be x_{k+1} and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ satisfies $p^*_{i_{k+1}} = \dfrac{p^*_{i_k}}{p_{i_k}}$.
2. The case where $p = 1 - p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S\backslash\{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the AND gate to be $\neg x_{k+1}$ and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ satisfies
$$p^*_{i_{k+1}} = 1 - \frac{p^*_{i_k}}{p_{i_k}}.$$

Iteratively, using the procedure above, we can construct circuits $C_1, C_2, \ldots, C_n$. Finally, we construct a circuit C_{n+1}, which is built from C_n by replacing its input x_n with an OR gate or an AND gate with two inputs x_n and x_{n+1}. We keep the probabilities of the inputs $x_0, \ldots, x_n$ the same as those of the circuit C_n. The input x_{n+1} is set to a deterministic value of zero or one. Thus, the probability of the input x_{n+1} is either zero or or one. The choice of either an OR gate or an AND gate depends on whether $p^*_{i_n} > p_{i_n}$. When $p^*_{i_n} > p_{i_n}$, we choose an OR gate. The ideal probability value for the input x_{n+1} is
$$p^*_{i_{n+1}} = \frac{p^*_{i_n} - p_{i_n}}{1 - p_{i_n}}. \tag{4.15}$$
When $p^*_{i_n} \leq p_{i_n}$, we choose an AND gate. The ideal probability value for the input x_{n+1} is
$$p^*_{i_{n+1}} = \frac{p^*_{i_n}}{p_{i_n}}. \tag{4.16}$$

The choice of setting the input x_{n+1} to a deterministic value of zero or one depends on which one is closer to the value $p^*_{i_{n+1}}$: If $|p^*_{i_{n+1}}| < |1 - p^*_{i_{n+1}}|$, then we set the input x_{n+1} to zero; otherwise, we set it to one.

There is no evidence to show that the difference between the output probability of the circuit and q decreases as the number of inputs increases. Thus, we choose the one with the smallest difference among the circuits $C_0, \ldots, C_{n+1}$ as the final construction. It is easy to see that this algorithm completes in $O(n^2)$ time. For all $1 \leq k \leq n+1$, the circuit C_k has k fanin-two gates. Thus, the final solution contains at most $n+1$ fanin-two logic gates.

The following theorem shows that the difference between the target probability q and the output probability of the circuit synthesized by the greedy algorithm is bounded.

Theorem 11

In Scenario Two, given a set $S = \{p_1, p_2, \ldots, p_n\}$ and a target probability q, let p be the output probability of the circuit constructed by the greedy algorithm. We have

$$|p - q| \leq \frac{1}{2} \prod_{k=1}^{n} \max\{p_k, 1 - p_k\}. \qquad \square$$

Please see Appendix C for the proof.

Example 10

Given a set of input probabilities $S = \{0.4, 0.7, 0.8\}$ and a target probability $q = 0.63$, we show how to synthesize a circuit to generate the target probability based on the greedy algorithm.

For the circuit C_0, since $1 - q < q$, we set its input x_0 to be a deterministic value of one, or, equivalently, $p_{i_0} = 1$. The circuit C_0 is shown in Figure 4.8(a). The ideal value $p^*_{i_0} = q = 0.63$.

Since $p^*_{i_0} < p_{i_0}$, to construct the circuit C_1, we replace the input x_0 of the circuit C_0 by an AND gate. The probability of the first input of the AND gate is 1. The *probability p of the second input of the AND gate is chosen from the set*

$$S_1 = \{0.2, 0.3, 0.4, 0.6, 0.7, 0.8\}$$

so that it is the closest to the value $p^*_{i_0}/p_{i_0} = 0.63$. Thus, we choose $p = 0.6$. Notice that $p = 1 - 0.4$. We set the second input of the AND gate to be $\neg x_1$ and set its probability as $P(x_1 = 1) = p_{i_1} = 0.4$. The ideal value $p^*_{i_1} = 1 - p^*_{i_0}/p_{i_0} = 0.37$. The circuit C_1 is shown in Figure 4.8(b). (Again, we use a black dot to represent an inverter.)

Iteratively, we can get circuit C_2, C_3, and C_4 as those shown in Figures 4.8(c), (d), and (e), respectively. The ideal values are $p^*_{i_2} = 0.925$, $p^*_{i_3} = 0.625$, and $p^*_{i_4} = 0.893$. The circuit whose output probability is the closest to the target probability 0.63 is the circuit C_3. Thus, we choose C_3 as the final construction. Since the input x_0 of C_3 is a deterministic value of one, we can further optimize C_3. The final result is shown in Figure 4.8(f). $\square$

4.4 Scenario Three: Set S is not Specified and the Elements Cannot Be Duplicated

In Scenario Two, when solving the optimization problem, the minimal difference $\left|\sum_{i=0}^{2^n-1} z_i r_i - q\right|$ is actually a function of q, which we denote as $h(q)$. That is,

$$h(q) = \min_{\forall i, z_i \in \{0,1\}} \left| \sum_{i=0}^{2^n-1} z_i r_i - q \right|. \tag{4.17}$$

Assume that q is uniformly distributed on the unit interval. The mean of $h(q)$ for $q \in [0, 1]$ is solely determined by the set S. We can see that the smaller the mean is, the better the set S is for generating arbitrary probabilities. Thus, the mean of $h(q)$ is a good measure for the *quality* of S. We will denote it as $H(S)$. That is,

$$H(S) = \int_0^1 h(q)\,\mathrm{d}q. \tag{4.18}$$

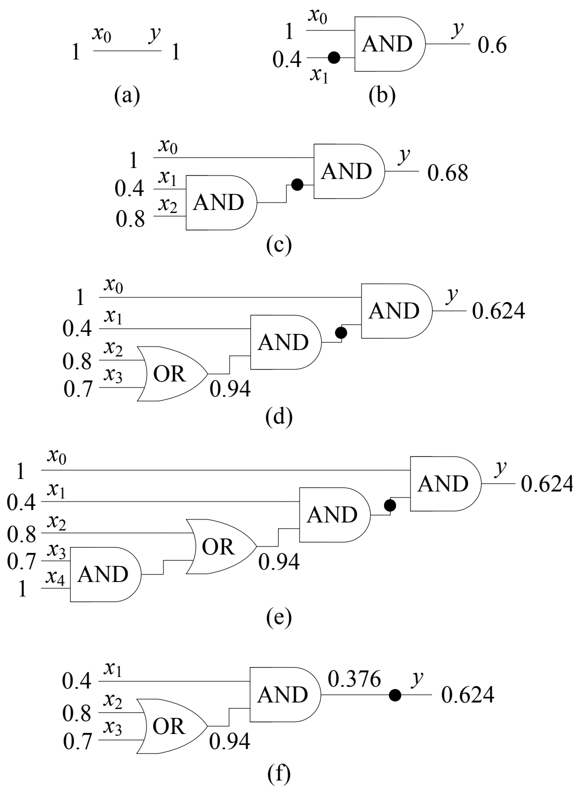


Figure 4.8: A group of circuits synthesized by the greedy algorithm to generate the target probability $q = 0.63$ from the set of input probability $S = \{0.4, 0.7, 0.8\}$. The black dots in the figure represent inverters. (a): The circuit C_0. (b): The circuit C_1. (c): The circuit C_2. (d): The circuit C_3. (e): The circuit C_4. (f): The final construction.

The problem considered in this scenario is: given an integer n, choose the n elements of the set S so that they produce a minimal $H(S)$.

Note that the only difference between Scenario Two and Scenario Three is that in Scenario Three, we are able to choose the elements of S. When constructing circuits, each element of S is still constrained to be used no more than once. As in Scenario Two, we are constructing a circuit with n inputs to realize each target probability. A circuit with n inputs has a truth table of 2^n rows. There are a total of 2^{2^n} different truth tables for n inputs. For a given assignment of input probabilities, we can get 2^{2^n} output probabilities.

Example 11

Consider the truth table shown in Table 4.3. Here, we assume that $P(x = 1) = 4/5$ *and* $P(y = 1) = 2/3$*. The corresponding probability of each input combination is given in the fourth column. For different assignments* $(z_0z_1z_2z_3)$ *of the output column, we obtain different output probabilities. For example, if* $(z_0z_1z_2z_3) = (1010)$*, then the output probability is* $5/15$*; if* $(z_0z_1z_2z_3) = (1011)$*, then the output probability is* $13/15$*. There are 16 different assignments for* $(z_0z_1z_2z_3)$*, so we can get 16 output probabilities. In this example, they are* $0, 1/15, \ldots, 14/15$ *and* 1. □

Table 4.3: A truth table for two variables. The output column $(z_0z_1z_2z_3)$ has a total of 16 different assignments.

x	y	z	Probability
0	0	z_0	1/15
0	1	z_1	2/15
1	0	z_2	4/15
1	1	z_3	8/15

Let $N = 2^{2^n}$. For a set S with n elements, call the N possible probability values $b_1, b_2, \ldots, b_N$ and assume that they are arranged in increasing order. That is $b_1 \leq b_2 \leq \cdots \leq b_N$. Note that if the output column of the truth table consists of all zeros, the output probability is 0. If it consists of all ones, the output probability is 1. Thus, we have $b_1 = 0$ and $b_N = 1$.

The first question is: what is a lower bound for $H(S)$? We have the following theorem.

Theorem 12

A lower bound for $H(S)$ is $\dfrac{1}{4(N-1)}$. $\square$

Proof. Note that for a q satisfying $b_i \le q \le \dfrac{b_i + b_{i+1}}{2}$, $h(q) = q - b_i$; for a q satisfying $\dfrac{b_i + b_{i+1}}{2} < q \le b_{i+1}$, $h(q) = b_{i+1} - q$. Thus,

$$\begin{aligned} H(S) &= \int_0^1 h(q)\,\mathrm{d}q \\ &= \sum_{i=1}^{N-1}\left(\int_{b_i}^{\frac{b_i+b_{i+1}}{2}} (q-b_i)\,\mathrm{d}q + \int_{\frac{b_i+b_{i+1}}{2}}^{b_{i+1}} (b_{i+1}-q)\,\mathrm{d}q\right) \\ &= \frac{1}{4}\sum_{i=1}^{N-1}(b_{i+1}-b_i)^2. \end{aligned} \tag{4.19}$$

Let $c_i = b_{i+1} - b_i$, for $i = 1, \ldots, N-1$. Since $\sum_{i=1}^{N-1} c_i = b_N - b_1 = 1$, by the Cauchy-Schwarz inequality, we have

$$H(S) = \frac{1}{4}\sum_{i=1}^{N-1} c_i^2 \ge \frac{1}{4(N-1)}\left(\sum_{i=1}^{N-1} c_i\right)^2 = \frac{1}{4(N-1)}. \qquad \square$$

The second question is: can this lower bound for $H(S)$ be achieved? We will show that the lower bound is achieved for the set

$$S = \{p | p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \ldots, n-1\}. \tag{4.20}$$

Lemma 3

For a truth table on the inputs $x_1, \ldots, x_n$ arranged in the order $x_n, \ldots, x_1$, let

$$P(x_k = 1) = \frac{2^{2^{k-1}}}{2^{2^{k-1}}+1}, \text{ for } k = 1, \ldots, n.$$

The probability of the i-th input combination $(0 \leq i \leq 2^n - 1)$ *is* $\dfrac{2^i}{2^{2^n}-1}$. □

PROOF. We prove the lemma by induction on n.

Base case: When $n = 1$, by assumption, $P(x_1 = 1) = \dfrac{2}{3}$. The 0-th input combination is $x_1 = 0$ and has probability

$$\frac{1}{3} = \frac{2^0}{2^{2^n}-1}.$$

The first input combination is $x_1 = 1$ and has probability

$$\frac{2}{3} = \frac{2^1}{2^{2^n}-1}.$$

Inductive step: Assume that the statement holds for $(n-1)$. Denote the probability of the i-th input combination in the truth table of n variables as $p_{i,n}$. By the induction hypothesis, for $0 \leq i \leq 2^{n-1} - 1$,

$$p_{i,n-1} = \frac{2^i}{2^{2^{n-1}}-1}.$$

Consider the truth table of n variables. Note that the input probabilities for $x_1, \ldots, x_{n-1}$ are the same as those in the case of $(n-1)$ and $P(x_n = 1) = \dfrac{2^{2^{n-1}}}{2^{2^{n-1}}+1}$.

When $0 \leq i \leq 2^{n-1} - 1$, the i-th row of the truth table has $x_n = 0$; the assignment to the rest of the variables is the same as the i-th row of the truth table of $(n-1)$ variables. Thus,

$$p_{i,n} = P(x_n = 0) \cdot p_{i,n-1} = \frac{1}{2^{2^{n-1}}+1} \cdot \frac{2^i}{2^{2^{n-1}}-1} = \frac{2^i}{2^{2^n}-1}. \tag{4.21}$$

When $2^{n-1} \leq i \leq 2^n - 1$, the i-th row of the truth table has $x_n = 1$; the assignment to the rest of the variables is the same as the $(i - 2^{n-1})$-th row of the truth table of $(n-1)$ variables. Thus,

$$p_{i,n} = P(x_n = 1) \cdot p_{i-2^{n-1},n-1} = \frac{2^{2^{n-1}}}{2^{2^{n-1}}+1} \cdot \frac{2^{i-2^{n-1}}}{2^{2^{n-1}}-1} = \frac{2^i}{2^{2^n}-1}. \tag{4.22}$$

Combining Equation (4.21) and (4.22), the statement holds for n. Thus, the statement in the lemma holds for all n. □

Based on Lemma 3, we will show that the set S in Equation (4.20) achieves the lower bound for $H(S)$.

Theorem 13
The set $S = \{p | p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \ldots, n-1\}$ *achieves the lower bound* $\frac{1}{4(N-1)}$ *for* $H(S)$. $\square$

Proof. By Lemma 3, for the given set S, the probability of the i-th input combination $(0 \le i \le 2^n - 1)$ is $\frac{2^i}{2^{2^n}-1}$. Therefore, the set of $N = 2^{2^n}$ possible probabilities is

$$R = \{p | p = \sum_{i=0}^{2^n-1} z_i \frac{2^i}{2^{2^n}-1}, z_i \in \{0, 1\}, \forall i = 0, \ldots, 2^n - 1\}.$$

It is not hard to see that the N possible probabilities in increasing order are

$$b_0 = 0, b_1 = \frac{1}{N-1}, \ldots, b_i = \frac{i}{N-1}, \ldots, b_{N-1} = 1.$$

(Example 11 shows the situation for $n = 2$. We can see that with the set $S = \{2/3, 4/5\}$, we can get 16 possible probabilities: $0, 1/15, \ldots, 14/15$ and 1.)
Thus, by Equation (4.19), we have $H(S) = \frac{1}{4(N-1)}$. $\square$

To summarize, if we have the freedom to choose n real numbers for the set S of source probabilities but each number can be used only once, the best choice is

$$S = \{p | p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \ldots, n-1\}.$$

With the optimal set S, the truth table for a target probability q is easy to determine. First, round q to the closest fraction in the form of $\frac{i}{2^{2^n}-1}$. Suppose the closest fraction is $\frac{g(q)}{2^{2^n}-1}$. Then, the output of the i-th row of the truth table is set as the i-th least significant digit of the binary representation of $g(q)$. Again, a circuit implementing this solution can be readily synthesized.

4.5 Related Work

The problem of synthesizing circuits to transform a given set of probabilities into a new set of probabilities appears in an early set of papers by Gill [31, 32]. He focused on synthesizing *sequential state machines* for this task.

Motivated by problems in neural computation, Jeavons *et al.* considered the problem of transforming stochastic binary sequences through what they call "local algorithms:" fixed functions applied to concurrent bits in different sequences [30]. This is equivalent to performing operations on stochastic bit streams with combinational logic, so in essence they were considering the same problem as we are. Their main result was a method for generating binary sequences with probability $\frac{m}{n^d}$ from a set of stochastic binary sequences with probabilities in the set $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$. This is equivalent to our Theorem 9. In contrast to the work of Jeavons *et al.*, our primary focus is on minimizing the number of source probabilities needed to realize arbitrary base-n fractional probabilities.

The proponents of PCMOS discussed the problem of synthesizing combinational logic to transform probability values [7]. These authors suggested using a tree-based circuit to realize a set of target probabilities. This was positioned as future work; no details were given.

Wilhelm and Bruck proposed a general framework for synthesizing *switching circuits* to achieve a desired probability [28]. Switching circuits were originally discussed by Shannon [33]. These consist of relays that are either open or closed; the circuit computes a logical value of one if there exists a closed path through the circuit. Wilhelm and Bruck considered *stochastic* switching circuits, in which each switch has a certain probability of being open or closed. They proposed an algorithm that generates the requisite stochastic switching circuit to compute any *binary* probability.

Zhou and Bruck generalized Wilhelm and Bruck's work [34]. They considered the problem of synthesizing a stochastic switching circuit to realize an arbitrary base-n fractional probability $\frac{m}{n^d}$ from a probabilistic switch set $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$. They showed that when n is a multiple of 2 or 3, such a realization is possible. However, for any prime number n greater than 3, there exists a base-n fractional probability that cannot be realized by any stochastic switching circuit.

In contrast to the work of Gill, to that of Wilhelm and Bruck, and to that of Zhou and Bruck, we consider combinational circuits: memoryless circuits consisting of logic gates. Our approach dovetails nicely with the circuit-level PCMOS constructs. It is orthogonal to the switch-based approach of Zhou and Bruck. Note that Zhou and Bruck assume that the probabilities in the given set S can be duplicated. We also consider the case where they cannot.

Chapter 5

Synthesizing Two-Level Logic to Generate Probabilities

In the previous chapter, we described a method for synthesizing combinational logic to transform a set of source probabilities into different target probabilities. We demonstrated that we are able to design a circuit to realize a required output probability. However, the circuit synthesized by our algorithm is not guaranteed to be an optimal design. In this chapter, we tackle a more challenging task: optimizing the circuits that compute on stochastic bit streams. We aim at minimizing the area of the circuits. Such a problem is vastly different from the traditional circuit optimization problem. Traditional optimization techniques at the logical level manipulate different logic implementations of the same Boolean function. In a sense, the traditional optimization algorithms search among a large number of logical implementations of a given Boolean function and choose one that results the optimal area. For example, in two-level logic optimization, the synthesis routine searches over different sum-of-product representations of a Boolean function and tries to find one with a minimum number of product terms [35]. It should be noted that the traditional logic optimization does not change the underlying Boolean function: its mere object is to find a good implementation of that Boolean function.

These traditional optimization techniques are not sufficient to fully minimize the area of the circuits that compute on stochastic bit streams. In fact, a target probability can be realized by two circuits with different Boolean functions. Figure 5.1 shows two circuits generating the output probability 0.3 from the input probabilities 0.4 and 0.5. Obviously, the two circuits have different Boolean functions; they even have different numbers of inputs! In terms of area, the circuit in Figure 5.1(b) is superior to that in Figure 5.1(a).

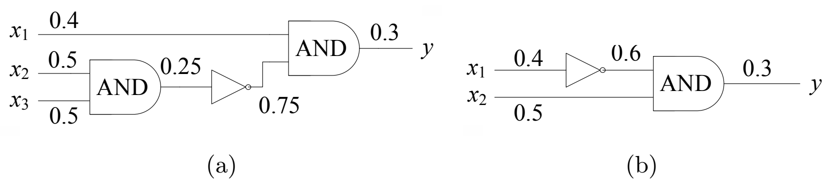


Figure 5.1: Two circuits generating the output probability 0.3 from the source probabilities 0.4 and 0.5.

Thus, optimization for circuits operating on random bit streams mandates entirely new techniques for manipulating Boolean functions. In searching for an optimal solution, the algorithm must cut across rigid boundaries of different Boolean functions, examining many different candidate solutions that generate the required output probability.

In this chapter, we focus on a fundamental problem pertaining to generating probabilities: synthesizing an optimal circuit that generates an arbitrary given probability value from a set of independent probabilistic inputs, each with an unbiased probability value of 0.5.

5.1 Arithmetic Two-Level Minimization Problem

Given combinational logic with a single output and n inputs, if all inputs independently have probability 0.5 of being one, then each input combination has probability

$\frac{1}{2^n}$ of occurring. If the output of the combinational circuit represents a Bolean function with exactly m minterms, then the probability that the output is one is $\frac{m}{2^n}$. Thus, the set of probabilities that can be realized by this model is $\{s | s = \frac{k}{2^n}, k = 0, 1, \ldots, 2^n\}$.

Example 12

Table 5.1 shows a truth table for a Boolean function

$$y = x_0 x_1 \vee x_2.$$ [1]

The last column shows the probability of each input combination occurring. If each input variable has probability of 0.5 *of being one, each input combination has probability of* 1/8 *of occurring. Since the Boolean function contains 5 minterms, the probability of* y *being one is* 5/8. □

Table 5.1: A truth table for the Boolean function $y = x_0 x_1 \vee x_2$. The last column shows the probability of each input combination occurring, under the assumption that each input variable has probability 0.5 of being one.

x_0	x_1	x_2	y	Probability
0	0	0	0	1/8
0	0	1	1	1/8
0	1	0	0	1/8
0	1	1	1	1/8
1	0	0	0	1/8
1	0	1	1	1/8
1	1	0	1	1/8
1	1	1	1	1/8

We consider the synthesis problem of implementing a probability $\frac{m}{2^n}$, where $0 \leq m \leq 2^n$ is an arbitrarily given integer. To implement the probability $\frac{m}{2^n}$, we can simply choose m input combinations and set their output values to be one. However, there are

[1] In this chapter, we use the common notation to represent logical AND and logical negation: we represent the AND of x and y as xy and the negation of x as $\bar{x}$. However, we will use $\vee$ to represent logical OR, since later we will use $+$ to represent *arithmetic* addition.

many ways to choose the m input combinations out of a total of 2^n input combinations; different choices may result in vastly different complexity of implementation. This motivates a new and interesting problem in logic synthesis:

What is the optimal way to synthesize logic that covers exactly m minterms if the choice of which m minterms are covered does not matter?

The complexity of a logic circuit depends on its implementation. In this chapter, we focus on the two-level implementation of logic circuit [35]. Since two-level logic synthesis plays an important role in multilevel logic synthesis [36], we believe that first understanding the two-level version of the synthesis problem will facilitate future research in tackling the multilevel version.

Minimizing the area of the two-level implementation is equivalent to minimizing the number of product terms of the sum-of-product (SOP) representation of a Boolean function [37]. Thus, the problem, which we will refer to as the *arithmetic two-level minimization problem*, can be formulated as:

Given integers $n > 0$ and $0 \leq m \leq 2^n$, find an n-variable Boolean function f with exactly m minterms and having a sum-of-product expression with the minimum number η of products.

Example 13
Suppose that we want to synthesize a 4-variable Boolean function with 7 minterms. This is equivalent to filling in the Karnaugh map of 4 variables with exactly 7 ones. Figure 5.2 shows two different ways to fill ones in. The optimal SOP Boolean expression for the function shown in Figure 5.2(a) is

$$\bar{x}_0\bar{x}_2 \vee \bar{x}_0 x_3 \vee x_1 x_2 x_3,$$

which contains three product terms. The optimal SOP Boolean expression for the function shown in Figure 5.2(b) is

$$\bar{x}_0\bar{x}_2 \vee x_1x_3,$$

which contains two product terms.

We can see that different choices for filling in 7 ones in the Karnaugh map can lead to optimal SOP Boolean expressions with different numbers of product terms. In this example, the second way to fill ones in is better than the first one. Indeed, it is an optimal filling of 7 ones that gives an SOP Boolean expression with the minimum number of products. □

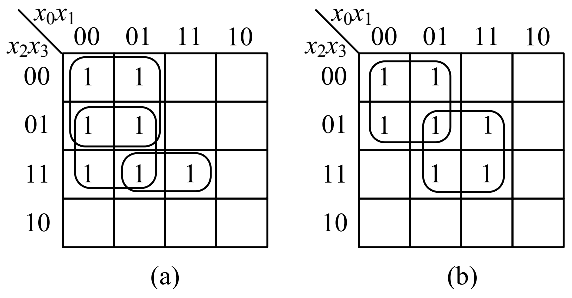


Figure 5.2: The Karnaugh maps of two different Boolean functions both containing 7 minterms: (a) The optimal SOP expression is $\bar{x}_0\bar{x}_2 \vee \bar{x}_0x_3 \vee x_1x_2x_3$, which contains three product terms. (b) The optimal SOP expression is $\bar{x}_0\bar{x}_2 \vee x_1x_3$, which contains two product terms.

In traditional two-level logic synthesis, a Boolean product term is also known as a cube. In what follows, we will refer to a "product term" as a cube. For convenience, we introduce the following notation.

Definition 6

Define $V(f)$ to be the number of minterms contained in a Boolean function f. □

For the arithmetic two-level minimization problem, our proposed solution is based on the inclusion-exclusion principle:

Given λ cubes $c_0, \ldots, c_{\lambda-1}$, the number of minterms cover by the union of the λ cubes is

$$\begin{aligned} V\left(\bigvee_{i=0}^{\lambda-1} c_i\right) &= \sum_{i=0}^{\lambda-1} V(c_i) - \sum_{\substack{i,j: \\ 0\le i<j\le\lambda-1}} V(c_i c_j) \\ &+ \sum_{\substack{i,j,k: \\ 0\le i<j<k\le\lambda-1}} V(c_i c_j c_k) - \cdots + (-1)^{\lambda-1} V\left(\prod_{i=0}^{\lambda-1} c_i\right). \end{aligned} \tag{5.1}$$

The right-hand side of Equation (5.1) contains a set of numbers, each of which corresponds to the number of minterms covered by the intersection of one of the subsets of the set of cubes $c_0, \ldots, c_{\lambda-1}$. We refer to this set of numbers as an *intersection pattern* of the set of cubes. For example, given a set of three cubes c_0, c_1, and c_2, its intersection pattern consists of numbers $V(c_0)$, $V(c_1)$, $V(c_2)$, $V(c_0c_1)$, $V(c_0c_2)$, $V(c_1c_2)$, and $V(c_0c_1c_2)$.

We intend to apply a search-based approach to solve the arithmetic two-level minimization problem. Initially, we will set λ to be a lower bound on the number of cubes to cover m minterms [38]. Then we will test whether we can find λ cubes so that they cover m minterms. In order to do so, we will first construct an intersection pattern such that the sum of the elements in that pattern according to Equation (5.1) equals the target value m. Then, we need to check whether we can find λ cubes to satisfy that intersection pattern. If we find a solution, then we obtain an optimal solution to the arithmetic two-level minimization problem. If not, we will try another intersection pattern on λ cubes. After a number of unsuccessful trials, we will increase λ by one.

Example 14

Synthesize an optimal SOP Boolean expression on 4 *variables to cover* 11 *minterms.*

Since we cannot cover 11 *minterms with just* 1 *cube, the lower bound on the number of cubes is* 2. *Thus, initially, we set $\lambda = 2$. For $\lambda = 2$, we first construct an intersection* *pattern* $\{V(c_0), V(c_1), V(c_0c_1)\}$, *so that*

$$V(c_0) + V(c_1) - V(c_0c_1) = 11.$$

One intersection pattern that satisfies the above equation has elements as $V(c_0) = 8$, $V(c_1) = 4$, *and* $V(c_0c_1) = 1$. *However, we cannot find two cubes to satisfy this intersection pattern. Thus, we will try other intersection patterns on* 2 *cubes which cover* 11 *minterms. Indeed, there are no other intersection patterns on* 2 *cubes to cover* 11 *minterms. Then, we raise* λ *to* 3.

For $\lambda = 3$, *we first construct intersection pattern*

$$\{V(c_0), V(c_1), V(c_2), V(c_0c_1), V(c_0c_2), V(c_1c_2), V(c_0c_1c_2)\},$$

so that

$$\begin{aligned} V(c_0) + V(c_1) + V(c_2) - V(c_0c_1) - V(c_0c_2) \\ - V(c_1c_2) + V(c_0c_1c_2) = 11. \end{aligned}$$

One intersection pattern that satisfies the above equation has elements as $V(c_0) = 8$, $V(c_1) = 2$, $V(c_2) = 1$, *and* $V(c_0c_1) = V(c_0c_2) = V(c_1c_2) = V(c_0c_1c_2) = 0$. *We could synthesize cubes* $c_0 = x_0$, $c_1 = \bar{x}_0x_1x_2$, *and* $c_2 = \bar{x}_0\bar{x}_1\bar{x}_2x_3$ *to satisfy the given intersection pattern. Thus, we get an optimal solution of* 3 *cubes to the original arithmetic two-level minimization problem.* □

5.2 λ-Cube Intersection Problem

A crucial step for our proposed solution to the arithmetic two-level minimization problem is to answer the following question: given a set of numbers that corresponds to an intersection pattern of λ cubes, how can one synthesize a set of λ cubes to satisfy the given intersection pattern, or prove that there is no solution to the given intersection pattern? We will call this problem as the λ-*cube intersection problem.* It is the focus of this chapter.

Example 15

In a 3-cube intersection problem on 4 variables x_0, x_1, x_2, x_3, if we are given the intersection pattern as

$$V(c_0) = 4, V(c_1) = 8, V(c_2) = 4,$$
$$V(c_0c_1) = V(c_0c_2) = V(c_1c_2) = 2, V(c_0c_1c_2) = 1,$$

we can synthesize cubes $c_0 = x_0x_1$, $c_1 = x_2$, and $c_2 = x_1x_3$ to satisfy the intersection pattern.

In another 2-cube intersection problem on 4 variables $x_0, \ldots, x_3$, if we are given the intersection pattern as

$$V(c_0) = 4, V(c_1) = 8, V(c_0c_1) = 1,$$

it is easily seen from the Karnaugh map on 4 variables that there does not exist a set of 2 cubes to satisfy the given intersection pattern. □

In the rest of this section, we will first introduce some basic definitions. Then, we will give a formal definition of the λ-cube intersection problem. Some of the basic definitions are adopted from [39].

The n *variables* of a Boolean function are denoted by $x_0, \ldots, x_{n-1}$. For a variable x, x and $\bar{x}$ are referred to as *literals*. A *cube*, denoted by c, is a conjunction of literals such that x and $\bar{x}$ do not appear simultaneously. For example, $x_1\bar{x}_2\bar{x}_3$ is a cube. A *minterm* is a cube in which each of the n variables appear exactly once, in either its complemented or uncomplemented form. If cube c_2 takes the value one whenever cube c_1 equals one, we say that cube c_1 *implies* cube c_2 and write as $c_1 \subseteq c_2$. If cube c_1 implies cube c_2, then we have $V(c_1) \leq V(c_2)$. If $c_1 \cdot c_2 = 0$, we say that cube c_1 and cube c_2 are *disjoint*.

If a cube c contains k literals ($0 \leq k \leq n$), then the number of minterms contained in the cube is $V(c) = 2^{n-k}$. Note that when a cube contains 0 literals, it is a special cube $c = 1$, which contains all minterms in the entire Boolean space. There is another special cube called *empty cube*, which is $c = 0$. The number of minterms contained in an empty cube is $V(c) = 0$. Thus, the number of minterms contained in a cube is in the set $S = \{s | s = 0 \text{ or } s = 2^k, k = 0, 1, \ldots, n\}$.

To make the representation compact, we use the following definitions.

Definition 7

Given two integers A and B, let their binary representation be $A = \sum_{i=0}^{k-1} a_i 2^i$ and $B = \sum_{i=0}^{k-1} b_i 2^i$, where $a_i, b_i \in \{0, 1\}$. We write $A \succeq B$ if for all $0 \leq i \leq k-1$, $a_i \geq b_i$; we write $A \preceq B$ if for all $0 \leq i \leq k-1$, $a_i \leq b_i$. $\Box$

Definition 8

Given a cube c and a $\gamma \in \{0, 1\}$, define

$$c^\gamma = \begin{cases} 1, & \text{if } \gamma = 0 \\ c, & \text{if } \gamma = 1. \end{cases}$$

Given a set of λ cubes $c_0, \ldots, c_{\lambda-1}$ and an integer $\Gamma = \sum_{i=0}^{\lambda-1} \gamma_i 2^i$, where $\gamma_i \in \{0, 1\}$, define C^Γ to be the intersection of a subset of cubes c_i's for those i's such that $\gamma_i = 1$, i.e., $C^\Gamma = \prod_{i=0}^{\lambda-1} c_i^{\gamma_i}$. $\Box$

Definition 9

For an integer $a \geq 0$, define $||a||$ to be the number of ones in the binary representation of a. More formally, suppose that a can be represented as $a = \sum_{i=0}^{k-1} a_i 2^i$ with all $a_i \in \{0, 1\}$. Then, $||a|| = \sum_{i=0}^{k-1} a_i$. $\Box$

For example, $||7|| = 1 + 1 + 1 = 3$.

With the above definition, we can more formally define the λ-cube intersection problem as follows:

Given $n > 0$, $\lambda > 0$, and a vector of 2^λ numbers $(v_0, v_1, \ldots v_{2^\lambda - 1})$, determine whether there exists a set of λ cubes $c_0, \ldots, c_{\lambda-1}$ on n variables $x_0, \ldots, x_{n-1}$, such that for all

$0 \leq \Gamma \leq 2^\lambda - 1$, $V(C^\Gamma) = v_\Gamma$.

We refer to the vector of numbers $(v_0, \ldots, v_{2^\lambda - 1})$ as *an intersection pattern* on λ cubes, or simply as an intersection pattern. If a set of λ cubes $c_0, \ldots, c_{\lambda-1}$ satisfies the property that for any $0 \leq \Gamma \leq 2^\lambda - 1$, $V(C^\Gamma) = v_\Gamma$, then we say that the set of cubes satisfies the intersection pattern $(v_0, \ldots, v_{2^\lambda - 1})$.

If there exists a set of λ cubes to satisfy the intersection pattern, then for all $0 \leq \Gamma \leq 2^\lambda - 1$, we have

$$v_\Gamma = V(C^\Gamma) \in S = \{s | s = 0 \text{ or } s = 2^k, k = 0, 1, \ldots, n\}.$$

Further, the number $v_0 = V(C^0) = V(1) = 2^n$. Thus, in the remaining of the chapter, we will only consider the instances of the problem with $v_0 = 2^n$ and $v_1, \ldots, v_{2^\lambda - 1} \in S$. For the other instances of the problem, it is obvious that no solution exists. Since it is more meaningful to consider a set of nonempty cubes $c_0, \ldots, c_{\lambda-1}$, we assume that for any $0 \leq i \leq \lambda - 1$, $v_{2^i} > 0$.

Based on the given intersection pattern, we define some sets as follows.

Definition 10

Let the set P be the set of numbers Γ such that $v_\Gamma > 0$ and let the set Z be the set of numbers Γ such that $v_\Gamma = 0$, i.e.,

$$P = \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1 \text{ and } v_\Gamma > 0\},$$
$$Z = \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1 \text{ and } v_\Gamma = 0\}.$$

For any $0 \leq i \leq \lambda$, let the set P_i be the set of numbers Γ such that the number of ones in the binary representation of Γ is i and $v_\Gamma > 0$; let the set Z_i be the set of Γ such that the number of ones in the binary representation of Γ is i and $v_\Gamma = 0$, i.e.,

$$P_i = \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, ||\Gamma|| = i, \text{ and } v_\Gamma > 0\},$$
$$Z_i = \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, ||\Gamma|| = i, \text{ and } v_\Gamma = 0\}. \qquad \square$$

From the definition of P and Z, we have the following straightforward lemma, which gives a necessary condition on the existence of λ cubes to satisfy the given intersection pattern.

Lemma 4

If a set of λ cubes $c_0, \ldots, c_{\lambda-1}$ satisfies the given intersection pattern, then for any $\Gamma \in P$, $C^\Gamma \neq 0$ and for any $\Gamma \in Z$, $C^\Gamma = 0$. $\square$

For any $\Gamma \in P$, we define a number k_Γ as follows.

Definition 11

For any $\Gamma \in P$, define $k_\Gamma = \log_2(v_\Gamma)$. $\square$

Since we assume that $v_\Gamma \in S = \{s | s = 0 \text{ or } s = 2^k, k = 0, 1, \ldots, n\}$, thus for any $\Gamma \in P$, k_Γ is an integer and $0 \leq k_\Gamma \leq n$. Note that since $v_0 = 2^n$, we have $k_0 = n$.

For convenience, we represent a cube as a *cube-variable row vector* and a set of cubes as a *cube-variable matrix*. These are defined as follows.

Definition 12

Given a nonempty cube c on n variables $x_0, \ldots, x_{n-1}$, we represent it by a cube-variable row vector *U of length n, whose elements are from the set $\{0, 1, *\}$. If the j-th ($0 \leq j \leq n-1$) element $U_j = 1$, then the literal x_j appears in the cube c; if $U_j = 0$, then the literal $\bar{x}_j$ appears in the cube c; if $U_j = *$, then the cube c does not depend on the variable x_j, i.e., neither literal x_j nor literal $\bar{x}_j$ appears in the cube c.*

Given a set of λ nonempty cubes $c_0, \ldots, c_{\lambda-1}$ on n variables $x_0, \ldots, x_{n-1}$, we represent them by a cube-variable matrix *D of size $\lambda \times n$, so that the i-th row of the matrix is the* cube-variable row vector *of c_i.* $\square$

For example, a set of two cubes $c_0 = x_0\bar{x}_1$ and $c_1 = \bar{x}_0 x_2$ is represented as a cube-variable matrix

$$\begin{bmatrix} 1 & 0 & * \\ 0 & * & 1 \end{bmatrix}$$

Given a cube-variable row vector, the following simple lemma suggests how to obtain the number of minterms covered by the corresponding cube.

Lemma 5

If the cube-variable row vector of a nonempty cube contains k $*$'s, then the cube covers 2^k number of minterms. $\square$

Definition 13

For a value a in $\{0, 1, *\}$, the negation of a is defined as follows:

$$\bar{a} = \begin{cases} 1, & \textit{if } a = 0 \\ 0, & \textit{if } a = 1 \\ *, & \textit{if } a = *. \end{cases}$$

The negation of a cube-variable matrix (column vector) is the element-wise negation of the matrix (column vector). $\square$

In what follows, we will say that a cube-variable matrix satisfies the given intersection pattern if the corresponding set of cubes satisfies the intersection pattern. The following lemma is straightforward.

Lemma 6

Suppose that a cube-variable matrix D satisfies the intersection pattern $(v_0, \ldots, v_{2^\lambda - 1})$. Then D' satisfies the same intersection pattern if D' is obtained from D by column permutation or column negation. $\square$

Before we go through the details of our proposed solution, we will briefly talk about the basic idea of our solution. Our solution is a column-based method: synthesizing a cube-variable matrix is equivalent to determining what each column of the matrix should be. Since each entry of the matrix is in the set $\{0, 1, *\}$, each column, which has λ entries, has a total of 3^λ choices. Indeed, by the symmetry between different column choices and the disjoint relation among some cubes, we only need to consider a small subset of all 3^λ column choices as the candidate choices. Furthermore, by Lemma 6, since the order of the column does not matter, we only need to determine the number of occurrences of each candidate column choice in the cube-variable matrix, which we treat as unknowns. We establish a system of equations over those unknowns and the given intersection pattern. The λ-cube intersection problem can be solved by finding a non-negative solution to the system of equations.

5.3 A Special Case of the λ-Cube Intersection Problem

Here we consider a specific case in which $v_{2^\lambda-1} > 0$. First, we have the following theorem, which gives a necessary condition for the existence of a cube-variable matrix to satisfy the given intersection pattern.

Theorem 14

If $v_{2^\lambda-1} > 0$ and there exists a cube-variable matrix to satisfy the λ-cube intersection problem, then for any $0 \leq \Gamma \leq 2^\lambda - 1$, $\Gamma \in P$. $\square$

PROOF. Based on Definition 8, for any $0 \leq \Gamma \leq 2^\lambda - 1$, we have $C^{2^\lambda-1} \subseteq C^\Gamma$. Therefore,

$$0 < v_{2^\lambda-1} = V(C^{2^\lambda-1}) \leq V(C^\Gamma) = v_\Gamma.$$

By the definition of the set P, we have $\Gamma \in P$. $\square$

In what follows, we will assume that there exists a cube-variable matrix D to satisfy the given intersection pattern. Without loss of generality, we can assume that each entry of the cube-variable matrix is either 1 or $*$. Since $\prod_{i=0}^{\lambda-1} c_i \neq 0$, no column of the matrix D simultaneously contains both a 0 and a 1. Otherwise, $\prod_{i=0}^{\lambda-1} c_i = 0$. Therefore, each column of the matrix D contains either only 0's and $*$'s or only 1's and $*$'s. By Lemma 6, if we negate those columns of the matrix D that contain only 0's and $*$'s, then we obtain a new matrix D' which still satisfies the given intersection pattern. Note that the matrix D' only contains 1's and $*$'s. Thus, we could assume that each column of the cube-variable matrix is in the set $\{1,*\}^\lambda$. The set $\{1,*\}^\lambda$ contains 2^λ elements. We denote those elements by $\psi_0, \psi_1, \ldots, \psi_{2^\lambda-1}$ as follows:

Definition 14

Given any $0 \leq \Gamma \leq 2^\lambda - 1$, suppose that $\Gamma = \sum_{i=0}^{\lambda-1} \gamma_i 2^i$, where $\gamma_i \in \{0,1\}$. Define ψ_Γ to be a column vector of length λ with entries from the set $\{1,*\}$, such that the i-th element $(0 \leq i \leq \lambda - 1)$ of it is

$$(\psi_\Gamma)_i = \begin{cases} 1, & \text{if } \gamma_i = 0 \\ *, & \text{if } \gamma_i = 1. \end{cases}$$

Define the set $\Psi = \{\psi_0, \psi_1, \ldots, \psi_{2^\lambda-1}\}$. $\square$

For example, if $\lambda = 3$, then $\psi_0 = (1,1,1)^T$ and $\psi_5 = (*,1,*)^T$.[2]

The basic idea of our proposed solution is to determine which column patterns from the set Ψ should be present in the cube-variable matrix. Indeed, as pointed out at the end of Section 5.2, we only need to determine how many column patterns of the form ψ_Γ are present in the matrix. We define the number of occurrences of column pattern ψ_Γ as z_Γ.

Definition 15

For any $0 \leq \Gamma \leq 2^\lambda - 1$, define J_Γ to be the set of indices of the columns in the matrix D of the form ψ_Γ, i.e., $J_\Gamma = \{j | D_{\cdot j} = \psi_\Gamma\}$. Define z_Γ to be the cardinality of the set J_Γ.
$\square$

In the special case, if there exists a cube-variable matrix to satisfy the intersection pattern, then based on Theorem 14, we have $P = \{0, 1, \ldots, 2^\lambda - 1\}$. Thus, based on Definition 11, we have a set of numbers $k_0, \ldots, k_{2^\lambda-1}$. The following theorem gives relation between $\{z_0, \ldots z_{2^\lambda-1}\}$ and $\{k_0, \ldots, k_{2^\lambda-1}\}$.

[2] The superscript T here means the transpose of a matrix.

Theorem 15

If there exists a cube-variable matrix D to satisfy the intersection pattern, then for all $0 \leq L \leq 2^\lambda - 1$, we have

$$k_L = \sum_{0 \leq \Gamma \leq 2^{\lambda-1} : \Gamma \succeq L} z_\Gamma. \tag{5.2}$$

□

Proof. Since the total number of columns in matrix D is n, we have $\sum_{\Gamma=0}^{2^\lambda - 1} z_\Gamma = n = k_0$, or

$$\sum_{0 \leq \Gamma \leq 2^\lambda - 1 : \Gamma \succeq 0} z_\Gamma = k_0.$$

Thus, Equation (5.2) holds for $L = 0$.

Now consider $1 \leq L \leq 2^\lambda - 1$. Then L can be represented as $L = \sum_{j=0}^{r-1} 2^{l_j}$, where $1 \leq r \leq \lambda$ and $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$. Then, C^L represents the intersection of the set of cubes $c_{l_0}, \ldots, c_{l_{r-1}}$. The i-th entry in the cube-variable row vector of the intersection C^L is $*$ if and only if the column $D_{\cdot i}$ has $*$'s on the row $l_0, l_1, \ldots, l_{r-1}$. Therefore, on the one hand, the number of $*$'s in the cube-variable row vector of the intersection C^L is the number of columns in D whose entries on the row $l_0, l_1, \ldots, l_{r-1}$ are all $*$'s, or mathematically, the sum

$$\sum_{\substack{0 \leq \Gamma \leq 2^\lambda - 1: \\ (\psi_\Gamma)_{l_0} = \cdots = (\psi_\Gamma)_{l_{r-1}} = *}} z_\Gamma.$$

On the other hand,by Lemma 5, since $V(C^L) = v_L = 2^{k_L}$, the number of $*$'s in the cube-variable row vector of C^L is k_L. Therefore, we have

$$k_L = \sum_{\substack{0 \leq \Gamma \leq 2^\lambda - 1: \\ (\psi_\Gamma)_{l_0} = \cdots = (\psi_\Gamma)_{l_{r-1}} = *}} z_\Gamma = \sum_{\substack{0 \leq \Gamma \leq 2^\lambda - 1, \\ \Gamma = \sum_{i=0}^{\lambda-1} \gamma_i 2^i: \\ \gamma_{l_0} = \cdots = \gamma_{l_{r-1}} = 1}} z_\Gamma, \tag{5.3}$$

By Definition 7, we can rewrite Equation (5.3) as

$$k_L = \sum_{0 \leq \Gamma \leq 2^{\lambda-1} : \Gamma \succeq L} z_\Gamma. \quad \square$$

Note that Equation (5.2) is a linear equation on $z_0, \ldots, z_{2^\lambda-1}$ and holds for all $0 \le L \le 2^\lambda - 1$. Therefore, we can derive a system of 2^λ linear equations on unknowns $z_0, \ldots, z_{2^\lambda-1}$:

$$\sum_{0 \le \Gamma \le 2^\lambda-1 : \Gamma \succeq L} z_\Gamma = k_L, \text{ for } L = 0, 1, \ldots, 2^\lambda - 1. \tag{5.4}$$

We can represent the above system of linear equations in matrix form, as shown by the following theorem.

Theorem 16

Let vector $\vec{k} = (k_0, \ldots, k_{2^\lambda-1})^T$ and vector $\vec{z} = (z_0, \ldots, z_{2^\lambda-1})^T$. Then we can represent the system of 2^λ linear equations (5.4) in matrix form as

$$R_\lambda \vec{z} = \vec{k}, \tag{5.5}$$

where R_λ is a $2^\lambda \times 2^\lambda$ square matrix defined recursively as follows:

$$R_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, R_i = \begin{bmatrix} R_{i-1} & R_{i-1} \\ 0 & R_{i-1} \end{bmatrix}, \text{ for } i = 2, \ldots, \lambda. \quad \square$$

Proof. For convenience, we use $\vec{z}[j, k]$ ($0 \le j \le k \le 2^\lambda - 1$) to represent the column vector $(z_j, \ldots, z_k)^T$.

We claim that given any $1 \le i \le \lambda$, the set of 2^i linear expressions

$$\sum_{0 \le \Gamma \le 2^i-1 : \Gamma \succeq L} z_\Gamma, \text{ for } L = 0, 1, \ldots, 2^i - 1$$

can be represented in matrix form as

$$R_i \vec{z}[0, 2^i - 1].$$

We prove this claim by induction on i.

Base case: When $i = 1$, the set of 2 linear expressions

$$\begin{cases} \sum_{0 \le \Gamma \le 1 : \Gamma \succeq 0} z_\Gamma \\ \sum_{0 \le \Gamma \le 1 : \Gamma \succeq 1} z_\Gamma \end{cases}$$

is

$$\begin{cases} z_0 + z_1 \\ z_1 \end{cases}$$

Therefore, in the matrix form, the set of expressions can be represented as

$$R_1 \vec{z}[0, 1].$$

Inductive step: Assume that the claim holds for i. Now consider the set of 2^{i+1} linear expressions

$$\sum_{0 \le \Gamma \le 2^{i+1}-1 : \Gamma \succeq L} z_\Gamma, \text{ for } L = 0, 1, \ldots, 2^{i+1} - 1.$$

For any $0 \le L \le 2^{i+1} - 1$, we have

$$\begin{aligned} \sum_{\substack{0 \le \Gamma \le 2^{i+1}-1: \\ \Gamma \succeq L}} z_\Gamma &= \sum_{\substack{0 \le \Gamma \le 2^{i}-1: \\ \Gamma \succeq L}} z_\Gamma + \sum_{\substack{2^i \le \Gamma \le 2^{i+1}-1: \\ \Gamma \succeq L}} z_\Gamma \\ &= \sum_{\substack{0 \le \Gamma \le 2^{i}-1: \\ \Gamma \succeq L}} z_\Gamma + \sum_{\substack{0 \le \Gamma \le 2^{i}-1: \\ (\Gamma + 2^i) \succeq L}} z_{\Gamma + 2^i}. \end{aligned} \tag{5.6}$$

When $0 \le L \le 2^i - 1$, it is not hard to see that

$$\{\Gamma | 0 \le \Gamma \le 2^i - 1, (\Gamma + 2^i) \succeq L\} = \{\Gamma | 0 \le \Gamma \le 2^i - 1, \Gamma \succeq L\}.$$

Thus, from Equation (5.6), for any $0 \le L \le 2^i - 1$, we have

$$\sum_{0 \le \Gamma \le 2^{i+1}-1 : \Gamma \succeq L} z_\Gamma = \sum_{0 \le \Gamma \le 2^{i}-1 : \Gamma \succeq L} z_\Gamma + \sum_{0 \le \Gamma \le 2^{i}-1 : \Gamma \succeq L} z_{\Gamma + 2^i}.$$

By the induction hypothesis, the first 2^i expressions

$$\sum_{0 \le \Gamma \le 2^{i+1}-1 : \Gamma \succeq L} z_\Gamma, \text{ for } L = 0, \ldots, 2^i - 1$$

can be represented in matrix form as

$$R_i\vec{z}[0, 2^i - 1] + R_i\vec{z}[2^i, 2^{i+1} - 1]. \tag{5.7}$$

When $2^i \le L \le 2^{i+1} - 1$, it is not hard to see that

$$\{\Gamma | 0 \le \Gamma \le 2^i - 1, \Gamma \succeq L\} = \phi,$$
$$\{\Gamma | 0 \le \Gamma \le 2^i - 1, (\Gamma + 2^i) \succeq L\} = \{\Gamma | 0 \le \Gamma \le 2^i - 1, \Gamma \succeq (L - 2^i)\}.$$

Therefore, from Equation (5.6), for any $2^i \le L \le 2^{i+1} - 1$, we have

$$\sum_{0 \le \Gamma \le 2^{i+1}-1 : \Gamma \succeq L} z_\Gamma = \sum_{0 \le \Gamma \le 2^i - 1 : \Gamma \succeq (L - 2^i)} z_{\Gamma + 2^i}.$$

Note that $0 \le L - 2^i \le 2^i - 1$. Thus, by the induction hypothesis, the last 2^i expressions

$$\sum_{0 \le \Gamma \le 2^{i+1}-1 : \Gamma \succeq L} z_\Gamma, \text{ for } L = 2^i, \ldots, 2^{i+1} - 1$$

can be represented in matrix form as

$$R_i\vec{z}[2^i, 2^{i+1} - 1]. \tag{5.8}$$

Based on Equation (5.7) and (5.8), the set of linear expressions

$$\sum_{0 \le \Gamma \le 2^{i+1}-1 : \Gamma \succeq L} z_\Gamma, \text{ for } L = 0, \ldots, 2^{i+1} - 1$$

can be represented in matrix form as

$$\begin{bmatrix} R_i & R_i \\ 0 & R_i \end{bmatrix} \begin{bmatrix} \vec{z}[0, 2^i - 1] \\ \vec{z}[2^i, 2^{i+1} - 1] \end{bmatrix} = R_{i+1}\vec{z}[0, 2^{i+1} - 1].$$

Therefore, the claim holds for $i + 1$. Thus, by induction, the claim holds for all $i = 1, 2, \ldots, \lambda$.

Thus, the system of linear equations

$$\sum_{0 \le \Gamma \le 2^\lambda - 1 : \Gamma \succeq L} z_\Gamma = k_L, \text{ for } L = 0, 1, \ldots, 2^\lambda - 1.$$

can be represented in matrix form as

$$R_\lambda \vec{z} = \vec{k}. \qquad \square$$

It is not hard to see that $\det(R_\lambda) = 1$. Therefore, R_λ is invertible. The following theorem shows what R_λ^{-1} is.

Theorem 17

R_λ^{-1} is recursively defined as follows:

$$R_1^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}, R_i^{-1} = \begin{bmatrix} R_{i-1}^{-1} & -R_{i-1}^{-1} \\ 0 & R_{i-1}^{-1} \end{bmatrix}, \text{ for } i = 2, \ldots, \lambda. \quad \square$$

Proof. We only need to show that for $i = 1, \ldots, \lambda$, $R_i^{-1} R_i = I_{2^i}$. We prove this claim by induction on i.

Base case: When $i = 1$,

$$R_1^{-1} R_1 = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Inductive step: Assume the claim holds for i. Then, based on the induction hypothesis,

$$R_{i+1}^{-1} R_{i+1} = \begin{bmatrix} R_i^{-1} & -R_i^{-1} \\ 0 & R_i^{-1} \end{bmatrix} \begin{bmatrix} R_i & R_i \\ 0 & R_i \end{bmatrix} = \begin{bmatrix} I_{2^i} & 0 \\ 0 & I_{2^i} \end{bmatrix} = I_{2^{i+1}}.$$

Therefore, the claim holds for $i + 1$. Thus, by induction, the claim holds for all $i = 1, \ldots, \lambda$. $\square$

Therefore, given $k_0, k_1, \ldots, k_{2^\lambda - 1}$, we can get $z_0, z_1, \ldots, z_{2^\lambda - 1}$ as $\vec{z} = R_\lambda^{-1} \vec{k}$.

Since for any $0 \leq \Gamma \leq 2^\lambda - 1$, z_Γ is the cardinality of the set J_Γ, therefore, z_Γ must be a non-negative integer. By Theorem 17, R_λ^{-1} is an integer matrix. Therefore, $z_0, \ldots, z_{2^\lambda - 1}$ are always integers. Thus, a necessary condition for the existence of λ

cubes to satisfy the given intersection pattern is that the vector $R_\lambda^{-1}\vec{k}$ has all entries non-negative. On the other hand, from Equation (5.5), we can see that the intersection pattern $(2^{k_0}, \ldots, 2^{k_{2^\lambda-1}})$ only depends on $z_0, \ldots, z_{2^\lambda-1}$. Therefore, as long as the vector $R_\lambda^{-1}\vec{k}$ has all entries non-negative, there exist λ cubes to satisfy the given intersection pattern. In fact, we can construct λ cubes with their cube-variable matrix as follows: for any column $0 \le j \le n-1$ of D, we can find a $0 \le \Gamma \le 2^\lambda - 1$ such that $\sum_{i=0}^{\Gamma-1} z_i \le j \le \sum_{i=0}^{\Gamma} z_i - 1$. Then, we let $D_{\cdot j} = \psi_\Gamma$. In summary, we have the following corollary.

Corollary 3

The necessary and sufficient condition for the existence of λ cubes to satisfy the given intersection pattern is that the vector $R_\lambda^{-1}\vec{k}$ has all entries non-negative, where $\vec{k} = (k_0, k_1, \ldots, k_{2^\lambda-1})^T$ and R_λ^{-1} is defined in Theorem 17. $\square$

Example 16

Given $v_0 = 32, v_1 = 16$, $v_2 = 16$, $v_3 = 8$, $v_4 = 8$, $v_5 = 4$, $v_6 = 4$, and $v_7 = 2$, determine whether there exists a set of three cubes c_0, c_1, and c_2 on 5 variables that satisfies the intersection pattern $(v_0, \ldots, v_7)$.

Solution: *From the given coditions, we have*

$$\vec{k} = (5, 4, 4, 3, 3, 2, 2, 1)^T.$$

Since

$$R_3^{-1} = \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

then by Equation (5.5), *we get*

$$\vec{z} = (0,0,0,2,0,1,1,1)^T.$$

Therefore, there are two ψ_3's, one ψ_5, one ψ_6, and one ψ_7 in the cube-variable matrix of c_0, c_1, and c_2. One realization of the cube-variable matrix is

$$\begin{bmatrix} * & * & * & 1 & * \\ * & * & 1 & * & * \\ 1 & 1 & * & * & * \end{bmatrix}$$

and the corresponding cubes are $c_0 = x_3$, $c_1 = x_2$, and $c_2 = x_0 \wedge x_1$. $\Box$

5.4 General λ-Cube Intersection Problem

In this section, we consider the more general situation where $v_{2^\lambda - 1} \geq 0$.

5.4.1 Necessary Conditions on the Positive v_Γ's

We first have the following theorem applicable for numbers $v_\Gamma > 0$.

Theorem 18

Suppose that there exist λ cubes $c_0, \ldots, c_{\lambda-1}$ to satisfy the intersection pattern. For any $0 \leq L \leq 2^\lambda - 1$, if $v_L > 0$, then for any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq L$, we have $v_\Gamma > 0$. $\Box$

Proof. For any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq L$, it is not hard to see that $C^L \subseteq C^\Gamma$. Therefore, $0 < v_L = V(C^L) \leq V(C^\Gamma) = v_\Gamma$. $\Box$

If a set of cubes is pairwise non-disjoint, then it has the following property.

Lemma 7

If a set of r cubes $c_{l_0}, \ldots, c_{l_{r-1}}$ $(3 \leq r \leq \lambda, 0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1)$ is pairwise non-disjoint, i.e., for any $0 \leq i < j \leq r-1$, $c_{l_i} \cdot c_{l_j} \neq 0$, then their intersection $\prod_{i=0}^{r-1} c_{l_i}$ is nonempty. $\Box$

Proof. By contraposition, suppose that $\prod_{i=0}^{r-1} c_{l_i} = 0$. Consider the cube-variable matrix on these r cubes. Since their intersection is empty, there exists a column in the matrix that contains both a 0 and a 1. The cube corresponding to the 0 entry and the cube corresponding to the 1 entry are disjoint. This contradicts the assumption that the given set of cubes is pairwise non-disjoint. □

Alternatively, Lemma 7 can be stated on the numbers v_Γ. This gives a necessary condition for the existence of a set of cubes to satisfy the given intersection pattern.

Theorem 19

Suppose that there exist λ cubes $c_0, \ldots, c_{\lambda-1}$ to satisfy the given intersection pattern. If a set of r $(3 \leq r \leq \lambda)$ numbers $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$ satisfies that for any $0 \leq i < j \leq r-1$, $v_{(2^{l_i}+2^{l_j})} > 0$, then for $L = \sum_{i=0}^{r-1} 2^{l_i}$, $v_L > 0$. □

For example, suppose that in a 4-cube intersection problem we are given $v_3 > 0$, $v_9 > 0$, and $v_{10} > 0$. If there exist 4 cubes to satisfy the given intersection pattern, then since $V(c_0c_1) > 0$, $V(c_0c_3) > 0$, and $V(c_1c_3) > 0$, we must have $v_{11} = V(c_0c_1c_3) > 0$.

If both the conditions in Theorem 18 and 19 are satisfied, then we have the following theorem, which will play an important role in proving the necessary and sufficient condition later.

Theorem 20

Suppose that the given intersection pattern satisfies that

1. *For any $0 \leq L \leq 2^\lambda - 1$, if $v_L > 0$, then for any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq L$, $v_\Gamma > 0$.*

2. *For any set of r $(3 \leq r \leq \lambda)$ numbers $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$, if it satisfies that for any $0 \leq i < j \leq r-1$, $v_{(2^{l_i}+2^{l_j})} > 0$, then for the number $L = \sum_{i=0}^{r-1} 2^{l_i}$, $v_L > 0$.*

Then, a necessary and sufficient condition for a set of λ nonempty cubes to satisfy the condition that for any $\Gamma \in P$, $C^\Gamma \neq 0$ and for any $\Gamma \in Z$, $C^\Gamma = 0$ is that for any $\Gamma \in P_2$, $C^\Gamma \neq 0$ and for any $\Gamma \in Z_2$, $C^\Gamma = 0$. $\Box$

Proof. The necessary part of the theorem is obvious, since the set P_2 is a subset of the set P and the set Z_2 is a subset of the set Z.

Now we prove the sufficient part. Suppose that a set of cubes satisfies that for any $\Gamma \in P_2$, $C^\Gamma \neq 0$ and for any $\Gamma \in Z_2$, $C^\Gamma = 0$.

It is not hard to see that the sets $P_0, \ldots, P_\lambda$ form a partition of the set P and that the sets $Z_0, \ldots, Z_\lambda$ form a partition of the set Z. Thus, we only need to prove that for all $0 \leq k \leq \lambda$, the set of cubes satisfies the condition that for any $\Gamma \in P_k$, $C^\Gamma \neq 0$ and for any $\Gamma \in Z_k$, $C^\Gamma = 0$.

We first consider the case that $k = 0$. By convention, $v_0 > 0$. Thus, $P_0 = \{0\}$ and $Z_0 = \phi$. Since $C^0 = 1$, thus we have that for any $\Gamma \in P_0$, $C^\Gamma \neq 0$. Since $Z_0 = \phi$, the statement that for any $\Gamma \in Z_0$, $C^\Gamma = 0$ also holds.

Now we consider the case that $k = 1$. Since we assume that for any $0 \leq i \leq \lambda - 1$, $v_{2^i} > 0$, therefore, $P_1 = \{2^i | i = 0, \ldots, \lambda - 1\}$ and $Z_1 = \phi$. Since $c_0, \ldots, c_{\lambda-1}$ are all nonempty, thus we have that for any $\Gamma \in P_1$, $C^\Gamma \neq 0$. Since $Z_1 = \phi$, the statement that for any $\Gamma \in Z_1$, $C^\Gamma = 0$ also holds.

When $k = 2$, the statement that the set of cubes satisfies that for any $\Gamma \in P_2$, $C^\Gamma \neq 0$ and for any $\Gamma \in Z_2$, $C^\Gamma = 0$ obviously holds.

Now we consider the case that $k \geq 3$. First, we consider any $L \in P_k$. Suppose that $L = \sum_{i=0}^{r-1} 2^{l_i}$, where $3 \leq r \leq \lambda$ and $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$. Then, for any $0 \leq i < j \leq r - 1$, $(2^{l_i} + 2^{l_j}) \preceq L$. Therefore, based on the given condition, we have $v_{(2^{l_i}+2^{l_j})} > 0$. Since $||2^{l_i} + 2^{l_j}|| = 2$, thus $(2^{l_i} + 2^{l_j}) \in P_2$. By the assumption that for any $\Gamma \in P_2$, $C^\Gamma \neq 0$, we have that $C^{(2^{l_i}+2^{l_j})} = c_{l_i} \cdot c_{l_j} \neq 0$. Thus, the r cubes $c_{l_0}, \ldots, c_{l_{r-1}}$ are pairwise non-disjoint. By Lemma 7, then $C^L = \prod_{i=0}^{r-1} c_{l_i} \neq 0$. Therefore, for any $L \in P_k$, $C^L \neq 0$.

Now we consider any $L \in Z_k$. Suppose that $L = \sum_{i=0}^{r-1} 2^{l_i}$, where $3 \le r \le \lambda$ and $0 \le l_0 < \cdots < l_{r-1} \le \lambda - 1$. We argue that there exist two numbers $0 \le u < v \le r-1$, such that $v_{(2^{l_u}+2^{l_v})} = 0$. Otherwise, for any $0 \le i < j \le r-1$, $v_{(2^{l_i}+2^{l_j})} > 0$. Then, based on the given conditions, we have $v_L > 0$. This contradicts the assumption that $L \in Z_k$. Thus, there exist two numbers $0 \le u < v \le r-1$, such that $v_{(2^{l_u}+2^{l_v})} = 0$. Since $||2^{l_u} + 2^{l_v}|| = 2$, thus $(2^{l_u} + 2^{l_v}) \in Z_2$. By the assumption that for any $\Gamma \in Z_2$, $C^\Gamma = 0$, we have that $C^{(2^{l_u}+2^{l_v})} = c_{l_u} \cdot c_{l_v} = 0$. Thus, $C^L = \prod_{i=0}^{r-1} c_{l_i} = 0$. Therefore, for any $L \in Z_k$, $C^L = 0$. □

5.4.2 Compatible Column Pattern Set

In the general case, the cube-variable matrix consists of 0, 1 and $*$ and so does each column of the matrix. There are a total of 3^λ different choices of patterns for each column. However, not all combinations of 0, 1 and $*$ as a column vector can be present in the matrix. For example, if the given intersection pattern indicates that $c_i \cdot c_j \neq 0$, then those column patterns that have a 0 on the i-th entry and a 1 on the j-th entry cannot be present in the matrix. On the other hand, some kinds of column patterns must be present at least once in the matrix. For example, if the given intersection pattern indicates that $c_i \cdot c_j = 0$, then at least one of the column patterns that have a 0 on the i-th entry and a 1 on the j-th entry or have a 1 on the i-th entry and a 0 on the j-th entry must be present in the matrix. In this section, we will show what kind of column patterns can be present in the matrix. For this purpose, we first introduce the *compatible column pattern set* for numbers $\Gamma \in Z_2$.

Definition 16

Suppose that $\Gamma \in Z_2$ and $\Gamma = 2^i + 2^j$, where $0 \le i < j \le \lambda - 1$. The compatible column pattern set for Γ is the set of column vectors W of length λ with entries from the set $\{0, 1, *\}$, such that

1. *$W_i = 0$ and $W_j = 1$ or $W_i = 1$ and $W_j = 0$,*

2. *for any number $L \in P_2$ such that $L = 2^k + 2^l$, where $0 \leq k < l \leq \lambda - 1$, the situation that $W_k = 0$ and $W_l = 1$ or $W_k = 1$ and $W_l = 0$ does not happen.* $\square$

It is not hard to see that if a cube-variable column vector is in the compatible column pattern set for a $\Gamma \in Z_2$, then the negation of that cube-variable column vector is also in that set. Therefore, we define the *representative compatible column pattern set* as follows.

Definition 17

The representative compatible column pattern set ρ_Γ for $\Gamma \in Z_2$ is a subset of the compatible column pattern set for Γ such that the first non-$*$ entry of each element in the representative set is 0. $\square$

Example 17

Consider a 4-cube intersection problem with

$$P_2 = \{(0011)_2, (0101)_2, (1001)_2\},$$
$$Z_2 = \{(0110)_2, (1010)_2, (1100)_2\}.$$

The compatible column pattern set for $\Gamma = (0110)_2 \in Z_2$ is

$$\{(*010)^T, (*101)^T, (*011)^T, (*100)^T, (*01*)^T, (*10*)^T\}.$$

The representative compatible column pattern set for $\Gamma = (0110)_2$ is

$$\{(*010)^T, (*011)^T, (*01*)^T\}. \qquad \square$$

Definition 18

We define the set Y as the union of the representative compatible column pattern sets ρ_Γ for all $\Gamma \in Z_2$, i.e., $Y = \bigcup_{\Gamma \in Z_2} \rho_\Gamma$. We define the set $F = Y \cup \Psi$. $\square$

The following lemma shows that only those column patterns in the set F are needed to construct the cube-variable matrix.

Lemma 8

If there exists a cube-variable matrix D to satisfy the given intersection pattern, then there exists another matrix D' which also satisfies the given intersection pattern and each column of which is in the set F. $\square$

Proof. First, we argue that for any column of D which contains both a 0 and a 1 entry, the column is in the compatible column pattern set of a certain $\Gamma \in Z_2$.

Suppose that a column r $(0 \leq r \leq n-1)$ of D has the i-th entry being 0 and the j-th entry being 1, where $0 \leq i, j \leq \lambda - 1$ and $i \neq j$. Then, $c_i \cdot c_j = 0$. Since the matrix D satisfies the given intersection pattern, we have $v_{2^i+2^j} = V(c_i \cdot c_j) = 0$. Therefore, the number $2^i + 2^j$ is in the set Z_2. Now consider any $L \in P_2$. Suppose that $L = 2^k + 2^l$, where $0 \leq k < l \leq \lambda - 1$. Since the necessary condition for the cube-variable matrix to satisfy a given intersection pattern is that for $L \in P_2$, $C^L \neq 0$, thus the situation that $D_{kr} = 0$ and $D_{lr} = 1$ or $D_{kr} = 1$ and $D_{lr} = 0$ cannot happen. Therefore, the column r of D is in the compatible column pattern set for the number $(2^i + 2^j) \in Z_2$.

We can construct a D' from D as follows. For any column $0 \leq r \leq \lambda - 1$:

1. If $D_{\cdot r}$ contains only 1's and $*$'s, we let $D'_{\cdot r}$ be $D_{\cdot r}$. Then $D'_{\cdot r}$ is in the set Ψ.

2. If $D_{\cdot r}$ contains only 0's and $*$'s, we let $D'_{\cdot r}$ be the negation of the column $D_{\cdot r}$. Then $D'_{\cdot r}$ is in the set Ψ.

3. If $D_{\cdot r}$ contains both a 0 and a 1 and the first non-$*$ entry of $D_{\cdot r}$ is 0, we let $D'_{\cdot r}$ be $D_{\cdot r}$. Then, there exists a $\Gamma \in Z_2$ such that $D'_{\cdot r}$ is in the set ρ_Γ.

4. If $D_{\cdot r}$ contains both a 0 and a 1 and the first non-$*$ entry of $D_{\cdot r}$ is 1, we let $D'_{\cdot r}$ be the negation of the column $D_{\cdot r}$. Then, there exists a $\Gamma \in Z_2$ such that $D'_{\cdot r}$ is in the set ρ_Γ.

Then, by the above construction, each column of D' is in the set F. Further, D' is obtained from D by column negations. Thus, by Lemma 6, D' also satisfies the given intersection pattern. □

Based on Lemma 8, we only need to answer whether there exists a cube-variable matrix with columns from the set F to satisfy the given intersection pattern. The following lemma states that if such a matrix exists, then for each $\Gamma \in Z_2$, at least one of the column pattern elements from the set ρ_Γ must be present in that matrix.

Lemma 9

If a cube-variable matrix D with columns from the set F satisfies the given intersection pattern, then for any $\Gamma \in Z_2$, there exists a column in D which is in the set ρ_Γ. □

PROOF. For any $\Gamma \in Z_2$, suppose that $\Gamma = 2^i + 2^j$, where $0 \leq i < j \leq \lambda - 1$. Since the cube-variable matrix satisfies the given intersection pattern, then based on Lemma 4, for the $\Gamma \in Z_2$, we must have $C^\Gamma = 0$ or $c_i \cdot c_j = 0$. Thus, there must exist a column r in D, such that $D_{ir} = 0$ and $D_{jr} = 1$ or $D_{ir} = 1$ and $D_{jr} = 0$. Now consider any $L \in P_2$. Suppose that $L = 2^k + 2^l$, where $0 \leq k < l \leq \lambda - 1$. Since the necessary condition for the cube-variable matrix to satisfy a given intersection pattern is that for the $L \in P_2$, $C^L \neq 0$, the situation that $D_{kr} = 0$ and $D_{lr} = 1$ or $D_{kr} = 1$ and $D_{lr} = 0$ cannot happen. Therefore, the column r of D is in the compatible column pattern set for Γ. Further, since all the columns of D are in the set F, then column r must be in the set ρ_Γ. □

5.4.3 A Necessary and Sufficient Condition

In this section, we will show a necessary and sufficient condition for the existence of a set of cubes to satisfy the given intersection pattern. As a byproduct, the proof provides a way of synthesizing a set of cubes to satisfy the given intersection pattern. Based on Lemma 8, we only need to consider cube-variable matrix that consists of column patterns from the set F. The basic idea to solve the general case problem is similar to that applied in the special case — we will establish relations between the numbers of occurrences of those elements of the set F in the cube-variable matrix and the k_Γ's. First, we define *root cube-variable matrix*, which links the general case problem to the special case problem we discussed in Section 5.3.

Definition 19

Given a cube-variable matrix D on λ cubes $c_0, \ldots, c_{\lambda-1}$, we define root cube-variable matrix *$t(D)$ of D as the cube-variable matrix formed by replacing the 0 entries in D with 1's and keeping the other entries in D unchanged. The set of cubes $c'_0, \ldots, c'_{\lambda-1}$ corresponding to the root matrix is called the set of* root cubes *to the original set of cubes.* $\Box$

For example, the root matrix of the cube-variable matrix

$$\begin{bmatrix} 1 & 0 & * \\ 0 & * & 1 \end{bmatrix} \quad \text{is} \quad \begin{bmatrix} 1 & 1 & * \\ 1 & * & 1 \end{bmatrix}.$$

The set of root cubes is $c'_0 = x_0x_1$ and $c'_1 = x_0x_2$.

Based on the definition of the set of root cubes, we have the following lemma.

Lemma 10

Suppose that the set of root cubes to the set of original cubes $c_0, \ldots, c_{\lambda-1}$ is $c'_0, \ldots, c'_{\lambda-1}$. Then, for any $\Gamma \in P$, we have $V(C'^{\Gamma}) = V(C^{\Gamma})$. $\Box$

Proof. If $\Gamma = 0$, then obviously, $V(C'^{0}) = 2^n = V(C^0)$. Now consider any $\Gamma \in P$ such that $\Gamma \neq 0$. Suppose that C^Γ represents the intersection of a set of cubes $c_{l_0}, \ldots, c_{l_{r-1}}$, where $1 \le r \le \lambda$ and $0 \le l_0 < \cdots < l_{r-1} \le \lambda - 1$. Let the cube-variable matrix corresponding to the set of cubes $c_{l_0}, \ldots, c_{l_{r-1}}$ be D_Γ and the cube-variable matrix corresponding to the set of cubes $c'_{l_0}, \ldots, c'_{l_{r-1}}$ be D'_Γ. Since $V(C^\Gamma) > 0$, the intersection of $c_{l_0}, \ldots, c_{l_{r-1}}$ is nonempty. Based on the definition of the set of root cubes, each column of the matrix D'_Γ contains only 1's and $*$'s. Therefore, the intersection of $c'_{l_0}, \ldots, c'_{l_{r-1}}$ is also nonempty. Since D'_Γ is the root matrix of D_Γ, the columns of D'_Γ that contain all $*$'s are in one-to-one correspondence to the columns of D_Γ that contain all $*$'s. Since the number of $*$'s in the cube-variable row vector of the nonempty intersection of a set of cubes equals the number of columns of the matrix that contain all $*$'s, the number of $*$'s in the cube-variable row vector of C'^{Γ} equals that in the cube-variable row vector of C^{Γ}. By Lemma 5, we have $V(C'^{\Gamma}) = V(C^{\Gamma})$. $\square$

Since the root matrix $t(D)$ is a matrix containing only 1's and $*$'s, we can apply the definition of z_Γ in Definition 15 to $t(D)$. Then, based on the fact that for any $\Gamma \in P$, $V(C'^{\Gamma}) = V(C^{\Gamma}) = 2^{k_\Gamma}$, it is not hard to show that the following theorem characterizing the relation between z_Γ's and k_L's holds.

Theorem 21

If there exist λ cubes to satisfy the given intersection pattern, then for any $L \in P$,

$$\sum_{0 \le \Gamma \le 2^\lambda - 1 : \Gamma \succeq L} z_\Gamma = k_L,$$

where z_Γ's are defined on the root matrix $t(D)$ according to Definition 15. $\square$

Following a similar definition for a root cube-variable matrix, we define a *root column vector* as follows.

Definition 20

Given a column vector W with each element in the set $\{0, 1, *\}$, define its root column vector $t(W)$ *as the column vector obtained from W by replacing the* 0 *entries in W with* 1*'s and keeping the other entries in W unchanged.* $\square$

Based on the definition of the root column vector, we can regroup the elements in the set Y according to their root column vectors, which results in the following definition. The relation between the elements in the set Y and their root column vectors will be used later to derive a set of inequalities on the numbers of occurrences of the elements of the set F in the cube-variable matrix (See Theorem 22).

Definition 21

We define the set M to be the set of numbers $0 \leq \Gamma \leq 2^{\lambda} - 1$ such that there exists an element in the set Y, whose root column vector is ψ_{Γ}, i.e.,

$$M = \{\Gamma | 0 \leq \Gamma \leq 2^{\lambda} - 1, \ s.t. \ \exists W \in Y \ s.t. \ t(W) = \psi_{\Gamma}\}.$$

Define $\overline{M}$ as $\overline{M} = \{\Gamma | 0 \leq \Gamma \leq 2^{\lambda} - 1, \Gamma \notin M\}$.

For any $\Gamma \in M$, we define the set Y_{Γ} to be the set of elements in the set Y such that their root column vectors are ψ_{Γ}, i.e., $Y_{\Gamma} = \{W | W \in Y \text{ and } t(W) = \psi_{\Gamma}\}$. $\Box$

Notice that the sets Y_{Γ} $(\Gamma \in M)$ form a partition of the set Y.

Example 18

For the intersection pattern shown in Example 17, we have $Z_2 = \{6, 10, 12\}$ and

$$\begin{aligned}
\rho_6 &= \{(*010)^T, (*011)^T, (*01*)^T\}, \\
\rho_{10} &= \{(*001)^T, (*011)^T, (*0*1)^T\}, \\
\rho_{12} &= \{(*010)^T, (*001)^T, (**01)^T\}.
\end{aligned}$$

Thus,

$$\begin{aligned}
Y &= \{(*010)^T, (*001)^T, (*011)^T, (**01)^T, (*0*1)^T, (*01*)^T\}, \\
M &= \{1, 3, 5, 9\},
\end{aligned}$$

and $Y_1 = \{(*010)^T, (*001)^T, (*011)^T\}$, $Y_3 = \{(**01)^T\}$, $Y_5 = \{(*0*1)^T\}$, and $Y_9 = \{(*01*)^T\}$. $\Box$

Based on Lemma 8, we can assume that each column of the cube-variable matrix is from the set $F = Y \cup \Psi$. To solve the general case problem, we only need to determine the number of occurrences of each element of the set F in the cube-variable matrix. In order to establish equations, we first define the number of occurrences of each element of the set Y in the cube-variable matrix, which is actually defined on each partition Y_{Γ} of Y, as stated by the following definition.

Definition 22

For any $\Gamma \in M$, we let the $|Y_\Gamma|$ elements in the set Y_Γ be $\delta_{\Gamma,0}, \ldots, \delta_{\Gamma,|Y_\Gamma|-1}$. For any $0 \leq i \leq |Y_\Gamma| - 1$, we define $K_{\Gamma,i}$ to be the set of indices of the columns in the matrix D of the form $\delta_{\Gamma,i}$, i.e., $K_{\Gamma,i} = \{k | D_{\cdot k} = \delta_{\Gamma,i}\}$. We define $w_{\Gamma,i}$ to be the cardinality of the set $K_{\Gamma,i}$. $\Box$

The following theorem establishes a set of linear inequalities on $w_{\Gamma,i}$'s and z_Γ's, where the z_Γ's are defined on the root matrix according to Definition 15.

Theorem 22

Suppose that there exists a cube-variable matrix D to satisfy the given intersection pattern, whose columns are from the set F. Then, we have that for any $\Gamma \in M$,

$$\sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \leq z_\Gamma, \tag{5.9}$$

where z_Γ's are defined on the root matrix $t(D)$ according to Definition 15. We also have that for any $L \in Z_2$,

$$\sum_{\substack{\Gamma \in M, 0 \leq i \leq |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} w_{\Gamma,i} \geq 1. \tag{5.10}$$

$\Box$

Proof. Consider any $\Gamma \in M$. For any number $k \in \bigcup_{i=0}^{|Y_\Gamma|-1} K_{\Gamma,i}$, the column vector $D_{\cdot k}$ is in the set Y_Γ. Thus, the root column vector of $D_{\cdot k}$ is ψ_Γ. Thus, $k \in J_\Gamma$, where J_Γ is defined on the root matrix $t(D)$. Therefore, $\bigcup_{i=0}^{|Y_\Gamma|-1} K_{\Gamma,i} \subseteq J_\Gamma$. As a result, we have

$$\left| \bigcup_{i=0}^{|Y_\Gamma|-1} K_{\Gamma,i} \right| \leq |J_\Gamma|,$$

or

$$\sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \leq z_\Gamma.$$

By Lemma 9, for any $L \in Z_2$, there exists a column in D which is in the set ρ_L. Suppose that column is of the form $\delta_{\Gamma^*,i^*} \in \rho_L$, where $\Gamma^* \in M$ and $0 \leq i \leq |Y_{\Gamma^*}| - 1$.

Thus,

$$1 \leq w_{\Gamma^*,i^*} \leq \sum_{\substack{\Gamma \in M, 0 \leq i \leq |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} w_{\Gamma,i}. \qquad \square$$

Example 19

For the intersection pattern given in Example 17, based on the result shown in Example 18, we have

$$\delta_{1,0} = (*010)^T, \delta_{1,1} = (*001)^T, \delta_{1,2} = (*011)^T,$$
$$\delta_{3,0} = (**01)^T, \delta_{5,0} = (*0*1)^T, \delta_{9,0} = (*01*)^T.$$

The set of equations (5.9) *for all* $\Gamma \in M$ *in this example is*

$$\begin{cases} w_{\Gamma,0} \leq z_\Gamma, \text{ for any } \Gamma \in \{3,5,9\} \\ w_{1,0} + w_{1,1} + w_{1,2} \leq z_1 \end{cases}$$

The set of equations (5.10) *for all* $L \in Z_2$ *in this example is*

$$\begin{cases} w_{1,0} + w_{1,2} + w_{9,0} \geq 1 \\ w_{1,1} + w_{1,2} + w_{5,0} \geq 1 \\ w_{1,0} + w_{1,1} + w_{3,0} \geq 1 \end{cases} \qquad \square$$

Finally, combining the conditions of Theorem 18, 19, 21, and 22, we can derive the following necessary and sufficient condition.

Theorem 23

There exists a cube-variable matrix D *to satisfy the given intersection pattern* $(v_0, \ldots, v_{2^\lambda-1})$ *if and only if*

1. *for any* $0 \leq L \leq 2^\lambda - 1$*, if* $v_L > 0$*, then for any* $0 \leq \Gamma \leq 2^\lambda - 1$ *such that* $\Gamma \preceq L$*,* $v_\Gamma > 0$*,*

2. *for any set of* r $(3 \le r \le \lambda)$ *numbers* $0 \le l_0 < \cdots < l_{r-1} \le \lambda - 1$, *if it satisfies that for any* $0 \le i < j \le r-1$, $v_{(2^{l_i}+2^{l_j})} > 0$, *then for the number* $L = \sum_{i=0}^{r-1} 2^{l_i}$, $v_L > 0$,

3. *the system of equations on unknowns* $\tilde{z}_\Gamma$ (*for all* $0 \le \Gamma \le 2^\lambda - 1$) *and* $\tilde{w}_{\Gamma,i}$ (*for all* $\Gamma \in M$ *and* $0 \le i \le |Y_\Gamma| - 1$)

$$\begin{aligned} \sum_{0 \le \Gamma \le 2^\lambda - 1 : \Gamma \succeq L} \tilde{z}_\Gamma &= k_L, \text{ for all } L \in P \\ \sum_{i=0}^{|Y_\Gamma|-1} \tilde{w}_{\Gamma,i} &\le \tilde{z}_\Gamma, \text{ for all } \Gamma \in M \\ \sum_{\substack{\Gamma \in M, 0 \le i \le |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} \tilde{w}_{\Gamma,i} &\ge 1, \text{ for all } L \in Z_2 \end{aligned} \tag{5.11}$$

has a non-negative integer solution. □

PROOF. "**only if**" part: Statement 1 in the theorem is due to Theorem 18 and Statement 2 in the theorem is due to Theorem 19.

Since D satisfies the given intersection pattern, then by Lemma 8, there exists another matrix D' which also satisfies the given intersection pattern and each column of which is in the set F. For any $0 \le \Gamma \le 2^\lambda - 1$, let $\tilde{z}_\Gamma = z_\Gamma$, where z_Γ's are defined on the root matrix $t(D')$ according to Definition 15. For any $\Gamma \in M$ and $0 \le i \le |Y_\Gamma| - 1$, let $\tilde{w}_{\Gamma,i} = w_{\Gamma,i}$, where $w_{\Gamma,i}$'s are defined on the matrix D' according to Definition 22. By Theorem 21 and 22, the set of numbers $\tilde{z}_\Gamma$ and $\tilde{w}_{\Gamma,i}$ satisfies the system of equations (5.11). Since $\tilde{z}_\Gamma$ is the cardinality of the set J_Γ and $\tilde{w}_{\Gamma,i}$ is the cardinality of the set $K_{\Gamma,i}$, therefore, $\tilde{z}_\Gamma$'s and $\tilde{w}_{\Gamma,i}$'s are all non-negative integers. Thus, the system of equations (5.11) has a non-negative solution.

"**if**" part: Let a non-negative solution to the system of equations (5.11) be $\tilde{z}_\Gamma = z_\Gamma$, for all $0 \le \Gamma \le 2^\lambda - 1$, and $\tilde{w}_{\Gamma,i} = w_{\Gamma,i}$, for all $\Gamma \in M$ and $0 \le i \le |Y_\Gamma| - 1$. Since for all $0 \le \Gamma \le 2^\lambda - 1$, $z_\Gamma \ge 0$, for all $\Gamma \in M$ and $0 \le i \le |Y_\Gamma| - 1$, $w_{\Gamma,i} \ge 0$, and for all $\Gamma \in M$, $\sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \le z_\Gamma$, then, we can construct a cube-variable matrix D so that

1. for all $\Gamma \in \overline{M}$, the matrix contains z_Γ columns of the form ψ_Γ,
2. for all $\Gamma \in M$, the matrix contains $z_\Gamma - \sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i}$ columns of the form ψ_Γ, and
3. for all $\Gamma \in M$ and all $0 \leq i \leq |Y_\Gamma| - 1$, the matrix contains $w_{\Gamma,i}$ columns of the form $\delta_{\Gamma,i}$.

All columns of the matrix D are in the set F. Next, we prove that the matrix D satisfies the given intersection pattern.

For any $L \in Z_2$, suppose $L = 2^i + 2^j$, where $0 \leq i < j \leq \lambda - 1$. Since

$$\sum_{\substack{\Gamma \in M, 0 \leq k \leq |Y_\Gamma|-1: \\ \delta_{\Gamma,k} \in \rho_L}} w_{\Gamma,k} \geq 1,$$

there exists a $\Gamma^* \in M$ and a $0 \leq k^* \leq |Y_{\Gamma^*}| - 1$, such that $\delta_{\Gamma^*,k^*} \in \rho_L$ and $w_{\Gamma^*,k^*} \geq 1$. Therefore, the matrix D contains a column from the set ρ_L. Based on the definition of ρ_L, $C^L = c_i \cdot c_j = 0$. Thus, for any $L \in Z_2$, $C^L = 0$.

Now consider any $L \in P_2$. Suppose $L = 2^i + 2^j$, where $0 \leq i < j \leq \lambda - 1$. We argue that $C^L = c_i \cdot c_j \neq 0$. Otherwise, $c_i \cdot c_j = 0$. Therefore, there exists a column r in D, such $D_{ir} = 0$ and $D_{jr} = 1$ or $D_{ir} = 1$ and $D_{jr} = 0$. Since all the columns of D are in the set F, thus the column $D_{\cdot r}$ must be in the set Y. However, based on the definition of representative compatible column pattern set, each element W in the set Y satisfies that for the $L \in P_2$, the situation that $W_i = 0$ and $W_j = 1$ or $W_i = 1$ and $W_j = 0$ does not happen. Therefore, the column $D_{\cdot r}$ does not belong to the set Y. We get a contradiction. Thus, for any $L \in P_2$, we have $C^L \neq 0$.

Since for any $\Gamma \in Z_2$, $C^\Gamma = 0$, for any $\Gamma \in P_2$, $C^\Gamma \neq 0$, and the given intersection pattern satisfies the conditions of Theorem 20, then, based on Theorem 20, we have that for any $\Gamma \in Z$, $C^\Gamma = 0$ and for any $\Gamma \in P$, $C^\Gamma \neq 0$. Thus, for all these $\Gamma \in Z$, $V(C^\Gamma) = v_\Gamma = 0$.

Now consider any $L \in P$. When $L = 0$, we have that $V(C^0) = 2^n = v_0$.

For any $L \in P$ and $L > 0$, L can be represented as $L = \sum_{j=0}^{r-1} 2^{l_j}$, where $1 \leq r \leq \lambda$ and $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$. Since $C^L \neq 0$, the number of $*$'s in the cube-variable row vector C^L is the number of columns in D, whose entries on the row $l_0, l_1, \ldots, l_{r-1}$ are all $*$'s. Note that for any $0 \leq \Gamma \leq 2^\lambda - 1$, the column pattern ψ_Γ has all entries on the row $l_0, l_1, \ldots, l_{r-1}$ being $*$'s if and only if $\Gamma \succeq L$. Since the root column vector of $\delta_{\Gamma,i}$ is ψ_Γ, thus for any $\Gamma \in M$ and any $0 \leq i \leq |Y_\Gamma| - 1$, the column pattern $\delta_{\Gamma,i}$ has all entries on the row $l_0, l_1, \ldots, l_{r-1}$ being $*$'s if and only if $\Gamma \succeq L$. Therefore, the number of columns in D, whose entries on the row $l_0, l_1, \ldots, l_{r-1}$ are all $*$'s, is

$$\begin{aligned}
&\sum_{\substack{\Gamma \in \overline{M}: \\ \Gamma \succeq L}} z_\Gamma + \sum_{\substack{\Gamma \in M: \\ \Gamma \succeq L}} \left(z_\Gamma - \sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \right) + \sum_{\substack{\Gamma \in M: \\ \Gamma \succeq L}} \sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \\
&= \sum_{0 \leq \Gamma \leq 2^\lambda - 1 : \Gamma \succeq L} z_\Gamma = k_L.
\end{aligned}$$

Therefore, the number of $*$'s in the row vector C^L is k_L. Since $C^L \neq 0$, by Lemma 5, $V(C^L) = 2^{k_L}$. Thus, for any $L \in P$ and $L > 0$, $V(C^L) = 2^{k_L} = v_L$.

In summary, for any $0 \leq \Gamma \leq 2^\lambda - 1$, $V(C^\Gamma) = v_\Gamma$. Thus, the matrix D satisfies the given intersection pattern. $\Box$

Comment: The above proof provides a way of synthesizing a cube-variable matrix to satisfy the given intersection pattern when the three conditions are all satisfied.

Example 20

Given $v_0 = 64, v_1 = 4, v_2 = 8, v_3 = 0, v_4 = 16, v_5 = 2, v_6 = 2, v_7 = 0, v_8 = 8, v_9 = 1, v_{10} = 2, v_{11} = 0, v_{12} = 0, v_{13} = 0, v_{14} = 0, v_{15} = 0$, *determine whether there exists a set of four cubes* $c_0, \ldots, c_3$ *on* 6 *variables* $x_0, \ldots, x_5$ *that satisfies the intersection pattern* $(v_0, \ldots, v_{15})$.

Solution: *First, it is not hard to check that both Statement 1 and Statement 2 in Theorem 23 hold for the given pattern.*

Now we check whether Statement 3 in Theorem 23 holds. For the given intersection pattern, we have $P = \{0, 1, 2, 4, 5, 6, 8, 9, 10\}$, $Z = \{3, 7, 11, 12, 13, 14, 15\}$, *and*

$$\begin{aligned} &k_0 = 6, \quad k_1 = 2, \quad k_2 = 3, \quad k_4 = 4, \quad k_5 = 1, \\ &k_6 = 1, \quad k_8 = 3, \quad k_9 = 0, \quad k_{10} = 1. \end{aligned}$$

Notice that $Z_2 = \{3, 12\}$. *The corresponding representative compatible column pattern sets are* $\rho_3 = \{(01**)^T\}$ *and* $\rho_{12} = \{(**01)^T\}$, *respectively. Thus, we have*

$$Y = \bigcup_{\Gamma \in Z_2} \rho_\Gamma = \{(01**)^T, (**01)^T\}.$$

Since the root column vector of $(01**)^T$ *is* ψ_{12} *and the root column vector of* $(**01)^T$ *is* ψ_3, *we have* $M = \{3, 12\}$. *We can partition the set* Y *as* $Y_3 = \{(**01)^T\}$ *and* $Y_{12} = \{(01**)^T\}$.

Based on Definition 22, the element in the set Y_3 *is defined as* $\delta_{3,0} = (**01)^T$ *and the element in the set* Y_{12} *is defined as* $\delta_{12,0} = (01**)^T$. *Notice that* $\rho_3 = \{\delta_{12,0}\}$ *and* $\rho_{12} = \{\delta_{3,0}\}$.

We can derive the system of equations (5.11) *for this example as*

$$\begin{cases} \sum_{i=0}^{15} \tilde{z}_i = 6 \\ \tilde{z}_1 + \tilde{z}_3 + \tilde{z}_5 + \tilde{z}_7 + \tilde{z}_9 + \tilde{z}_{11} + \tilde{z}_{13} + \tilde{z}_{15} = 2 \\ \tilde{z}_2 + \tilde{z}_3 + \tilde{z}_6 + \tilde{z}_7 + \tilde{z}_{10} + \tilde{z}_{11} + \tilde{z}_{14} + \tilde{z}_{15} = 3 \\ \tilde{z}_4 + \tilde{z}_5 + \tilde{z}_6 + \tilde{z}_7 + \tilde{z}_{12} + \tilde{z}_{13} + \tilde{z}_{14} + \tilde{z}_{15} = 4 \\ \tilde{z}_5 + \tilde{z}_7 + \tilde{z}_{13} + \tilde{z}_{15} = 1 \\ \tilde{z}_6 + \tilde{z}_7 + \tilde{z}_{14} + \tilde{z}_{15} = 1 \\ \sum_{i=8}^{15} \tilde{z}_i = 3 \\ \tilde{z}_9 + \tilde{z}_{11} + \tilde{z}_{13} + \tilde{z}_{15} = 0 \\ \tilde{z}_{10} + \tilde{z}_{11} + \tilde{z}_{14} + \tilde{z}_{15} = 1 \\ \tilde{w}_{3,0} \leq \tilde{z}_3 \\ \tilde{w}_{12,0} \leq \tilde{z}_{12} \\ \tilde{w}_{3,0} \geq 1 \\ \tilde{w}_{12,0} \geq 1 \end{cases}$$

The above system of equations has a non-negative solution

$$\begin{aligned} &\tilde{z}_3 = 1, \tilde{z}_4 = 1, \tilde{z}_7 = 1, \tilde{z}_{10} = 1, \tilde{z}_{12} = 2, \\ &\tilde{z}_0 = \tilde{z}_1 = \tilde{z}_2 = \tilde{z}_5 = \tilde{z}_6 = \tilde{z}_8 = 0, \\ &\tilde{z}_9 = \tilde{z}_{11} = \tilde{z}_{13} = \tilde{z}_{14} = \tilde{z}_{15} = 0, \\ &\tilde{w}_{3,0} = 1, \tilde{w}_{12,0} = 1. \end{aligned}$$

Thus, Statement 3 in Theorem 23 also holds. Therefore, there exists a cube-variable matrix to satisfy the given intersection pattern. Based on the proof of Theorem 23, we can synthesize a cube-variable matrix that satisfies the given intersection pattern based *on the above non-negative solution as*

$$\begin{bmatrix} * & 1 & * & 1 & 1 & 0 \\ * & 1 & * & * & 1 & 1 \\ 0 & * & * & 1 & * & * \\ 1 & 1 & 1 & * & * & * \end{bmatrix}$$

and the corresponding cubes are

$$c_0 = x_1 \wedge x_3 \wedge x_4 \wedge \bar{x}_5$$

$$c_1 = x_1 \wedge x_4 \wedge x_5$$

$$c_2 = \bar{x}_0 \wedge x_3$$

$$c_3 = x_0 \wedge x_1 \wedge x_2$$

It is not hard to verify that cubes $c_0, \ldots, c_3$ satisfy the given intersection pattern. $\square$

5.5 Implementation

In this section, we will discuss the implementation of the procedure to solve the λ-cube intersection problem, based on the theory in Section 5.4.

5.5.1 Checking Statement 1 in Theorem 23

We can represent Statement 1 in Theorem 23 in an alternative way, as shown by the following theorem.

Theorem 24

The following two statements are equivalent:

1. *The intersection pattern $(v_0, \ldots, v_{2^\lambda-1})$ satisfies that for any $0 \leq L \leq 2^\lambda - 1$, if $v_L > 0$, then for any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq L$, $v_\Gamma > 0$.*

2. *The intersection pattern $(v_0, \ldots, v_{2^\lambda-1})$ satisfies that for any $1 \leq k \leq \lambda$ and any $L \in P_k$, if $0 \leq \Gamma \leq 2^\lambda - 1$ satisfies that $||\Gamma|| = k - 1$ and $\Gamma \preceq L$, then $v_\Gamma > 0$.* $\square$

Proof. Statement 1 $\Rightarrow$ Statement 2: Consider any $L \in P_k$, where $1 \leq k \leq \lambda$. By the definition of P_k, we have $v_L > 0$. Since Statement 1 holds, therefore, for any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $||\Gamma|| = k - 1$ and $\Gamma \preceq L$, we have $v_\Gamma > 0$. Thus, Statement 2 holds.

Statement 2 $\Rightarrow$ Statement 1: When $L = 0$, we have $v_0 > 0$. Notice that the only $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq 0$ is $\Gamma = 0$. Thus, for any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq 0$, we have $v_\Gamma > 0$.

Now consider any $1 \leq L \leq 2^\lambda - 1$ such that $v_L > 0$. Suppose that $||L|| = r$. Then, $1 \leq r \leq \lambda$ and $L \in P_r$. For any Γ such that $0 \leq \Gamma \leq 2^\lambda - 1$ and $\Gamma \preceq L$, suppose that $||\Gamma|| = t$. Then, we have $0 \leq t \leq r$. We can find $r - t + 1$ numbers $\Gamma_t, \ldots, \Gamma_r$, such that $\Gamma_t = \Gamma$, $\Gamma_r = L$, and for any $t \leq k \leq r - 1$, $||\Gamma_k|| = k$ and $\Gamma_k \preceq \Gamma_{k+1}$. Since Statement 2 holds and $v_{\Gamma_r} = v_L > 0$, we can see that for any $t \leq k \leq r - 1$, $v_{\Gamma_k} > 0$. In particular, $v_\Gamma = v_{\Gamma_t} > 0$. Thus, for any $0 \leq \Gamma \leq 2^\lambda - 1$ such that $\Gamma \preceq L$, we have $v_\Gamma > 0$. This concludes the proof. $\square$

Based on Theorem 24, in order to check whether Statement 1 in Theorem 23 holds, we only need to check whether Statement 2 in Theorem 24 holds. Thus, whether Statement 1 in Theorem 23 holds can be checked by the procedure shown in Algorithm 5.

5.5.2 Checking Statement 2 in Theorem 23

Whether Statement 2 in Theorem 23 holds can be checked by representing the given intersection pattern by an undirected graph and listing all maximal cliques of the undirected graph.

For a given intersection pattern on λ cubes, we can construct an undirected graph $G(N, E)$ from that pattern, where N is a set of λ nodes $n_0, \ldots, n_{\lambda-1}$ and E is a set of edges. There is an edge between the node n_i and n_j $(0 \leq i < j \leq \lambda - 1)$ if and only if the number $(2^i + 2^j)$ is in the set P_2.

Algorithm 5 CheckRuleOne(λ, v): the procedure to check whether Statement 1 in Theorem 23 holds. It returns 1 if the statement holds; otherwise, it returns 0.

```
{Given an integer $\lambda \geq 1$ and a non-negative integer array $v = (v_0, \ldots, v_{2^\lambda - 1})$.}
for $i \Leftarrow 0$ to $\lambda$ do
  $P_i \Leftarrow \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, ||\Gamma|| = i,$ and $v_\Gamma > 0\}$;
end for
for $i \Leftarrow 1$ to $\lambda$ do
  for all $L \in P_i$ do
    for all $0 \leq \Gamma \leq 2^\lambda - 1$ s.t. $||\Gamma|| = i - 1$ and $\Gamma \preceq L$ do
      if $v_\Gamma = 0$ then
        return 0;
      end if
    end for
  end for
end for
return 1;
```

For example, we can represent the intersection pattern shown in Example 17 by the undirected graph shown in Figure 5.3.

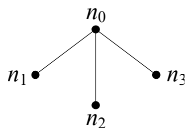


Figure 5.3: An undirected graph constructed from the intersection pattern of Example 17.

In graph theory, a *clique* in an undirected graph $G(N, E)$ is defined as a subset Q of the node set N, such that for every two nodes in Q, there exists an edge connecting the two. A *maximal clique* is a clique that cannot be extended by including one more adjacent node.

For an intersection pattern, if a set of r $(3 \leq r \leq \lambda)$ numbers $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$ satisfies that for any $0 \leq i < j \leq r - 1$, $v_{(2^{l_i} + 2^{l_j})} > 0$, then, the set of nodes $n_{l_0}, \ldots, n_{l_{r-1}}$ forms a clique of the undirected graph constructed from the intersection pattern. Thus, Statement 2 in Theorem 23 can be stated in another way as: For any clique $Q = \{n_{l_0}, \ldots, n_{l_{r-1}}\}$ of size r in the undirected graph constructed from the intersection pattern, where $3 \leq r \leq \lambda$ and $0 \leq l_0 < \cdots < l_{r-1} \leq \lambda - 1$, we have $v_L > 0$, where $L = \sum_{i=0}^{r-1} 2^{l_i}$.

The following theorem shows that if Statement 1 in Theorem 23 holds, then to check whether Statement 2 holds, we only need to focus on all maximal cliques of the undirected graph $G(N, E)$.

Theorem 25

If Statement 1 in Theorem 23 holds, then Statement 2 in Theorem 23 holds if and only if for any maximal *clique $Q^* = \{n_{d_0}, \ldots, n_{d_{t-1}}\}$ of size t in the undirected graph constructed from the intersection pattern, where $3 \leq t \leq \lambda$ and $0 \leq d_0 < \cdots < d_{t-1} \leq \lambda - 1$, we have $v_{L^*} > 0$, where $L^* = \sum_{i=0}^{t-1} 2^{d_i}$.* $\square$

Proof. The "only if" part of the above theorem is obvious. We now prove the "if" part. Consider any clique $Q = \{n_{l_0}, \ldots, n_{l_{r-1}}\}$ in the undirected graph $G(N, E)$. By the definition of maximal clique, Q is contained in a maximal clique $Q^* = \{n_{d_0}, \ldots, n_{d_{t-1}}\}$, where $r \leq t \leq \lambda$, $0 \leq d_0 < \cdots < d_{t-1} \leq \lambda - 1$. Since the clique Q is contained in the clique Q^*, we have $Q \subseteq Q^*$. Let $L = \sum_{i=0}^{r-1} 2^{l_i}$ and $L^* = \sum_{i=0}^{t-1} 2^{d_i}$. Since Q^* is a maximal clique, by the assumption, we have $v_{L^*} > 0$. Since $Q \subseteq Q^*$, we have $L \preceq L^*$. Since Statement 1 in Theorem 23 holds, we obtain $v_L > 0$. Thus, for any clique $Q = \{n_{l_0}, \ldots, n_{l_{r-1}}\}$ in the undirected graph $G(N, E)$, we have $v_L > 0$. Therefore, Statement 2 in Theorem 23 holds. $\square$

Therefore, if Statement 1 in Theorem 23 holds, then whether Statement 2 in Theorem 23 holds can be answered by checking whether all v_L's corresponding to all maximal cliques in the undirected graph $G(N, E)$ are greater than zero. The problem of listing all maximal cliques in an undirected graph is a classical problem in graph theory and can be solved, for example, by the Born-Kerbosch algorithm [40].

Assuming that Statement 1 in Theorem 23 holds, then whether Statement 2 in Theorem 23 holds can be checked by the procedure shown in Algorithm 6.

Algorithm 6 CheckRuleTwo(λ, v): the procedure to check whether Statement 2 in Theorem 23 holds under the assumption that Statement 1 in Theorem 23 holds. It returns 1 if the statement holds; otherwise, it returns 0.

```
{Given an integer $\lambda \geq 1$ and a non-negative integer array $v = (v_0, \ldots, v_{2^\lambda - 1})$.}
$N \Leftarrow \{n_0, \ldots, n_{\lambda-1}\}$; $E \Leftarrow \phi$;
for $i \Leftarrow 0$ to $\lambda - 1$ do
  for $j \Leftarrow i + 1$ to $\lambda - 1$ do
    if $v_{(2^i+2^j)} > 0$ then
      $E \Leftarrow E \cup \{e(n_i, n_j)\}$; {Add an edge between the node $n_i$ and the node $n_j$ into the edge set $E$.}
    end if
  end for
end for
for all maximal clique $Q$ in the graph $G(N, E)$ do
  $L \Leftarrow \sum_{i:n_i \in Q} 2^i$;
  if $v_L = 0$ then
    return 0;
  end if
end for
return 1;
```

5.5.3 Checking Statement 3 in Theorem 23

The following theorem shows that to check whether the system of equations (5.11) has a non-negative solution, we only need to check whether an alternative system of equations with fewer unknowns has a non-negative solution.

Theorem 26

The system of equations (5.11) *has a non-negative integer solution if and only if the system of equations on unknowns $\hat{z}_\Gamma$ (for all $\Gamma \in \overline{M}$) and $\hat{w}_{\Gamma,i}$ (for all $\Gamma \in M$ and $0 \leq i \leq |Y_\Gamma| - 1$)*

$$\begin{aligned} \sum_{\Gamma \in \overline{M}, \Gamma \succeq L} \hat{z}_\Gamma + \sum_{\Gamma \in M, \Gamma \succeq L} \sum_{i=0}^{|Y_\Gamma|-1} \hat{w}_{\Gamma,i} &= k_L, \text{ for all } L \in P \\ \sum_{\substack{\Gamma \in M, 0 \leq i \leq |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} \hat{w}_{\Gamma,i} &\geq 1, \text{ for all } L \in Z_2 \end{aligned} \tag{5.12}$$

has a non-negative integer solution. □

PROOF. **"if"** part: Suppose that a non-negative integer solution to the system of equations (5.12) is

$$\begin{cases} \hat{z}_\Gamma = z_\Gamma, & \text{for all } \Gamma \in \overline{M}, \\ \hat{w}_{\Gamma,i} = w_{\Gamma,i}, & \text{for all } \Gamma \in M, 0 \le i \le |Y_\Gamma| - 1. \end{cases}$$

We let

$$\begin{cases} \tilde{z}_\Gamma = z_\Gamma, & \text{for all } \Gamma \in \overline{M}, \\ \tilde{z}_\Gamma = \sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i}, & \text{for all } \Gamma \in M, \\ \tilde{w}_{\Gamma,i} = w_{\Gamma,i}, & \text{for all } \Gamma \in M, 0 \le i \le |Y_\Gamma| - 1. \end{cases}$$

Then, it is not hard to see that $\tilde{z}_\Gamma$ (for all $0 \le \Gamma \le 2^\lambda - 1$) and $\tilde{w}_{\Gamma,i}$ (for all $\Gamma \in M$ and $0 \le i \le |Y_\Gamma| - 1$) form a non-negative integer solution to the system of equations (5.11).

"only if" part: Suppose that a non-negative integer solution to the system of equations (5.11) is

$$\begin{cases} \tilde{z}_\Gamma = z_\Gamma, & \text{for all } 0 \le \Gamma \le 2^\lambda - 1, \\ \tilde{w}_{\Gamma,i} = w_{\Gamma,i}, & \text{for all } \Gamma \in M, 0 \le i \le |Y_\Gamma| - 1. \end{cases} \tag{5.13}$$

We let

$$\begin{cases} \hat{z}_\Gamma = z_\Gamma, & \text{for all } \Gamma \in \overline{M}, \\ \hat{w}_{\Gamma,i} = z_\Gamma - \sum_{i=1}^{|Y_\Gamma|-1} w_{\Gamma,i}, & \text{for all } \Gamma \in M, i = 0, \\ \hat{w}_{\Gamma,i} = w_{\Gamma,i}, & \text{for all } \Gamma \in M, 1 \le i \le |Y_\Gamma| - 1. \end{cases} \tag{5.14}$$

Then, for all $\Gamma \in \overline{M}$, $\hat{z}_\Gamma \ge 0$ and for all $\Gamma \in M, 1 \le i \le |Y_\Gamma| - 1$, $\hat{w}_{\Gamma,i} \ge 0$. Since for all $\Gamma \in M$, $\sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \le z_\Gamma$, then we have that for all $\Gamma \in M$,

$$0 \le z_\Gamma - \sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \le z_\Gamma - \sum_{i=1}^{|Y_\Gamma|-1} w_{\Gamma,i} = \hat{w}_{\Gamma,0}.$$

Based on Equation (5.11), (5.13), and (5.14), we have that for all $L \in P$,

$$\begin{aligned}&\sum_{\Gamma \in \overline{M}, \Gamma \succeq L} \hat{z}_\Gamma + \sum_{\Gamma \in M, \Gamma \succeq L} \sum_{i=0}^{|Y_\Gamma|-1} \hat{w}_{\Gamma,i} \\ = &\sum_{\Gamma \in \overline{M}, \Gamma \succeq L} z_\Gamma + \sum_{\Gamma \in M, \Gamma \succeq L} z_\Gamma = \sum_{0 \le \Gamma \le 2^\lambda - 1, \Gamma \succeq L} \tilde{z}_\Gamma = k_L.\end{aligned}$$

Since for all $\Gamma \in M$, $\sum_{i=0}^{|Y_\Gamma|-1} w_{\Gamma,i} \le z_\Gamma$, then we have that for all $\Gamma \in M$,

$$\hat{w}_{\Gamma,0} = z_\Gamma - \sum_{i=1}^{|Y_\Gamma|-1} w_{\Gamma,i} \ge w_{\Gamma,0}. \tag{5.15}$$

Combining Equation (5.15) with Equation (5.11), (5.13), and (5.14), we have that for all $\Gamma \in M$

$$\begin{aligned}1 \le &\sum_{\substack{\Gamma \in M, 0 \le i \le |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} \tilde{w}_{\Gamma,i} = \sum_{\substack{\Gamma \in M, 0 \le i \le |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} w_{\Gamma,i} \\ \le &\sum_{\substack{\Gamma \in M, 0 \le i \le |Y_\Gamma|-1: \\ \delta_{\Gamma,i} \in \rho_L}} \hat{w}_{\Gamma,i}.\end{aligned}$$

Then, $\hat{z}_\Gamma$ (for all $\Gamma \in \overline{M}$) and $\hat{w}_{\Gamma,i}$ (for all $\Gamma \in M, 1 \le i \le |Y_\Gamma| - 1$) form a non-negative integer solution to the system of equations (5.12). □

Based on Theorem 26, to check whether Statement 3 in Theorem 23 holds, we only need to check whether the system of equations (5.12) has a non-negative solution. Note that the system of equations (5.12) has $|M|$ fewer unknowns and $|M|$ fewer inequalities than the original system of equations (5.11). Thus, a certain amount of computation will be saved.

5.5.4 The Procedure to Solve the λ-Cube Intersection Problem

Based on the above discussion, we give the procedure to solve the λ-cube intersection problem in Algorithm 7. In the procedure, the function CheckRuleOne(λ, v) and the function CheckRuleTwo(λ, v) are shown in Algorithm 5 and 6, respectively. The function RCCPS(Γ, λ, P_2) returns the representative compatible column pattern set for a $\Gamma \in Z_2$. The function

$$\text{SetEqn}(P, Z_2, M, \overline{M}, \{k_L | L \in P\}, \{\rho_L | L \in Z_2\}, \{Y_L | L \in M\})$$

returns the matrices A_{ze}, A_{we}, A_w and the column vectors b_e and b in the matrix representation of the system of equations (5.12), which is

$$\begin{cases} A_{ze}\vec{z} + A_{we}\vec{w} = b_e, \\ A_w\vec{w} \geq b, \end{cases} \tag{5.16}$$

where $\vec{z}$ is a column vector of unknowns $\hat{z}_\Gamma$, for all $\Gamma \in \overline{M}$, and $\vec{w}$ is a column vector of unknowns $\hat{w}_{\Gamma,i}$, for all $\Gamma \in M$ and $0 \leq i \leq |Y_\Gamma|-1$. The function NonNegSln($A_{ze}, A_{we}, b_e, A_w, b$) finds a non-negative integer solution to the system of equations (5.16). If the system of equations (5.16) has a non-negative integer solution, then the function returns one such solution; otherwise, it returns ϕ. Given a non-negative solution $(\vec{z}, \vec{w})$ to the system of equations (5.16), the function SynCubes($\vec{z}, \vec{w}, \lambda$) synthesizes a set of λ cubes from that solution.

5.6 Experimental Results

We tested our algorithm on two-level logic benchmarks that accompany the two-level logic minimizer Espresso [41]. For each benchmark, we ignored the output part of the cubes and just set the number of outputs to be one. We optimized each modified benchmark by Espresso and then call a program to generate an intersection pattern file of that benchmark. This intersection pattern file serves as the input to our program.

We performed two sets of experiments to test our algorithm. In the first set of experiments, we tested our algorithm on solving special cases. The main goal was to study the runtime of our algorithm. The benchmarks we tested are listed in Table 5.2.

Algorithm 7 CubePattern(λ, v): the procedure to check whether there exists a set of λ cubes to satisfy the given intersection pattern $v = (v_0, \ldots, v_{2^\lambda - 1})$. If the answer is yes, the procedure returns a set of cubes that satisfies the intersection pattern; otherwise, it returns ϕ.

```
{Given an integer $\lambda \geq 1$ and a non-negative integer array $v = (v_0, \ldots, v_{2^\lambda - 1})$, where
  each entry is from the set $\{0, 2^0, 2^1, \ldots 2^n\}$.}
$P \Leftarrow \phi$; $Z \Leftarrow \phi$;
**for** $i \Leftarrow 0$ to $2^\lambda - 1$ **do**
  **if** $v_\Gamma > 0$ **then**
    $P \Leftarrow P \cup \{\Gamma\}$;
    $k_\Gamma \Leftarrow \log_2 v_\Gamma$;
  **else** $\{v_\Gamma = 0\}$
    $Z \Leftarrow Z \cup \{\Gamma\}$;
  **end if**
**end for**
**if** CheckRuleOne($\lambda, v$) $= 0$ **then**
  **return** $\phi$;
**end if**
**if** CheckRuleTwo($\lambda, v$) $= 0$ **then**
  **return** $\phi$;
**end if**
$P_2 \Leftarrow \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, ||\Gamma|| = 2, \text{ and } v_\Gamma > 0\}$;
$Z_2 \Leftarrow \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, ||\Gamma|| = 2, \text{ and } v_\Gamma = 0\}$;
**for all** $\Gamma \in Z_2$ **do**
  $\rho_\Gamma = \text{RCCPS}(\Gamma, \lambda, P_2)$;
**end for**
$Y \Leftarrow \bigcup_{\Gamma \in Z_2} \rho_\Gamma$;
$M \Leftarrow \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, \text{ s.t. } \exists W \in Y \text{ s.t. } t(W) = \psi_\Gamma\}$;
$\overline{M} \Leftarrow \{\Gamma | 0 \leq \Gamma \leq 2^\lambda - 1, \Gamma \notin M\}$;
**for all** $\Gamma \in M$ **do**
  $Y_\Gamma \Leftarrow \{W | W \in Y \text{ and } t(W) = \psi_\Gamma\}$;
**end for**
$(A_{ze}, A_{we}, b_e, A_w, b) \Leftarrow \text{SetEqn}(P, Z_2, M, \overline{M}, \{k_L | L \in P\}, \{\rho_L | L \in Z_2\}, \{Y_L | L \in
  M\})$;
$(\vec{z}, \vec{w}) \Leftarrow \text{NonNegSln}(A_{ze}, A_{we}, b_e, A_w, b)$;
**if** $(\vec{z}, \vec{w}) = \phi$ **then**
  **return** $\phi$;
**end if**
**return** SynCubes($\vec{z}, \vec{w}, \lambda$);
```

Since just a few benchmarks generated a special intersection pattern, we manually created some test cases. For example, the benchmark `mark1_11` was created from the original benchmark `mark1` by deleting five cubes. Notice that by deleting some cubes, the new benchmark still has its intersection of all cubes nonempty. Not surprisingly, the runtime increased exponentially with the number of cubes λ. This is because the number of unknowns increases exponentially with λ. However, since the size of the inputs to our program is $O(2^\lambda)$, which is proportional to the number of intersections, the runtime complexity compared to the size of the inputs is linear. Further, for the benchmark `shift`, although the number of unknowns is more than 2 million, our algorithm is able to obtain the solution in about 70 seconds.

Table 5.2: Number of unknowns and runtime for special case problems.

circuit	#cubes	#inputs	#unknowns	time (s)
`newtpla2`	9	10	512	0
`in3`	10	35	1024	0
`mark1_11`	11	20	2048	0.01
`mark1_12`	12	20	4096	0.04
`mark1_13`	13	20	8192	0.08
`mark1_14`	14	20	16384	0.2
`mark1_15`	15	20	32768	0.48
`mark1`	16	20	65536	1.18
`shift_17`	17	19	131072	1.73
`shift_18`	18	19	262144	3.19
`shift_19`	19	19	524288	7.84
`shift_20`	20	19	1048576	24.97
`shift`	21	19	2097152	71.33

In the second set of experiments, we tested our algorithm to solve general cases. We developed a program that takes an intersection pattern file and writes out the system of equations (5.12). This system of equations can be fed into specialized programs to solve for non-negative solution. We list the numbers of unknowns and the numbers of equations on some benchmarks in Table 5.3. We compared the number of unknowns obtained by our method to the number of unknowns of a naive method in which all 3^λ combinations of column patterns are taken as unknowns to set up equations. The number of unknowns generated by our method and the number of unknowns generated by the naive method are listed in the fourth column and the fifth column of Table 5.3, respectively. The ratio of the number of unknowns generated by our method to that generated by the naive method is listed in the sixth column. We can see that our algorithm greatly reduced the number of unknowns: for most of the benchmarks, our method can reduce more than 95% of unknowns. Thus, we believe that our proposed algorithm will greatly reduce the runtime to solve the general case problem compared to the naive method.

Table 5.3: Number of unknowns and number of equations for general case problems.

circuit	#cubes	#inputs	#unknowns			#equations
			our (a)	naive (b)	ratio (a/b)	
`luc`	6	8	66	729	0.091	32
`br2`	6	12	228	729	0.31	22
`tms`	8	8	262	6561	0.040	69
`prom2`	9	9	512	19683	0.026	265
`br1`	10	12	8108	59049	0.137	58
`vg2`	10	25	1294	59049	0.022	71
`exps`	12	8	4130	531441	0.008	399
`alu1`	12	12	4096	531441	0.008	1300
`exp`	14	8	69470	4782969	0.015	122
`newtpla`	14	15	127908	4782969	0.027	117

Chapter 6

Conclusion and Future Directions

The computation that we are advocating in this dissertation has a pseudo *analog* character, reminiscent of computations performed by physical systems such as electronics on continuously variable signals such as voltage. In our case, the variable signal is the probability of obtaining a one in a stochastic yet *digital* bit stream. Indeed, our system is built from ordinary, cheap digital electronics such as CMOS. Digital constructs in CMOS operate on physical signals such as voltage, of course. However, they are designed with the premise that these signals can always be unequivocally interpreted as zero or as one.

This is certainly counterintuitive: why impose an analog view on digital values? As we have demonstrated in this dissertation, it might often be advantageous to do so, both from the standpoint of the hardware resources required as well as the error tolerance of the computation. Many of the functions that we seek to implement for computational systems such as signal processing are *arithmetic* functions, consisting of operations like addition and multiplication. Complex functions, such as exponentials and trigonometric functions, are generally computed through polynomial approximations, so through multiplications and additions. As we have argued, these operations are very natural and efficient when performed on stochastic bit streams.

We are the first to tackle the problem of synthesizing arbitrary arithmetic functions through logical computation on stochastic bit streams. The synthesis results for our stochastic implementations of a variety of functions are convincing. The hardware cost is much less than that of conventional implementations with adders and multipliers. Since stochastic bit streams are uniform, with no bit privileged above any other, the computation is highly error tolerant. As higher and higher rates of bit flips occur, the accuracy degrades gracefully.

Indeed, computation on stochastic bit streams could offer tunable precision: as the length of the stochastic bit stream increases, the precision of the value represented by it also increases. Thus, without hardware redesign, we have the flexibility to tradeoff precision and computation time. In contrast, with a conventional binary-radix implementation, when a higher precision is required, the underlying hardware has to be redesigned.

Finally, we would like to point out several future research directions related to logical computation on stochastic bit streams.

One exciting future direction is to study the relationship between *coding* and *computation*. Traditional research in coding theory focuses on the relationship between information coding and the *transmission* of information. Such research falls into one of two categories: data compression coding, which studies how to compress the data for efficient transmission, and error correction coding, which studies how to encode the data so that it can be transmitted more reliably. Our research in logical computation on stochastic bit streams shows that the stochastic encoding has advantages over the conventional binary radix encoding in terms of hardware cost and fault tolerance, but has disadvantages in terms of data length and computation time. This suggests that there is a strong connection between data coding and computation with the data. Given that the binary radix encoding and the stochastic encoding are at the extreme ends of the spectrum of data encoding, we believe that an encoding that lies in the middle of the spectrum may take both the advantages of the binary radix encoding and the stochastic encoding. A promising approach is to represent a number as a concatenation of short stochastic bit streams, each stochastically encoding a value and associated with a different weight.

We have demonstrated that logical computation on stochastic bit streams is tolerant to bit flip errors on the stream. Indeed, the effect of bit flip error on the stochastic bit stream is *predictable*. To illustrate this, consider a stochastic bit stream X that encodes a value p. Suppose that bit flip error occurs at a rate of ϵ and it corrupts the original stochastic bit stream X into another stochastic bit stream X'. As shown by Equation (1.1), the value p' represented by the stream X' is $p' = P(X' = 1) = \epsilon + (1 - 2\epsilon)p$. Thus, if we know the bit flip rate ϵ *a priori*, we could extract the correct value p from the corrupted bit stream. This is a significant advantage of the stochastic representation over the binary radix representation. For binary radix, even if we know the bit flip rate *a priori*, we still cannot infer the correct value: we need to know exactly which bits of the binary number are flipped to reconstruct the correct value. An interesting future direction is to design an error correcting unit in the probabilistic domain to fully exploit such an advantage of the stochastic representation. The error correcting unit will further enhance the fault tolerance of logical computation on stochastic bit streams.

Logical computation on stochastic bit streams is also a promising paradigm for designing circuits with emerging nanoscale devices. As the semiconductor industry contemplates the end of Moore's Law, there has been a groundswell of interest in technologies that offer a path to scaling beyond the limits of the current CMOS technology. Nanoscale technologies such as carbon nanotubes, nanowire arrays, and molecular switches present both challenges and opportunities for digital circuit design. On the one hand, such technologies promise unprecedented densities, which will translate into vast numbers of switches, logic gates, and bits. On the other hand, both the scale and the assembly techniques constrain the circuits, particularly if they are self-assembled. Most of the technologies are characterized by very high defect rates, as well as inherent randomness in their wiring [42].

Most research in implementing circuits with nanoscale technologies focuses on how to adapt the technology to the well-established paradigm used by CMOS circuit. At the device and the circuit level, much effort is expended on eliminating the randomness and defects of the basic constructs so that they behave much like reliable CMOS transistors [43]. At the logical and the architectural level, fault-tolerant techniques are applied to ensure the correct behavior of the circuit [44].

However, in our view, since such emerging nanotechnologies differ from today's CMOS technology in many fundamental aspects, maintaining the computational paradigm used by CMOS technology for nanoscale computation is not wise. Instead, we should explore new paradigms that can exploit the inherent randomness of the nanoscale devices. Logical computation on stochastic bit streams is a promising choice.

References

[1] W. Qian, J. Backes, and M. D. Riedel. The synthesis of stochastic circuits for nanoscale computation. *International Journal of Nanotechnology and Molecular Computation*, 1(4):39–57, 2010.

[2] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Transactions on Computers*, 24(6):668–670, 1975.

[3] J. Savir, G. Ditlow, and P. H. Bardell. Random pattern testability. *IEEE Transactions on Computers*, 33(1):79–90, 1984.

[4] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. In *Design Automation Conference*, pages 661–666, 2001.

[5] R. Marculescu, D. Marculescu, and M. Pedram. Logic level power estimation considering spatiotemporal correlations. In *International Conference on Computer-Aided Design*, pages 294–299, 1994.

[6] S. Cheemalavagu, P. Korkmaz, K. Palem, B. Akgul, and L. Chakrapani. A probabilistic CMOS switch and its realization by exploiting noise. In *IFIP International Conference on VLSI*, pages 535–541, 2005.

[7] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem. Probabilistic system-on-a-chip architecture. *ACM Transactions on Design Automation of Electronic Systems*, 12(3):1–28, 2007.

[8] G. Lorentz. *Bernstein Polynomials.* University of Toronto Press, 1953.

[9] J. Berchtold and A. Bowyer. Robust arithmetic for multivariate Bernstein-form polynomials. *Computer-Aided Design*, 32(11):681–689, 2000.

[10] R. Farouki and V. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[11] W. Qian and M. D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *Design Automation Conference*, pages 648–653, 2008.

[12] D. Lee, R. Cheung, and J. Villasenor. A flexible architecture for precise gamma correction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(4):474–478, 2007.

[13] B. Dufort and G. W. Roberts. *Analog Test Signal Generation Using Periodic* ΣΔ-*Encoded Data Streams.* Kluwer Academic, 2000.

[14] Irotek. EasyRGB, 2008, http://www.easyrgb.com/index.php?X=MATH.

[15] D. Phillips. *Image Processing in C.* R & D Publications, 1994.

[16] T. Urabe. 3D examples, 2002, http://mathmuse.sci.ibaraki.ac.jp/geom/param1E.html.

[17] S. T. Ribeiro. Random-pulse machines. *IEEE Transactions on Electronic Computers*, 16(3):261–276, 1967.

[18] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, volume 2, chapter 2, pages 37–172. Plenum Press, 1969.

[19] C. Janer, J. Quero, J. Ortega, and L. Franquelo. Fully parallel stochastic computation architecture. *IEEE Transactions on Signal Processing*, 44(8):2110–2117, 1996.

[20] S. Toral, J. Quero, and L. Franquelo. Stochastic pulse coded arithmetic. In *International Symposium on Circuits and Systems*, volume 1, pages 599–602, 2000.

[21] B. Brown and H. Card. Stochastic neural computation I: Computational elements. *IEEE Transactions on Computers*, 50(9):891–905, 2001.

[22] C. Bishop. *Neural Networks for Patten Recognition.* Clarendon Press, 1995.

[23] S. Deiss, R. Douglas, and A. Whatley. A pulse coded communications infrastructure for neuromorphic systems. In *Pulsed Neural Networks*, chapter 6, pages 157–178. MIT Press, 1999.

[24] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[25] S. Narayanan, J. Sartori, R. Kumar, and D. Jones. Scalable stochastic processors. In *Design, Automation and Test in Europe*, pages 335–338, 2010.

[26] P. S. Duggirala, S. Mitra, R. Kumar, and D. Glazeski. On the theory of stochastic processors. In *International Conference on Quantitative Evaluation of Systems*, pages 292–301, 2010.

[27] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.

[28] D. Wilhelm and J. Bruck. Stochastic switching circuit synthesis. In *International Symposium on Information Theory*, pages 1388–1392, 2008.

[29] A. Mishchenko. ABC: A system for sequential synthesis and verification, 2007, http://www.eecs.berkeley.edu/ alanmi/abc/.

[30] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor. Generating binary sequences for stochastic computing. *IEEE Transactions on Information Theory*, 40(3):716–720, 1994.

[31] A. Gill. Synthesis of probability transformers. *Journal of the Franklin Institute*, 274(1):1–19, 1962.

[32] A. Gill. On a weight distribution problem, with application to the design of stochastic generators. *Journal of the ACM*, 10(1):110–121, 1963.

[33] C. E. Shannon. The synthesis of two terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.

[34] H. Zhou and J. Bruck. On the expressibility of stochastic switching circuits. In *International Symposium on Information Theory*, pages 2061–2065, 2009.

[35] R. K. Brayton, C. McMullen, G. D. Hachtel, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[36] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):1062–1081, 1987.

[37] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design*, 4(3):269–285, 1985.

[38] W. Qian and M. D. Riedel. Two-level logic synthesis for probabilistic computation. In *International Workshop on Logic and Synthesis*, pages 95–102, 2010.

[39] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, 78(2):264–300, 1990.

[40] C. Born and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[41] Espresso, 1994, http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm.

[42] A. DeHon. Nanowire-based programmable architectures. *ACM Journal on Emerging Technologies in Computing Systems*, 1(2):109–162, 2005.

[43] N. Patil, J. Deng, A. Lin, H.-S. P. Wang, and S. Mitra. Design methods for misaligned and mispositioned carbon-nanotube immune circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1725–1736, 2008.

[44] J. Han and P. Jonker. A system architecture solution for unreliable nanoelectronic devices. *IEEE Transactions on Nanotechnology*, 1(4):201–208, 2002.

Appendix A

A Proof of Theorem 1

For convenience, given a Bernstein polynomial $g(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t)$, we can also express it as

$$g(t) = \sum_{k=0}^{n} c_{k,n} t^k (1-t)^{n-k}, \tag{A.1}$$

where

$$c_{k,n} = \binom{n}{k} \beta_{k,n}, \tag{A.2}$$

for $k = 0, 1, \ldots, n$. Substituting Equation (A.2) into Equation (2.12), we have

$$c_{k,m+1} = \begin{cases} c_{0,m}, & \text{for } k = 0 \\ c_{k-1,m} + c_{k,m}, & \text{for } 1 \leq k \leq m \\ c_{m,m}, & \text{for } k = m+1. \end{cases} \tag{A.3}$$

Suppose that the polynomial g is of degree n. Applying Equation (A.3) recursively, we can express $c_{k,m}$ as a linear combination of $c_{0,n}, c_{1,n}, \ldots, c_{n,n}$.

Lemma 11

Let g be a polynomial of degree n. For any $m \geq n$, suppose that the Bernstein polynomial of degree m of g is $g(t) = \sum_{k=0}^{m} c_{k,m} t^k (1-t)^{m-k}$. Let $c_{k,m} = 0$ for all $k < 0$ and all

$k > m$. Then for all $k = 0, 1, \ldots, m$, we have

$$c_{k,m} = \sum_{i=0}^{m-n} \binom{m-n}{i} c_{k-m+n+i,n}. \qquad \square \tag{A.4}$$

Proof. We prove the lemma by induction on $m - n$.

Base case: For $m - n = 0$, the right-hand side of Equation (A.4) reduces to $\binom{0}{0} c_{k,n} = c_{k,m}$, so the equation holds.

Inductive step: Suppose that Equation (A.4) holds for some $m \geq n$ and all $k = 0, 1, \ldots, m$. Consider $m+1$. Since we assume that $c_{-1,m} = c_{m+1,m} = 0$, Equation (A.3) can be written as

$$c_{k,m+1} = c_{k-1,m} + c_{k,m}, \tag{A.5}$$

for all $k = 0, \ldots, m+1$. With our convention that $c_{i,n} = 0$ for all $i < 0$ and $i > n$, it is easily seen that

$$c_{-1,m} = 0 = \sum_{i=0}^{m-n} \binom{m-n}{i} c_{-1-m+n+i,n},$$
$$c_{m+1,m} = 0 = \sum_{i=0}^{m-n} \binom{m-n}{i} c_{m+1-m+n+i,n}.$$

Together with the induction hypothesis, we conclude that for all $k = -1, 0, \ldots, m, m+1$

$$c_{k,m} = \sum_{i=0}^{m-n} \binom{m-n}{i} c_{k-m+n+i,n}. \tag{A.6}$$

Based on Equations (A.5) and (A.6), for all $k = 0, 1, \ldots, m+1$, we have

$$c_{k,m+1} = \sum_{i=0}^{m-n} \binom{m-n}{i} c_{k-1-m+n+i,n} + \sum_{j=0}^{m-n} \binom{m-n}{j} c_{k-m+n+j,n}.$$

In the first sum, we change the summation index to $j = i - 1$. We obtain

$$\begin{aligned} c_{k,m+1} &= \sum_{j=-1}^{m-n-1} \binom{m-n}{j+1} c_{k-m+n+j,n} + \sum_{j=0}^{m-n} \binom{m-n}{j} c_{k-m+n+j,n} \\ &= \binom{m-n}{0} c_{k-m+n-1,n} + \sum_{j=0}^{m-n-1} \left[\binom{m-n}{j+1} + \binom{m-n}{j} \right] c_{k-m+n+j,n} + \binom{m-n}{m-n} c_{k,n}. \end{aligned}$$

Applying the basic formula $\binom{r}{q} = \binom{r-1}{q-1} + \binom{r-1}{q}$, we obtain

$$\begin{aligned} c_{k,m+1} &= c_{k-m+n-1,n} + \sum_{j=0}^{m-n-1} \binom{m+1-n}{j+1} c_{k-m+n+j,n} + c_{k,n} \\ &= \sum_{i=0}^{m+1-n} \binom{m+1-n}{i} c_{k-m-1+n+i,n}. \end{aligned}$$

Thus Equation (A.4) holds for $m + 1$. By induction, it holds for all $m \geq k$. □

Remark: Equation (A.4) can be formulated as

$$c_{k,m} = \sum_{i=\max\{0,k-m+n\}}^{\min\{k,n\}} \binom{m-n}{k-i} c_{i,n}, \tag{A.7}$$

for all $m \geq n$ and $k = 0, 1, \ldots, m$. Indeed, in Equation (A.4), first use the basic formula $\binom{r}{q} = \binom{r}{r-q}$ and then change the summation index to $j = k - m + n + i$ to obtain

$$c_{k,m} = \sum_{i=0}^{m-n} \binom{m-n}{m-n-i} c_{k-m+n+i,n} = \sum_{j=k-m+n}^{k} \binom{m-n}{k-j} c_{j,n}.$$

Note that $c_{j,n} \neq 0$ implies $0 \leq j \leq n$. This yields Equation (A.7). □

Lemma 12

Let n be a positive integer. For all integer m, k and i such that

$$m > n, \quad 0 \leq k \leq m, \quad \max\{0, k-m+n\} \leq i \leq \min\{k, n\}, \tag{A.8}$$

we have

$$\left| \left(\frac{k}{m}\right)^i \left(1 - \frac{k}{m}\right)^{n-i} - \frac{\binom{m-n}{k-i}}{\binom{m}{k}} \right| \leq \frac{n^2}{m}. \qquad \square \tag{A.9}$$

Proof. For simplicity, we define $\delta = \left(\frac{k}{m}\right)^i \left(1 - \frac{k}{m}\right)^{n-i} - \frac{\binom{m-n}{k-i}}{\binom{m}{k}}$. Now

$$\begin{aligned}\frac{\binom{m-n}{k-i}}{\binom{m}{k}} &= \frac{(m-n)!}{(k-i)!(m-n-k+i)!} \cdot \frac{k!(m-k)!}{m!} \\ &= \frac{k(k-1)\cdots(k-i+1)(m-k)(m-k-1)\cdots(m-n-k+i+1)}{m(m-1)\cdots(m-n+1)} \qquad \text{(A.10)} \\ &= \prod_{j=0}^{i-1} \frac{k-j}{m-j} \cdot \prod_{j=0}^{n-i-1} \frac{m-k-j}{m-i-j} = \prod_{j=0}^{i-1} \left(1 - \frac{m-k}{m-j}\right) \cdot \prod_{j=0}^{n-i-1} \left(1 - \frac{k-i}{m-i-j}\right).\end{aligned}$$

We obtain an upper bound for $\frac{\binom{m-n}{k-i}}{\binom{m}{k}}$ by replacing j in Equation (A.10) with its least value, 0:

$$\frac{\binom{m-n}{k-i}}{\binom{m}{k}} \leq \prod_{j=0}^{i-1} \left(1 - \frac{m-k}{m}\right) \cdot \prod_{j=0}^{n-i-1} \left(1 - \frac{k-i}{m-i}\right) = \left(\frac{k}{m}\right)^i \left(\frac{m-k}{m-i}\right)^{n-i}.$$

We need the following simple inequality: for real numbers $0 \leq t \leq y \leq 1$ and a non-negative integer l,

$$y^l - t^l = (y-t) \sum_{j=0}^{l-1} y^j t^{l-1-j} \leq (y-t)l. \qquad \text{(A.11)}$$

From Equation (A.8), we obtain $0 \leq i \leq \min\{k, n\} \leq k \leq m$ and so we can use Equation (A.11) for

$$0 \leq t = \frac{m-k}{m} \leq \frac{m-k}{m-i} = y \leq 1, \quad l = n - i \geq 0.$$

We obtain

$$\begin{aligned}\delta &= \left(\frac{k}{m}\right)^i \left(1 - \frac{k}{m}\right)^{n-i} - \frac{\binom{m-n}{k-i}}{\binom{m}{k}} \geq \left(\frac{k}{m}\right)^i \left(\left(\frac{m-k}{m}\right)^{n-i} - \left(\frac{m-k}{m-i}\right)^{n-i}\right) \\ &= -\left(\frac{k}{m}\right)^i \left(\left(\frac{m-k}{m-i}\right)^{n-i} - \left(\frac{m-k}{m}\right)^{n-i}\right) \\ &\geq -\left(\frac{k}{m}\right)^i \left(\frac{m-k}{m-i} - \frac{m-k}{m}\right)(n-i) = -\left(\frac{k}{m}\right)^i \frac{(m-k)i(n-i)}{(m-i)m}.\end{aligned}$$

Since $0 \leq \frac{k}{m} \leq 1$, $0 \leq \frac{m-k}{m-i} \leq 1$, and $0 \leq i \leq n$, we obtain

$$-\left(\frac{k}{m}\right)^i \frac{(m-k)i(n-i)}{(m-i)m} \geq -\frac{i(n-i)}{m} > -\frac{n^2}{m}.$$

Therefore,

$$\delta = \left(\frac{k}{m}\right)^i \left(1 - \frac{k}{m}\right)^{n-i} - \frac{\binom{m-n}{k-i}}{\binom{m}{k}} > -\frac{n^2}{m}. \tag{A.12}$$

Similarly, we obtain a lower bound for $\frac{\binom{m-n}{k-i}}{\binom{m}{k}}$ by replacing the index j in Equation (A.10) with i in the first product and with $n-i$ in the second product, obtaining

$$\begin{aligned}
\frac{\binom{m-n}{k-i}}{\binom{m}{k}} &= \prod_{j=0}^{i-1}\left(1 - \frac{m-k}{m-j}\right) \cdot \prod_{j=0}^{n-i-1}\left(1 - \frac{k-i}{m-i-j}\right) \\
&\geq \prod_{j=0}^{i-1}\left(1 - \frac{m-k}{m-i}\right) \cdot \prod_{j=0}^{n-i-1}\left(1 - \frac{k-i}{m-n}\right) \\
&= \left(\frac{k-i}{m-i}\right)^i \left(\frac{m-n-k+i}{m-n}\right)^{n-i} \geq \left(\frac{k-i}{m-i}\right)^i \left(\frac{m-n-k+i}{m-n+i}\right)^{n-i}.
\end{aligned}$$

Thus, proceeding as above, we have

$$\begin{aligned}
\delta &= \left(\frac{k}{m}\right)^i \left(1 - \frac{k}{m}\right)^{n-i} - \frac{\binom{m-n}{k-i}}{\binom{m}{k}} \\
&\leq \left(\frac{k}{m}\right)^i \left(\frac{m-k}{m}\right)^{n-i} - \left(\frac{k-i}{m-i}\right)^i \left(\frac{m-n-k+i}{m-n+i}\right)^{n-i} \\
&= \left[\left(\frac{k}{m}\right)^i - \left(\frac{k-i}{m-i}\right)^i\right] \left(\frac{m-k}{m}\right)^{n-i} \\
&\quad + \left[\left(\frac{m-k}{m}\right)^{n-i} - \left(\frac{m-n-k+i}{m-n+i}\right)^{n-i}\right] \left(\frac{k-i}{m-i}\right)^i.
\end{aligned}$$

Due to Equation (A.8), we have

$$0 \leq \frac{k-i}{m-i} \leq \frac{k}{m} \leq 1, \quad 0 \leq \frac{m-n-k+i}{m-n+i} \leq \frac{m-k}{m} \leq 1,$$

and so we obtain

$$\delta \leq \left(\frac{k}{m}\right)^i - \left(\frac{k-i}{m-i}\right)^i + \left(\frac{m-k}{m}\right)^{n-i} - \left(\frac{m-n-k+i}{m-n+i}\right)^{n-i}. \tag{A.13}$$

Applying Equation (A.11) twice to the right-hand side of Equation (A.13), we obtain

$$\begin{aligned}
\delta &\leq i\left(\frac{k}{m} - \frac{k-i}{m-i}\right) + (n-i)\left(\frac{m-k}{m} - \frac{m-n-k+i}{m-n+i}\right) \\
&= \frac{i^2}{m} \cdot \frac{m-k}{m-i} + \frac{(n-i)^2}{m} \cdot \frac{k}{m-n+i}.
\end{aligned}$$

From Equation (A.8), we have

$$0 \leq \frac{m-k}{m-i} \leq 1, \quad 0 \leq \frac{k}{m-n+i} \leq 1.$$

Therefore,

$$\delta = \left(\frac{k}{m}\right)^i \left(1 - \frac{k}{m}\right)^{n-i} - \frac{\binom{m-n}{k-i}}{\binom{m}{k}} \leq \frac{i^2 + (n-i)^2}{m} \leq \frac{ni + n(n-i)}{m} = \frac{n^2}{m}. \tag{A.14}$$

Equations (A.12) and (A.14) together yield Equation (A.9). □

Now we give a proof of Theorem 1.

Theorem 1

Let g be a polynomial of degree $n \geq 0$. For any $\epsilon > 0$, there exists a positive integer $M \geq n$ such that for all integer $m \geq M$ and $k = 0, 1, \ldots, m$, we have

$$\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| < \epsilon,$$

where $\beta_{0,m}, \beta_{1,m}, \ldots, \beta_{m,m}$ satisfy that $g(t) = \sum_{k=0}^{m} \beta_{k,m} b_{k,m}(t)$. □

Proof. For $n = 0$, g is a constant polynomial. Suppose that $g(t) = y$, where y is a constant value. We select $M = 1$. Then, for all integers $m \geq M$ and all integers $k = 0, 1, \ldots, m$, we have $\beta_{k,m} = y = g\left(\frac{k}{m}\right)$. Thus, the theorem holds.

For $n > 0$, we select M such that $M > \max\left\{\frac{n^2}{\epsilon}\sum_{i=0}^{n}|c_{i,n}|, 2n\right\}$, where the real numbers $c_{0,n}, c_{1,n}, \ldots, c_{n,n}$ satisfy

$$g(t) = \sum_{i=0}^{n} c_{i,n} t^i (1-t)^{n-i}. \tag{A.15}$$

Now consider any $m \geq M$. Since

$$2n \leq \max\left\{\frac{n^2}{\epsilon}\sum_{i=0}^{n}|c_{i,n}|, 2n\right\} < M \leq m,$$

we have $m - n > n$. Consider the following three cases for k.

1. The case where $n \le k \le m-n$. Here $\max\{0, k-m+n\} = 0$ and $\min\{k,n\} = n$. Thus, the summation indices in Equation (A.7) range from 0 to n. Therefore,

$$\beta_{k,m} = \frac{c_{k,m}}{\binom{m}{k}} = \sum_{i=0}^{n} \frac{\binom{m-n}{k-i}}{\binom{m}{k}} c_{i,n}. \tag{A.16}$$

Substituting t with $\frac{k}{m}$ in Equation (A.15), we have

$$g\left(\frac{k}{m}\right) = \sum_{i=0}^{n} c_{i,n} \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}. \tag{A.17}$$

By Lemma 12, since $0 < n < m$ and $0 \le k \le m$, Equation (A.9) holds for all $0 = \max\{0, k-m+n\} \le i \le \min\{k,n\} = n$. Thus, by Equations (A.9), (A.16), (A.17) and the well-known inequality $|\sum t_i| \le \sum |t_i|$, we have

$$\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| = \left|\sum_{i=0}^{n}\left[\frac{\binom{m-n}{k-i}}{\binom{m}{k}} - \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}\right] c_{i,n}\right|$$
$$\le \sum_{i=0}^{n}\left|\frac{\binom{m-n}{k-i}}{\binom{m}{k}} - \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}\right| |c_{i,n}| \le \frac{n^2}{m}\sum_{i=0}^{n}|c_{i,n}|.$$

Since $\frac{n^2}{\epsilon}\sum_{i=0}^{n}|c_{i,n}| < M \le m$, we have

$$\frac{n^2}{m}\sum_{i=0}^{n}|c_{i,n}| < \epsilon. \tag{A.18}$$

Therefore, for all $n \le k \le m-n$, we have $\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| < \epsilon$.

2. The case where $0 \le k < n$. Since $m > 2n$, we have $k-m+n < k-n < 0$. Thus, $\max\{0, k-m+n\} = 0$ and $\min\{k,n\} = k$. Thus, the summation indices in Equation (A.7) range from 0 to k. Therefore,

$$\beta_{k,m} = \frac{c_{k,m}}{\binom{m}{k}} = \sum_{i=0}^{k} \frac{\binom{m-n}{k-i}}{\binom{m}{k}} c_{i,n}. \tag{A.19}$$

When $k+1 \leq i \leq n$, we have that $1 \leq k+1 \leq i$ and so

$$\left|\left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}\right| = \left(\frac{k}{m}\right)\left|\left(\frac{k}{m}\right)^{i-1}\left(1-\frac{k}{m}\right)^{n-i}\right| \leq \frac{k}{m} < \frac{n}{m} \leq \frac{n^2}{m}. \quad \text{(A.20)}$$

By Lemma 12, since $0 < n < m$ and $0 \leq k \leq m$, Equation (A.9) holds for all $0 = \max\{0, k-m+n\} \leq i \leq \min\{k,n\} = k$. Thus, by Equations (A.9), (A.17), (A.18), (A.19), (A.20) and the inequality $|\sum t_i| \leq \sum |t_i|$, we have

$$\begin{aligned}
&\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| = \left|\sum_{i=0}^{k} \frac{\binom{m-n}{k-i}}{\binom{m}{k}} c_{i,n} - \sum_{i=0}^{n}\left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i} c_{i,n}\right| \\
&= \left|\sum_{i=0}^{k}\left[\frac{\binom{m-n}{k-i}}{\binom{m}{k}} - \left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i}\right] c_{i,n} - \sum_{i=k+1}^{n}\left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i} c_{i,n}\right| \\
&\leq \sum_{i=0}^{k}\left|\frac{\binom{m-n}{k-i}}{\binom{m}{k}} - \left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i}\right| |c_{i,n}| + \sum_{i=k+1}^{n}\left|\left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i}\right| |c_{i,n}| \\
&\leq \frac{n^2}{m}\sum_{i=0}^{n}|c_{i,n}| < \epsilon.
\end{aligned}$$

3. The case where $m-n < k \leq m$. Since $m > 2n$, we have $n < m-n < k$. Thus, $\max\{0, k-m+n\} = k-m+n$ and $\min\{k,n\} = n$. Now, the summation indices in Equation (A.7) range from $k-m+n$ to n. Therefore,

$$\beta_{k,m} = \frac{c_{k,m}}{\binom{m}{k}} = \sum_{i=k-m+n}^{n} \frac{\binom{m-n}{k-i}}{\binom{m}{k}} c_{i,n}. \quad \text{(A.21)}$$

When $0 \leq i \leq k-m+n-1$, we have that $1 \leq m+1-k \leq n-i$. Thus,

$$\begin{aligned}
\left|\left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i}\right| &= \left(1-\frac{k}{m}\right)\left|\left(\frac{k}{m}\right)^i\left(1-\frac{k}{m}\right)^{n-i-1}\right| \\
&\leq \frac{m-k}{m} < \frac{n}{m} \leq \frac{n^2}{m}.
\end{aligned} \quad \text{(A.22)}$$

By Lemma 12, since $0 < n < m$ and $0 \leq k \leq m$, Equation (A.9) holds for all $k-m+n = \max\{0, k-m+n\} \leq i \leq \min\{k,n\} = n$. Thus, by Equations (A.9), (A.17), (A.18), (A.21), (A.22) and the inequality $|\sum t_i| \le \sum |t_i|$, we have

$$\begin{aligned}
&\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| = \left|\sum_{i=k-m+n}^{n} \frac{\binom{m-n}{k-i}}{\binom{m}{k}} c_{i,n} - \sum_{i=0}^{n} \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i} c_{i,n}\right| \\
&= \left|\sum_{i=k-m+n}^{n} \left[\frac{\binom{m-n}{k-i}}{\binom{m}{k}} - \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}\right] c_{i,n} - \sum_{i=0}^{k-m+n-1} \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i} c_{i,n}\right| \\
&\le \sum_{i=k-m+n}^{n} \left|\frac{\binom{m-n}{k-i}}{\binom{m}{k}} - \left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}\right| |c_{i,n}| + \sum_{i=0}^{k-m+n-1} \left|\left(\frac{k}{m}\right)^i \left(1-\frac{k}{m}\right)^{n-i}\right| |c_{i,n}| \\
&\le \frac{n^2}{m} \sum_{i=0}^{n} |c_{i,n}| < \epsilon.
\end{aligned}$$

In conclusion, if $m \ge M$, then for all $k = 0, 1, \ldots, m$, we have

$$\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| < \epsilon. \qquad \square$$

Appendix B

A Proof of Theorem 2

In this section, we demonstrate that the sets U and V defined in Definitions 2 and 3 are one and the same. We demonstrate that $U \subseteq V$ and $V \subseteq U$ separately. First, we prove the former – the easier one. Then we use Theorem 1 to prove the latter.

Theorem 27

$$U \subseteq V. \qquad \square$$

Proof. Let $n \geq 1$ and $\beta_{k,n} = 0$, for all $0 \leq k \leq n$. Then the polynomial

$$p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) = 0.$$

Let $n \geq 1$ and $\beta_{k,n} = 1$, for all $0 \leq k \leq n$. Then, by Equation (2.4), the polynomial

$$p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) = 1.$$

Thus $0 \in U$ and $1 \in U$. From the definition of V, $0 \in V$ and $1 \in V$.

Now consider any polynomial $p \in U$ such that $p \not\equiv 0$ and $p \not\equiv 1$. There exist $n \geq 1$ and $0 \leq \beta_{0,n}, \beta_{1,n}, \ldots, \beta_{n,n} \leq 1$ such that

$$p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t).$$

From Equations (2.3), (2.4) and the fact that $0 \leq \beta_{0,n}, \beta_{1,n}, \ldots, \beta_{n,n} \leq 1$, for all t in $[0,1]$, we have

$$0 \leq p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) \leq \sum_{k=0}^{n} b_{k,n}(t) = 1.$$

We further claim that for all t in $(0,1)$, we must have $0 < p(t) < 1$. By contraposition, we assume that there exists a $0 < t_0 < 1$, such that $p(t_0) \leq 0$ or $p(t_0) \geq 1$. Since for $0 < t_0 < 1$, we have $0 \leq p(t_0) \leq 1$, thus $p(t_0) = 0$ or 1.

We first consider the case that $p(t_0) = 0$. Since $0 < t_0 < 1$, it is not hard to see that for all $k = 0, 1, \ldots, n$, $b_{k,n}(t_0) > 0$. Thus, $p(t_0) = 0$ implies that for all $k = 0, 1, \ldots, n$, $\beta_{k,n} = 0$. In this case, for any real number t, $p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) = 0$, which contradicts the assumption that $p(t) \not\equiv 0$.

Similarly, in the case that $p(t_0) = 1$, we can show that $p(t) \equiv 1$, which contradicts the assumption that $p(t) \not\equiv 1$. In both cases, we get a contradiction; this proves the claim that for all t in $(0,1)$, $0 < p(t) < 1$.

Therefore, for any polynomial $p \in U$ such that $p \not\equiv 0$ and $p \not\equiv 1$, we have $p \in V$. Since we showed at the outset that $0 \in U$, $1 \in U$, $0 \in V$ and $1 \in V$, thus, for any polynomial $p \in U$, we have $p \in V$. Therefore, $U \subseteq V$. $\square$

Next we prove the claim that $V \subseteq U$. We will first show that each of four possible different categories of polynomials in the set V are in the set U. The different categories are tackled in Theorems 28 and 29 and Corollaries 4 and 5.

Theorem 28

Let g be a polynomial of degree n mapping the open interval $(0,1)$ into $(0,1)$ with $0 \leq g(0), g(1) < 1$. Then $g \in U$. $\square$

Proof. Since g is continuous on the closed interval $[0,1]$, it attains its maximum value M_g on $[0,1]$. Since $g(t) < 1$, for all $t \in [0,1]$, we have $M_g < 1$.

Let $\epsilon_1 = 1 - M_g > 0$. By Theorem 1, there exists a positive integer $M_1 \geq n$ such that for all integers $m \geq M_1$ and $k = 0, 1, \ldots, m$, we have $\left|\beta_{k,m} - g\left(\frac{k}{m}\right)\right| < \epsilon_1$, where

$\beta_{0,m}, \beta_{1,m}, \ldots, \beta_{m,m}$ satisfy that $g(t) = \sum_{k=0}^{m} \beta_{k,m} b_{k,m}(t)$. Thus, for all $m \geq M_1$ and all $k = 0, 1, \ldots, m$,

$$\beta_{k,m} < g\left(\frac{k}{m}\right) + \epsilon_1 \leq M_g + 1 - M_g = 1. \tag{B.1}$$

Denote by r the multiplicity of 0 as a root of $g(t)$ (where $r = 0$ if $g(0) \neq 0$) and by s the multiplicity of 1 as a root of $g(t)$ (where $s = 0$ if $g(1) \neq 0$). We can factorize $g(t)$ as

$$g(t) = t^r (1-t)^s h(t), \tag{B.2}$$

where $h(t)$ is a polynomial, satisfying that $h(0) \neq 0$ and $h(1) \neq 0$.

We show that $h(0) > 0$. By the way of contraposition, suppose that $h(0) \leq 0$. Since $h(0) \neq 0$, we have $h(0) < 0$. By the continuity of the polynomial $h(t)$, there exists some $0 < t^* < 1$, such that $h(t^*) < 0$. Thus, $g(t^*) = t^{*r}(1-t^*)^s h(t^*) < 0$. However, $g(t) > 0$, for all $t \in (0, 1)$. Therefore, $h(0) > 0$. Similarly, we have $h(1) > 0$.

Since $g(t) > 0$ for all t in $(0, 1)$, we have $h(t) = \dfrac{g(t)}{t^r(1-t)^s} > 0$ for all t in $(0, 1)$. In view of the fact that $h(0) > 0$ and $h(1) > 0$, we have $h(t) > 0$, for all t in $[0, 1]$. Since $h(t)$ is continuous on the closed interval $[0, 1]$, it attains its minimum value m_h on $[0, 1]$. Clearly, $m_h > 0$.

Let $\epsilon_2 = m_h > 0$. By Theorem 1, there exists a positive integer $M_2 \geq n - r - s$, such that for all integers $d \geq M_2$ and $k = 0, 1, \ldots, d$, we have $\left|\gamma_{k,d} - h\left(\frac{k}{d}\right)\right| < \epsilon_2$, where $\gamma_{0,d}, \gamma_{1,d}, \ldots, \gamma_{d,d}$ satisfy that

$$h(t) = \sum_{k=0}^{d} \gamma_{k,d} b_{k,d}(t). \tag{B.3}$$

Thus, for all $d \geq M_2$ and all $k = 0, 1, \ldots, d$,

$$\gamma_{k,d} > h\left(\frac{k}{d}\right) - \epsilon_2 \geq m_h - m_h = 0.$$

Combining Equations (B.2) and(B.3), we have

$$\begin{aligned} g(t) &= t^r(1-t)^s h(t) = t^r(1-t)^s \sum_{k=0}^{d} \gamma_{k,d} b_{k,d}(t) = t^r(1-t)^s \sum_{k=0}^{d} \gamma_{k,d} \binom{d}{k} t^k (1-t)^{d-k} \\ &= \sum_{k=0}^{d} \frac{\gamma_{k,d}\binom{d}{k}}{\binom{d+r+s}{k+r}} \binom{d+r+s}{k+r} t^{k+r}(1-t)^{d+s-k} = \sum_{k=r}^{d+r} \frac{\gamma_{k-r,d}\binom{d}{k-r}}{\binom{d+r+s}{k}} b_{k,d+r+s}(t) \\ &= \sum_{k=0}^{d+r+s} \beta_{k,d+r+s} b_{k,d+r+s}(t), \end{aligned}$$

where $\beta_{k,d+r+s}$ are the coefficients of the Bernstein polynomial of degree $d+r+s$ of g and

$$\beta_{k,d+r+s} = \begin{cases} 0, & \text{for } 0 \le k < r \text{ and } d+r < k \le d+r+s \\ \frac{\gamma_{k-r,d}\binom{d}{k-r}}{\binom{d+r+s}{k}} > 0, & \text{for } r \le k \le d+r. \end{cases}$$

Thus, when $m = d+r+s \ge M_2 + r + s$, we have for all $k = 0, 1, \ldots, m$,

$$\beta_{k,m} \ge 0. \tag{B.4}$$

According to Equations (B.1) and (B.4), if we select an $m_0 \ge \max\{M_1, M_2 + r + s\}$, then $g(t)$ can be expressed as a Bernstein polynomial of degree m_0:

$$g(t) = \sum_{k=0}^{m_0} \beta_{k,m_0} b_{k,m_0}(t),$$

with $0 \le \beta_{k,m_0} \le 1$, for all $k = 0, 1, \ldots, m_0$. Therefore, $g \in U$. $\square$

Theorem 29

Let g be a polynomial of degree n mapping the open interval $(0,1)$ into $(0,1)$ with $g(0) = 0$ and $g(1) = 1$. Then $g \in U$. $\square$

PROOF. Denote by r the multiplicity of 0 as a root of $g(t)$. We can factorize $g(t)$ as

$$g(t) = t^r h(t), \tag{B.5}$$

where $h(t)$ is a polynomial satisfying $h(0) \neq 0$. By a similar reasoning as in the proof of Theorem 28, we obtain $h(0) > 0$. Since for all t in $(0,1]$, $h(t) = \dfrac{g(t)}{t^r} > 0$, we have for all t in $[0,1]$, $h(t) > 0$. Since $h(t)$ is continuous on the closed interval $[0,1]$, it attains its minimum value m_h on $[0,1]$. Clearly, $m_h > 0$.

Let $\epsilon_1 = m_h > 0$. By Theorem 1, there exists a positive integer $M_1 \geq n - r$ such that for all integers $d \geq M_1$ and $k = 0, 1, \ldots, d$, we have $\left|\gamma_{k,d} - h\left(\frac{k}{d}\right)\right| < \epsilon_1$, where $\gamma_{0,d}, \gamma_{1,d}, \ldots, \gamma_{d,d}$ satisfy

$$h(t) = \sum_{k=0}^{d} \gamma_{k,d} b_{k,d}(t). \tag{B.6}$$

Thus, for all $d \geq M_1$ and all $k = 0, 1, \ldots, d$,

$$\gamma_{k,d} > h\left(\frac{k}{d}\right) - \epsilon_1 \geq m_h - m_h = 0.$$

Combining Equations (B.5) and (B.6), we have

$$\begin{aligned} g(t) = t^r h(t) &= t^r \sum_{k=0}^{d} \gamma_{k,d} b_{k,d}(t) = t^r \sum_{k=0}^{d} \gamma_{k,d} \binom{d}{k} t^k (1-t)^{d-k} \\ &= \sum_{k=0}^{d} \frac{\gamma_{k,d}\binom{d}{k}}{\binom{d+r}{k+r}} \binom{d+r}{k+r} t^{k+r}(1-t)^{d-k} = \sum_{k=r}^{d+r} \frac{\gamma_{k-r,d}\binom{d}{k-r}}{\binom{d+r}{k}} b_{k,d+r}(t) = \sum_{k=0}^{d+r} \beta_{k,d+r} b_{k,d+r}(t), \end{aligned}$$

where $\beta_{k,d+r}$ are the coefficients of the Bernstein polynomial of degree $d+r$ of g and

$$\beta_{k,d+r} = \begin{cases} 0, & \text{for } 0 \leq k < r \\ \frac{\gamma_{k-r,d}\binom{d}{k-r}}{\binom{d+r}{k}} > 0, & \text{for } r \leq k \leq d+r. \end{cases}$$

Thus, when $m = d + r \geq M_1 + r$, we have for all $k = 0, 1, \ldots, m$,

$$\beta_{k,m} \geq 0. \tag{B.7}$$

Let

$$g^*(t) = 1 - g(t). \tag{B.8}$$

Then g^* maps the open interval $(0,1)$ into $(0,1)$ with $g^*(0) = 1$, $g^*(1) = 0$. Denote by s the multiplicity of 1 as a root of $g^*(t)$. Thus, we can factorize $g^*(t)$ as

$$g^*(t) = (1-t)^s h^*(t), \tag{B.9}$$

where $h^*(t)$ is a polynomial satisfying that $h^*(1) \neq 0$. As in the proof of Theorem 28, we obtain $h^*(1) > 0$. Since for all t in $[0,1)$, $h^*(t) = \dfrac{g^*(t)}{(1-t)^s} > 0$, we have for all $t \in [0,1]$, $h^*(t) > 0$. Since $h^*(t)$ is continuous on the closed interval $[0,1]$, it attains its minimum value m_h^* on $[0,1]$. Clearly, $m_h^* > 0$.

Let $\epsilon_2 = m_h^* > 0$. By Theorem 1, there exists a positive integer $M_2 \geq n-s$ such that for all integers $q \geq M_2$ and $k = 0, 1, \ldots, q$, we have $\left| \gamma_{k,q}^* - h^*\left(\dfrac{k}{q}\right) \right| < \epsilon_2$, where $\gamma_{0,q}^*, \gamma_{1,q}^*, \ldots, \gamma_{q,q}^*$ satisfy

$$h^*(t) = \sum_{k=0}^{q} \gamma_{k,q}^* b_{k,q}(t). \tag{B.10}$$

Thus, for all $q \geq M_2$ and all $k = 0, 1, \ldots, q$,

$$\gamma_{k,q}^* > h^*\left(\frac{k}{q}\right) - \epsilon_2 \geq m_h^* - m_h^* = 0.$$

Combining Equations (B.8), (B.9) and (B.10), we have

$$\begin{aligned} g(t) &= 1 - g^*(t) = 1 - (1-t)^s h^*(t) = 1 - (1-t)^s \sum_{k=0}^{q} \gamma_{k,q}^* b_{k,q}(t) \\ &= 1 - (1-t)^s \sum_{k=0}^{q} \gamma_{k,q}^* \binom{q}{k} t^k (1-t)^{q-k} = 1 - \sum_{k=0}^{q} \frac{\gamma_{k,q}^* \binom{q}{k}}{\binom{q+s}{k}} \binom{q+s}{k} t^k (1-t)^{q+s-k}. \end{aligned}$$

Further using (2.4), we obtain

$$g(t) = \sum_{k=0}^{q+s} b_{k,q+s}(t) - \sum_{k=0}^{q} \frac{\gamma_{k,q}^* \binom{q}{k}}{\binom{q+s}{k}} b_{k,q+s}(t) = \sum_{k=0}^{q+s} \beta_{k,q+s} b_{k,q+s}(t),$$

where the $\beta_{k,q+s}$'s are the coefficients of the Bernstein polynomial of degree $q+s$ of g:

$$\beta_{k,q+s} = \begin{cases} 1 - \frac{\gamma_{k,q}^* \binom{q}{k}}{\binom{q+s}{k}} < 1, & \text{for } 0 \leq k \leq q \\ 1, & \text{for } q < k \leq q+s. \end{cases}$$

Thus, when $m = q + s \geq M_2 + s$, we have for all $k = 0, 1, \ldots, m$,

$$\beta_{k,m} \leq 1. \tag{B.11}$$

According to Equations (B.7) and (B.11), if we select an $m_0 \geq \max\{M_1 + r, M_2 + s\}$, then $g(t)$ can be expressed as a Bernstein polynomial of degree m_0:

$$g(t) = \sum_{k=0}^{m_0} \beta_{k,m_0} b_{k,m_0}(t),$$

with $0 \leq \beta_{k,m_0} \leq 1$, for all $k = 0, 1, \ldots, m_0$, Therefore, $g \in U$. $\square$

Lemma 13

If a polynomial p is in the set U, then the polynomial $1 - p$ is also in the set U. $\square$

PROOF. Since p is in the set U, there exist $n \geq 1$ and $0 \leq \beta_{0,n}, \beta_{1,n}, \ldots, \beta_{n,n} \leq 1$ such that

$$p(t) = \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t).$$

By Equation (2.4), we have

$$1 - p(t) = \sum_{k=0}^{n} b_{k,n}(t) - \sum_{k=0}^{n} \beta_{k,n} b_{k,n}(t) = \sum_{k=0}^{n} (1 - \beta_{k,n}) b_{k,n}(t) = \sum_{k=0}^{n} \gamma_{k,n} b_{k,n}(t),$$

where $\gamma_{k,n} = 1 - \beta_{k,n}$ satisfying $0 \leq \gamma_{k,n} \leq 1$, for all $k = 0, 1, \ldots, n$. Therefore, $1 - p$ is in the set U. $\square$

Corollary 4

Let g be a polynomial of degree n mapping the open interval $(0, 1)$ into $(0, 1)$ with $0 < g(0), g(1) \leq 1$. Then $g \in U$. $\square$

PROOF. Let polynomial $h = 1 - g$. Then h maps $(0, 1)$ into $(0, 1)$ with $0 \leq h(0), h(1) < 1$. By Theorem 28, $h \in U$. By Lemma 13, $g = 1 - h$ is also in the set U. $\square$

Corollary 5

Let g be a polynomial of degree n mapping the open interval $(0, 1)$ into $(0, 1)$ with $g(0) = 1$ and $g(1) = 0$. Then $g \in U$. $\square$

PROOF. Let the polynomial $h = 1 - g$. Then h maps $(0, 1)$ into $(0, 1)$ with $h(0) = 0, h(1) = 1$. By Theorem 29, $h \in U$. By Lemma 13, $g = 1 - h$ is also in the set U. $\square$

Combining Theorem 28, Theorem 29, Corollary 4 and Corollary 5, we show that $V \subseteq U$.

Theorem 30

$$V \subseteq U. \qquad \square$$

PROOF. Based on the definition of V, for any polynomial $p \in V$, we have one of following five cases.

1. The case where $p \equiv 0$ or $p \equiv 1$. In the proof of Theorem 27, we have shown that $0 \in U$ and $1 \in U$. Thus $p \in U$.
2. The case where p maps the open interval $(0, 1)$ into $(0, 1)$ with $0 \leq p(0), p(1) < 1$. By Theorem 28, $p \in U$.
3. The case where p maps the open interval $(0, 1)$ into $(0, 1)$ with $0 < p(0), p(1) \leq 1$. By Corollary 4, $p \in U$.
4. The case where p maps the open interval $(0, 1)$ into $(0, 1)$ with $p(0) = 0$ and $p(1) = 1$. By Theorem 29, $p \in U$.
5. The case where p maps the open interval $(0, 1)$ into $(0, 1)$ with $p(0) = 1$ and $p(1) = 0$. By Corollary 5, $p \in U$.

In summary, for any polynomial $p \in V$, we have $p \in U$. Thus, $V \subseteq U$. $\square$

Based on Theorems 27 and 30, we have proved Theorem 2.

Appendix C

A Proof of Theorem 11

We first show that the ideal values $p^*_{i_k}$ are all in the unit interval.

Lemma 14

For all $0 \leq k \leq n+1$, $0 \leq p^*_{i_k} \leq 1$. $\Box$

Proof. We prove the claim by induction.

Base case: Since $p^*_{i_0} = q$, we have $0 \leq p^*_{i_0} \leq 1$.

Inductive step: Assume that for some $0 \leq k \leq n$, $0 \leq p^*_{i_k} \leq 1$. Based on our algorithm, when $p^*_{i_k} > p_{i_k}$, we have

$$p^*_{i_{k+1}} = \frac{p^*_{i_k} - p_{i_k}}{1 - p_{i_k}} \text{ or } \frac{1 - p^*_{i_k}}{1 - p_{i_k}}.$$

It is not hard to see that $0 \leq p^*_{i_{k+1}} \leq 1$.

When $p^*_{i_k} \leq p_{i_k}$, we have

$$p^*_{i_{k+1}} = \frac{p^*_{i_k}}{p_{i_k}} \text{ or } 1 - \frac{p^*_{i_k}}{p_{i_k}}.$$

It is not hard to see that $0 \leq p^*_{i_{k+1}} \leq 1$.

Thus, the statement holds for $k+1$. This completes the inductive proof. □

We can now prove Theorem 11.

Theorem 11

In Scenario Two, given a set $S = \{p_1, p_2, \ldots, p_n\}$ and a target probability q, let p be the output probability of the circuit constructed by the greedy algorithm. We have

$$|p - q| \leq \frac{1}{2} \prod_{k=1}^{n} \max\{p_k, 1 - p_k\}. \qquad \square$$

Proof. Let w be the output probability of the circuit C_{n+1}. Since we choose the circuit that has the smallest difference between its output probability and the output probability q among the circuits $C_0, \ldots, C_{n+1}$ as the final construction, we have $|p-q| \leq |w-q|$. We only need to prove that

$$|w - q| \leq \frac{1}{2} \prod_{k=1}^{n} \max\{p_k, 1 - p_k\}.$$

Based on our algorithm, the circuit C_{n+1} is a concatenation of $n+1$ logic gates, each being either an AND gate or an OR gate. Denote the output probability of the i-th gate from the beginning as w_i.

Suppose that $P(x_{n+1} = 1) = p_{i_{n+1}} \in \{0, 1\}$. Based on our choice of $p_{i_{n+1}}$, we have

$$|p_{i_{n+1}} - p^*_{i_{n+1}}| = \min\{|p^*_{i_{n+1}}|, |1 - p^*_{i_{n+1}}|\}.$$

Thus,

$$|p_{i_{n+1}} - p^*_{i_{n+1}}| \leq \frac{1}{2}(|p^*_{i_{n+1}}| + |1 - p^*_{i_{n+1}}|).$$

From Lemma 14, we have $0 \leq p^*_{i_{n+1}} \leq 1$. Thus, we obtain

$$|p_{i_{n+1}} - p^*_{i_{n+1}}| \leq \frac{1}{2}. \tag{C.1}$$

Next, we will show by induction that for all $1 \leq k \leq n+1$, we have

$$|w_k - p^*_{i_{n+1-k}}| \leq \frac{1}{2} \prod_{j=1}^{k} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \tag{C.2}$$

Base case: If the first gate is an OR gate, then we have

$$w_1 = p_{i_n} + (1 - p_{i_n})p_{i_{n+1}}.$$

From Equation (4.15), we have

$$p^*_{i_n} = p_{i_n} + (1 - p_{i_n})p^*_{i_{n+1}}.$$

Thus,

$$|w_1 - p^*_{i_n}| = (1 - p_{i_n})|p_{i_{n+1}} - p^*_{i_{n+1}}|.$$

Applying Equation (C.1), we have

$$\begin{aligned} |w_1 - p^*_{i_n}| &\le \frac{1}{2}(1 - p_{i_n}) \le \frac{1}{2}\max\{p_{i_n}, 1 - p_{i_n}\} \\ &= \frac{1}{2}\prod_{j=1}^{1}\max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \end{aligned} \tag{C.3}$$

Similarly, if the first gate is an AND gate, we can also get Equation (C.3). Thus, the statement holds for the base case.

Inductive step: Assume that the statement holds for some $1 \le k \le n$. Now consider $k + 1$. Based on our algorithm, there are four cases:

1. The $(k + 1)$-th gate from the beginning is an OR gate with one input connected to the output of the k-th gate.
2. The $(k + 1)$-th gate from the beginning is an OR gate with one input connected to the inverted output of the k-th gate.
3. The $(k + 1)$-th gate from the beginning is an AND gate with one input connected to the output of the k-th gate.
4. The $(k + 1)$-th gate from the beginning is an AND gate with one input connected to the inverted output of the k-th gate.

In the first case, we have

$$w_{k+1} = p_{i_{n-k}} + (1 - p_{i_{n-k}})w_k.$$

In this case, the relation between the ideal values $p^*_{i_{n+1-k}}$ and $p^*_{i_{n-k}}$ is

$$p^*_{i_{n-k}} = p_{i_{n-k}} + (1 - p_{i_{n-k}})p^*_{i_{n+1-k}}.$$

Thus,

$$\begin{aligned} |w_{k+1} - p^*_{i_{n-k}}| &= (1 - p_{i_{n-k}})|w_k - p^*_{i_{n+1-k}}| \\ &\le \max\{p_{i_{n-k}}, 1 - p_{i_{n-k}}\}|w_k - p^*_{i_{n+1-k}}|. \end{aligned} \tag{C.4}$$

Based on the induction hypothesis, we have

$$|w_k - p^*_{i_{n+1-k}}| \le \frac{1}{2}\prod_{j=1}^{k} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \tag{C.5}$$

Combining Equations (C.4) and (C.5), we have

$$|w_{k+1} - p^*_{i_{n-k}}| \le \frac{1}{2}\prod_{j=1}^{k+1} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \tag{C.6}$$

In the other three cases, we can similarly derive Equation (C.6). Thus, the statement holds for $k+1$. This completes the induction proof.

Note that $p_{i_0} \in \{0, 1\}$ and $\{p_{i_1}, \ldots, p_{i_n}\} = \{p_1, \ldots, p_n\}$. Thus, when $k = n+1$, Equation (C.2) can be written as

$$|w_{n+1} - p^*_{i_0}| \le \frac{1}{2}\prod_{j=1}^{n} \max\{p_j, 1 - p_j\}.$$

Based on our algorithm, we have $w_{n+1} = w$, the output probability of the circuit C_{n+1}, and $p^*_{i_0} = q$. Thus, we obtain

$$|w - q| \le \frac{1}{2}\prod_{j=1}^{n} \max\{p_j, 1 - p_j\}. \qquad \square$$