Transforming Probabilities with Combinational Logic

Weikang Qian, Marc D. Riedel, Hongchao Zhou, and Jehoshua Bruck

***Abstract*—Schemes for probabilistic computation can exploit physical sources to generate random values in the form of 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 this work, we demonstrate novel techniques for synthesizing combinational logic that transforms a set of *source* probabilities into different *target* probabilities. We consider three different 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. Further, we show that for any integer $n \geq 2$, we can find a single source probability that can be transformed into arbitrary base-n fractional probabilities of the form $\frac{m}{n^d}$. 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. In the case that the source probabilities are not specified, but once chosen cannot be duplicated, we provide an optimal choice.**

***Index Terms*—logic synthesis, combinational logic, probabilistic logic, probabilistic signals, random bit streams, stochastic bit streams**

I. Introduction and Background

Most digital circuits are designed to map *deterministic* inputs of zero and one to *deterministic* outputs of zero and one. An alternative paradigm is to design circuits that operate on *stochastic* bit streams. Each stream represents a real-valued number x $(0 \leq x \leq 1)$ through a sequence of random bits that have probability x of being one and probability $1 - x$ of being zero. Such circuits can be viewed as constructs that accept real-valued probabilities as inputs and compute real-valued probabilities as outputs.

Consider the example shown in Figure 1. Given independent stochastic bit streams as inputs, an AND gate performs multiplication: it produces an output bit stream with a probability that is the product of the probabilities of the input bit streams. In prior work, we proposed a general method for synthesizing arbitrary functions through logical computation on stochastic bit streams [1], [2].

Stochastic bit streams can be generated with pseudo-random constructs, such as linear feedback shift registers. Alternatively, if physical sources of randomness are available, these could be used directly. For example, in [3], the authors propose a so-called probabilistic CMOS (PCMOS) construct that generates random bits from intrinsic sources of noise. In [4], 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.

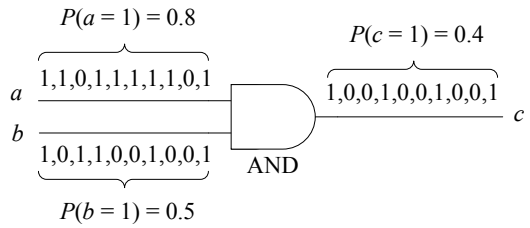

Fig. 1: An AND gate multiplies the probabilities of stochastic bit streams. Here the input streams have probabilities 0.8 and 0.5. The probability of the output stream is $0.8 \times 0.5 = 0.4$.

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 energy.

This paper presents a synthesis strategy to mitigate this issue: we describe a method for transforming a set of source probabilities into different target probabilities entirely through combinational logic. For what follows, when we say "with probability p," we mean "with a probability p of being at logical one." When we say "a circuit," we mean a combinational circuit built with logic gates.

Example 1

Suppose that we have a set of source probabilities $S = \{0.4, 0.5\}$. As illustrated in Figure 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*

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

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*

$$\begin{aligned} P(z = 1) &= P(x = 1, y = 1) \\ &= P(x = 1)P(y = 1). \end{aligned} \tag{2}$$

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

This work is supported by a grant from the Semiconductor Research Corporation's Focus Center Research Program on Functional Engineered Nano-Architectonics, contract No. 2003-NT-1107, as well as a CAREER Award, #0845650, from the National Science Foundation.

Weikang Qian and Marc Riedel are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA. Email: {qianx030, mriedel}@umn.edu.

Hongchao Zhou and Jehoshua Bruck are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125, USA. Email: {hzhou, bruck}@caltech.edu.

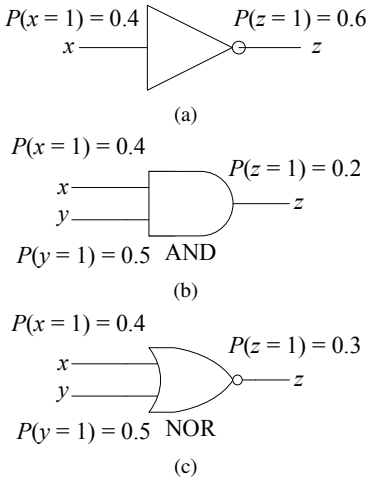

Fig. 2: An illustration of transforming a set of source probabilities into new probabilities with logic gates. (a): An inverter implementing $p_z = 1 - p_x$. (b): An AND gate implementing $p_z = p_x \cdot p_y$. (c): A NOR gate implementing $p_z = (1 - p_x) \cdot (1 - p_y)$.

0.3 since

$$P(z = 1) = P(x = 0, y = 0) = P(x = 0)P(y = 0)$$
$$= (1 - P(x = 1))(1 - P(y = 1)).$$

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

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:** Consider the situation in which we have the flexibility to choose the probabilities produced by physical sources, say by setting them with specific voltage values. We can produce multiple independent copies of each probability cheaply, since each copy uses the same voltage level. However, generating different probabilities is costly, since this entails generating different voltage levels. Here we seek to minimize the size of the source set of probabilities S, assuming that each probability in 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 values from this set into an arbitrary target probability q.
2) **Scenario Two:** Consider the situation in which there is no flexibility with the random sources; these produce a fixed set of probabilities S. The set S can be a *multiset*, i.e., one that could contain multiple elements of the same value. However, we cannot duplicate the probabilities; we have to work with what is given to us. The problem is how to synthesize logic that has input probabilities taken from S and produces an output probability q, where each element in S can be used as an input probability at most once.
3) **Scenario Three:** Consider the situation in which we have the flexibility to choose the probabilities but the values we choose cannot be duplicated cheaply; it costs as much to generate each copy as any other value. This situation occurs if we use pseudo-random constructs such as linear feedback shift registers: the cost of each pseudo-random bit stream is the same no matter what probability value is realized. Suppose that we establish a budget of n random or pseudo-random sources. 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. Here the elements of S cannot be duplicated; again, S can be a multiset.

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.

Our contributions are:

1) For Scenario One, we demonstrate that a particular set consisting of only two elements, $S = \{0.4, 0.5\}$, can be transformed into arbitrary decimal probabilities. Further, we propose an algorithm based on fraction factorization to optimize the depth of the resulting circuit. Figure 3 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.

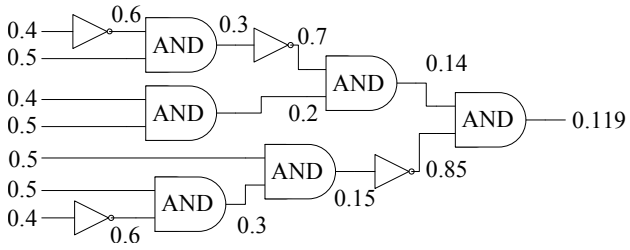

Fig. 3: A circuit synthesized by our algorithm to realize the decimal output probability 0.119 from the input probabilities 0.4 and 0.5.

2) Also for Scenario One, we prove that for any given integer $n \geq 2$, there exists a set S consisting of a single element that can be transformed into arbitrary base-n fractional probabilities of the form $\frac{m}{n^d}$.
3) For Scenario Two, we solve the problem by transforming it into a linear 0-1 programming problem. Although approximate, the solution is optimal in terms of the difference between the target probability and the actual output probability.
4) Also for Scenario Two, we provide a greedy algorithm. Although the solution that it yields is not optimal, the difference between the target probability and the actual output probability is bounded. The algorithm runs very efficiently, yielding a solution in $O(n^2)$ time, where n is the cardinality of the set S.
5) For Scenario Three, we provide an optimal choice of the set S. Specifically, we first define a quality measure $H(S)$ for each choice S consisting of arbitrary probabilities. We prove that if the cardinality of S is n, then a lower bound on $H(S)$ is $\frac{1}{4(2^{2^n} - 1)}$. Then we show that the set of source probabilities

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

achieves the lower bound.

II. Related Work

The task of *analyzing* circuits operating on probabilistic inputs is well understood [5]. Aspects such as signal correlations of reconvergent paths must be taken into account. Algorithmic details for such analysis were first fleshed out by the testing community [6]. They have also found mainstream application for tasks such as timing and power analysis [7], [8].

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 [9], [10]. 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 [11]. 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 2. 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 [4]. 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 [12]. Switching circuits were originally discussed by Shannon [13]. 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 [14]. 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.

III. 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.

A. 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 1
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. □

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 (1). An AND gate with two independent inputs performs a multiplication of the input probabilities, as was shown in Equation (2). 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. □

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.

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 ≤ z ≤ 1.}
2: Initialize ckt;
3: while GetDigits(z) > 1 do
4:     (ckt, z) ⇐ ReduceDigit(ckt, z);
5: ckt ⇐ AddBaseCkt(ckt, z); {Base case: z has at most one digit after the decimal point.}
6: return ckt;
```

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 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 1. 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.

Algorithm 2 ReduceDigit(ckt, z)

```
1: {Given a partial circuit ckt and an arbitrary decimal probability 0 ≤ z ≤ 1.}
2: n ⇐ GetDigits(z);
3: if z > 0.5 then {Case 4}
4:     z ⇐ 1 − z; AddInverter(ckt);
5: if 0.4 < z ≤ 0.5 then {Case 3}
6:     z ⇐ z/0.5; AddAND(ckt, 0.5);
7:     z ⇐ 1 − z; AddInverter(ckt);
8: if z ≤ 0.2 then {Case 1}
9:     z ⇐ z/0.4; AddAND(ckt, 0.4);
10:    z ⇐ z/0.5; AddAND(ckt, 0.5);
11:    if GetDigits(z) < n then
12:        go to END;
13:    if z > 0.5 then
14:        z ⇐ 1 − z; AddInverter(ckt);
15:    z = z/0.5; AddAND(ckt, 0.5);
16: else {Case 2: 0.2 < z ≤ 0.4}
17:    z ⇐ z/0.4; AddAND(ckt, 0.4);
18:    if GetDigits(z) < n then
19:        go to END;
20:    z ⇐ 1 − z; AddInverter(ckt);
21:    z ⇐ z/0.5; AddAND(ckt, 0.5);
22:    if GetDigits(z) < n then
23:        go to END;
24:    if z > 0.5 then
25:        z ⇐ 1 − z; AddInverter(ckt);
26:    z = z/0.5; AddAND(ckt, 0.5);
27: END: return ckt, z;
```

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 3–4 correspond to Case 4 in the proof; Lines 5–7 correspond to Case 3; Lines 8–15 correspond to Case 1; and Lines 16–26 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 after the decimal point.[1] The number of AND gates in the synthesized circuit is at most $3n$.

Example 2

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

[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.

transform 0.757 *to* 0.7*:*

$$0.757 \stackrel{1-}{\Longrightarrow} 0.243 \stackrel{/0.4}{\Longrightarrow} 0.6075 \stackrel{1-}{\Longrightarrow} 0.3925 \stackrel{/0.5}{\Longrightarrow} 0.785 \stackrel{1-}{\Longrightarrow} 0.215 \stackrel{/0.5}{\Longrightarrow} 0.43,$$
$$0.43 \stackrel{/0.5}{\Longrightarrow} 0.86 \stackrel{1-}{\Longrightarrow} 0.14 \stackrel{/0.4}{\Longrightarrow} 0.35 \stackrel{/0.5}{\Longrightarrow} 0.7.$$

Since 0.7 *can be realized as* $0.7 = 1-(1-0.4)\times 0.5$*, we obtain the circuit shown in Figure 4. (Note that here we use a black dot to represent an inverter.)* □

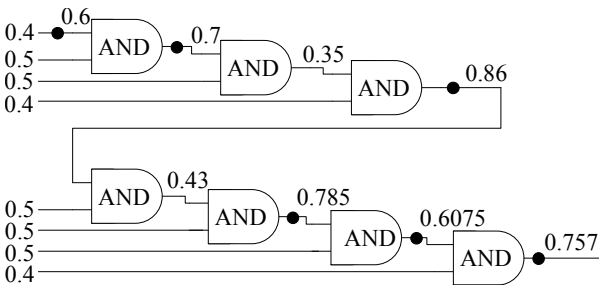

Fig. 4: A circuit transforming the set of source probabilities $\{0.4, 0.5\}$ into a decimal output probability of 0.757.

Remarks: One may question the usefulness of synthesizing a circuit that generates arbitrary *decimal* fractions. Wilhelm and Bruck proposed a scheme for synthesizing switching circuits that generate arbitrary *binary* probabilities [12]. By mapping every switch connected in series to an AND gate and every switch connected in parallel to an 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 [12], 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.

B. 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.

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 5. We refer to such optimization as *balancing*.

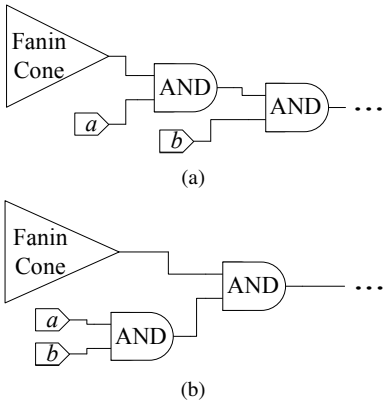

Fig. 5: An illustration of balancing to reduce the depth of the circuit. Here a and b are primary inputs. (a): The circuit before balancing. (b): The circuit after balancing.

The 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 3
Suppose we want to generate the decimal probability value 0.49.

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

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

The synthesized circuit is shown in Figure 6(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 6(b). Compared to the circuit in Figure 6(a), both the number of AND gates and the depth of the circuit are reduced.* □

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 $1 \le x \le 9$, the corresponding fraction is $x/10$. EstDepth(x) is set as the depth

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

[3] When counting depth, we ignore inverters.

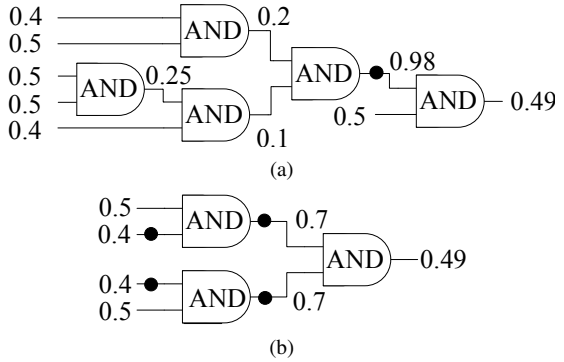

Fig. 6: Synthesizing combinational logic to generate the probability 0.49. (a): The circuit synthesized through Algorithm 1. (b): The circuit synthesized based on fraction factorization.

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

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 $\text{EstDepth}(x) = \lceil \log_{10}(x) \rceil + 1$. 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

$$\max\{\text{EstDepth}(a_l), \text{EstDepth}(a_r)\} + 1. \tag{3}$$

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 (3). 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: $\quad$ **return** 1;
8: **else if** $c_r > d_r$ **then** {The circuit w.r.t. the first pair has larger depth}
9: $\quad$ **return** -1;
10: **else**
11: $\quad$ **if** $c_l < d_l$ **then** {The circuit w.r.t. the first pair has fewer ANDs}
12: $\quad\quad$ **return** 1;
13: $\quad$ **else if** $c_l > d_l$ **then** {The circuit w.r.t. the first pair has more ANDs}
14: $\quad\quad$ **return** -1;
15: $\quad$ **else**
16: $\quad\quad$ **return** 0;

In Algorithm 3, Lines 2–5 correspond to the trivial fractions. If the fraction z is non-trivial, Lines 6–9 choose the best factor pair (u_l, u_r) of u, where u is the numerator of the fraction z. Lines 10–13 choose the best factor pair (w_l, w_r) of w, where w is the numerator of the fraction $1 - z$. Finally, Lines 14–16 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 17–20 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 21–25 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.

Finally, if it is possible to use the best factor pair, Lines 26–29 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 30–31 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).

C. Empirical Validation

We empirically validate the effectiveness of the synthesis scheme that was presented in the previous section. For logic-level optimization, we use the "balance" command of the synthesis tool ABC [15]. We find that it is very effective in reducing the depth of tree-style circuits.[4]

Table I 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 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 perform experiments on a set of target decimal probabilities that have n digits after the decimal point and average the results. Table I shows the results for n ranging from 2 to 12. When $n \leq 5$, we synthesize circuits for all possible decimal probabilities with n digits after the decimal point. When $n \geq 6$, we randomly choose 100,000 decimal probabilities with n digits after the decimal point as the synthesis targets. We show the average number of AND gates, the average depth, and the average CPU runtime.

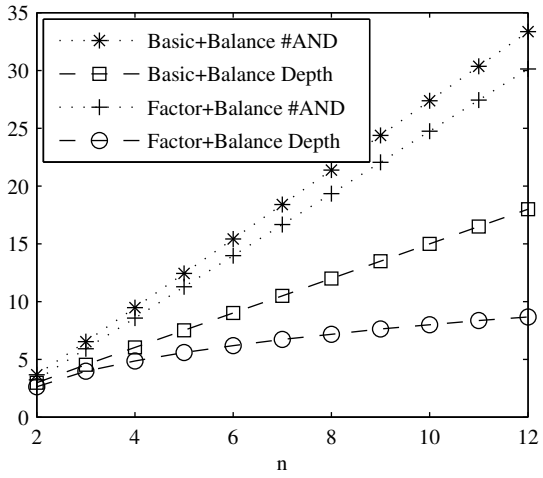

Fig. 7: Average number of AND gates and depth of the circuits versus n.

From Table I, we can see that both the "Basic+Balance" and the "Factor+Balance" synthesis schemes have only millisecond-order CPU runtimes. 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%.

In Figure 7, 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.

D. Generating Base-n Fractional Probabilities

In Section III-A, we showed that there exists a pair of probabilities that can be transformed into an arbitrary decimal probability. In [16], we show that we can further reduce the number of source probabilities down to one: there exists a real number $0 \leq r \leq 1$ that can be transformed into an arbitrary decimal probability with combinational logic. However, this number r is an irrational root of a polynomial. Here, we generalize this result. 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 2
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 8. The Boolean function of the multiplexer is:

$$y = (x_1 \wedge s) \vee (x_2 \wedge \neg s).^5$$

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 8. 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,

[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.

[5]When discussing Boolean functions, we 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.

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

Number of Digits n	Basic		Basic+Balance			Factor+Balance				
	#AND	Depth	#AND a_1	Depth d_1	Runtime (ms)	#AND a_2	Depth d_2	Runtime (ms)	#AND Imprv. (%) $100(a_1-a_2)/a_1$	Depth Imprv. (%) $100(d_1-d_2)/d_1$
2	3.67	3.67	3.67	2.98	0.22	3.22	2.62	0.22	12.1	11.9
3	6.54	6.54	6.54	4.54	0.46	5.91	3.97	0.66	9.65	12.5
4	9.47	9.47	9.47	6.04	1.13	8.57	4.86	1.34	9.45	19.4
5	12.43	12.43	12.43	7.52	0.77	11.28	5.60	0.94	9.21	25.6
6	15.40	15.40	15.40	9.01	1.09	13.96	6.17	1.48	9.36	31.5
7	18.39	18.39	18.39	10.50	0.91	16.66	6.72	1.28	9.42	35.9
8	21.38	21.38	21.38	11.99	0.89	19.34	7.16	1.35	9.55	40.3
9	24.37	24.37	24.37	13.49	0.75	22.05	7.62	1.34	9.54	43.6
10	27.37	27.37	27.37	14.98	1.09	24.74	7.98	2.41	9.61	46.7
11	30.36	30.36	30.36	16.49	0.92	27.44	8.36	2.93	9.61	49.3
12	33.35	33.35	33.35	17.98	0.73	30.13	8.66	4.13	9.65	51.8

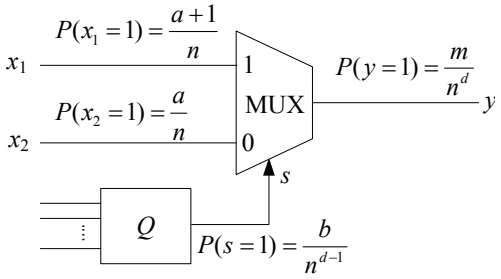

Fig. 8: The circuit generating the base-n fractional probability $\frac{m}{n^d}$, where m is written as $m = an^{d-1} + b$ with $0 \le a < n$ and $0 \le b \le n^{n-1}$. The circuit Q in the figure generates the base-n fractional probability $\frac{b}{n^{d-1}}$.

all the inputs of the multiplexer are also independent. The probability 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 $d \ge 1$.

□

Remarks:

1) An equivalent result to Theorem 2 can be found in [11]. There it is couched in information theoretic language in terms of concurrent operations on random binary sequences.
2) Our proof of Theorem 2 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 3

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

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 $R = \{p_1, p_2, \ldots, p_M\}$, we can derive a new set of probabilities $R^* = \{p_1^*, p_2^*, \ldots, p_M^*\}$, 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 $p_1, \ldots, p_M$ are rational probabilities, there exist positive integers $a_1, \ldots, a_M$ 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

$$a_M \sum_{k=1}^{h} c_k > 2^{h-1}. \tag{4}$$

In fact,

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

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) and the fact that $a_M \leq \frac{b}{2}$, we have

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

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

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

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

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

Theorems 2 and 3 lead to the following corollary.

Corollary 1
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 3, 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 2, the statement in the corollary holds. □

IV. 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.

A. 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 II. The fourth column is the probability that each input combination occurs. Here $P(x = 1) = p_x$ and $P(y = 1) = p_y$.

TABLE II: 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 $(0 \leq i \leq 2^n - 1)$ and that the output of the circuit corresponding to the i-th input combination is z_i $(z_i \in \{0,1\}, 0 \leq i \leq 2^n - 1)$. Then, the output probability is

$$p_o = \sum_{i=0}^{2^n - 1} z_i r_i. \tag{5}$$

For the example in Table II, 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 (5) 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 (5) and q is minimized. We can formulate this as the following optimization problem:

$$\text{Find } z_i \text{ that minimizes } \left| \sum_{i=0}^{2^n-1} z_i r_i - q \right| \tag{6}$$
$$\text{such that } z_i \in \{0,1\} \text{ for } i = 0, 1, \ldots, 2^n - 1. \tag{7}$$

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

Problem 1
Find z_i *that minimizes* $\text{obj}_1 = \sum_{i=0}^{2^n-1} r_i z_i - q$, *such that* $\sum_{i=0}^{2^n-1} r_i z_i - q \geq 0$ *and* $z_i \in \{0,1\}$ *for* $i = 0, 1, \ldots, 2^n - 1$.

Problem 2
Find z_i *that minimizes* $\text{obj}_2 = q - \sum_{i=0}^{2^n-1} r_i z_i$ *such that* $q - \sum_{i=0}^{2^n-1} r_i z_i \geq 0$ *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 $\text{obj}_1 = \text{obj}_1^*$ 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.[6]

B. 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 $n+1$ circuits $C_1, C_2, \ldots, C_{n+1}$ such that the circuit C_k $(1 \leq k \leq n)$ has k probabilistic inputs and the circuit C_{n+1} has n probabilistic inputs and one deterministic input of either zero or one. For all $1 \leq k \leq n$, the circuit C_{k+1} is constructed from C_k by replacing one input of C_k with a two-input gate.

The construction of the circuit C_1 is straightforward. It is achieved by either connecting a single input directly to the output or appending an inverter to a single input. As a result, its output probability is in the set

$$S_1 = \{p_1, \ldots, p_n, 1-p_1, \ldots, 1-p_n\}.$$

We can choose the number that is the closest to q in the set S_1 as its output probability and construct the circuit C_1 based on this probability. More specifically, suppose that p is the probability that is the closest to q in the set S_1. Then we have the following two cases for p.

[6]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 $i = 0, 1, \ldots, 2^n - 1$.

1) The case where $p = p_{i_1}$ for some $1 \leq i_1 \leq n$. We set the Boolean function of the circuit C_1 to $f_1(x_1) = x_1$ and set the input probability to $P(x_1 = 1) = p_{i_1}$.
2) The case where $p = 1 - p_{i_1}$ for some $1 \leq i_1 \leq n$. We set the Boolean function of the circuit C_1 to $f_1(x_1) = \neg x_1$ and set the input probability to $P(x_1 = 1) = p_{i_1}$.

In either of the two cases, in order for the circuit C_1 to realize the exact output probability q, there is an ideal value that should replace the value p_{i_1}: in the first case, the ideal value is q and in the second case, it is $1 - q$. We denote the ideal value that replaces p_{i_1} as $p_{i_1}^*$.

Now, we assume that the Boolean function of the circuit C_k is $f_k(x_1, x_2, \ldots, x_k)$ and the input probabilities are $P(x_1 = 1) = p_{i_1}, P(x_2 = 1) = p_{i_2}, \ldots, P(x_k = 1) = p_{i_k}$. 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

$$f_{k+1}(x_1, \ldots, x_{k+1}) = f_k(x_1, \ldots, x_{k-1}, g_{k+1}(x_k, x_{k+1})),$$

where $g_{k+1}(x_k, x_{k+1})$ is a Boolean function on two variables. We keep the probabilities of the inputs $x_1, x_2, \ldots, x_k$ 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 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 choices for the new 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 closest to $p_{i_k}^*$. Equivalently, p is the value in the set S_{k+1} that is closest to the value

$$\frac{p_{i_k}^* - p_{i_k}}{1 - p_{i_k}}.$$

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

$$p_{i_k} + (1 - p_{i_k})p^*_{i_{k+1}} = p^*_{i_k}, \tag{8}$$

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

$$p_{i_k} + (1 - p_{i_k})(1 - p^*_{i_{k+1}}) = p^*_{i_k}, \tag{9}$$

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 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

$$\frac{p^*_{i_k}}{p_{i_k}}.$$

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

$$p_{i_k} \cdot p^*_{i_{k+1}} = p^*_{i_k}, \tag{10}$$

or

$$p^*_{i_{k+1}} = \frac{p^*_{i_k}}{p_{i_k}}.$$

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

$$p_{i_k}(1 - p^*_{i_{k+1}}) = p^*_{i_k}, \tag{11}$$

or

$$p^*_{i_{k+1}} = 1 - \frac{p^*_{i_k}}{p_{i_k}}.$$

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 $x_1, \ldots, x_n$ 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

$$p^*_{i_{n+1}} = \frac{p^*_{i_n} - p_{i_n}}{1 - p_{i_n}}. \tag{12}$$

When $p^*_{i_n} \leq p_{i_n}$, we choose an AND gate. The ideal probability value for the input x_{n+1} is

$$p^*_{i_{n+1}} = \frac{p^*_{i_n}}{p_{i_n}}. \tag{13}$$

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 $C_1, \ldots, C_{n+1}$ 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 $k-1$ fanin-two gates. Thus, the final solution contains at most n 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 our greedy algorithm is bounded.

Theorem 4
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: See Appendix A. □

V. 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,

$$h(q) = \min_{\forall i, z_i \in \{0,1\}} \left| \sum_{i=0}^{2^n-1} z_i r_i - q \right|. \tag{14}$$

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,

$$H(S) = \int_0^1 h(q)\, \mathrm{d}q. \tag{15}$$

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 4
Consider the truth table shown in Table III. 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 III: 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 $b_1 \le b_2 \le \cdots \le b_N$. 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 5
A lower bound for $H(S)$ is $\dfrac{1}{4(N-1)}$. □

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,

$$\begin{aligned} H(S) &= \int_0^1 h(q)\,\mathrm{d}q \\ &= \sum_{i=1}^{N-1} \left(\int_{b_i}^{\frac{b_i+b_{i+1}}{2}} (q - b_i)\,\mathrm{d}q + \int_{\frac{b_i+b_{i+1}}{2}}^{b_{i+1}} (b_{i+1} - q)\,\mathrm{d}q \right) \\ &= \frac{1}{4} \sum_{i=1}^{N-1} (b_{i+1} - b_i)^2. \end{aligned} \tag{16}$$

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

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

□

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

$$S = \{p | p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \ldots, n-1\}. \tag{17}$$

Lemma 1
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 $(0 \le i \le 2^n - 1)$ is $\dfrac{2^i}{2^{2^n}-1}$. □

Proof: See Appendix B. □

Based on Lemma 1, we will show that the set S in Equation (17) achieves the lower bound for $H(S)$.

Theorem 6
The set $S = \{p | p = \dfrac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \ldots, n-1\}$ achieves the lower bound $\dfrac{1}{4(N-1)}$ for $H(S)$. □

Proof: By Lemma 1, for the given set S, the probability of the i-th input combination $(0 \le i \le 2^n - 1)$ is $\dfrac{2^i}{2^{2^n}-1}$. 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 4 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 (16), we have $H(S) = \dfrac{1}{4(N-1)}$. □

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 $\dfrac{i}{2^{2^n}-1}$. Suppose the closest fraction is $\dfrac{g(q)}{2^{2^n}-1}$. 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.

VI. Conclusions and Future Work

In this work, we considered the problem of transforming a set of input probabilities into a target probability with combinational logic. The assumption that we make is that the input probabilities are exact and independent. For example, in synthesizing decimal output probabilities, we use multiple independent copies of the exact input probabilities 0.4 and 0.5. Of course, if we use physical sources to generate the input probabilities, there likely will be fluctuations. Also, the probabilistic inputs will likely be correlated. A future direction of research is how to design circuits that behave robustly in spite of these realities.

In addition to the three scenarios that we presented, there exists a fourth one that we have not considered: one in which the source probabilities are specified and can be duplicated. In this scenario, we would not expect to generate the target probability exactly. Thus, the problem is how to synthesize an area or delay optimal circuit whose output probability is a close approximation to the target value. We will address this problem in future work.

VII. Acknowledgments

The authors thank Kia Bazargan and David Lilja for their contributions. They were co-authors on a preliminary version of this paper [16].

References

[1] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.

[2] 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 (to appear)*, 2010.

[3] 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*, 2005, pp. 535–541.

[4] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem, "Probabilistic system-on-a-chip architecture," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–28, 2007.

[5] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. 24, no. 6, pp. 668–670, 1975.

[6] J. Savir, G. Ditlow, and P. H. Bardell, "Random pattern testability," *IEEE Transactions on Computers*, vol. 33, pp. 79–90, 1984.

[7] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *Design Automation Conference*, 2001, pp. 661–666.

[8] R. Marculescu, D. Marculescu, and M. Pedram, "Logic level power estimation considering spatiotemporal correlations," in *International Conference on Computer-Aided Design*, 1994, pp. 294–299.

[9] A. Gill, "Synthesis of probability transformers," *Journal of the Franklin Institute*, vol. 274, no. 1, pp. 1–19, 1962.

[10] ——, "On a weight distribution problem, with application to the design of stochastic generators," *Journal of the ACM*, vol. 10, no. 1, pp. 110–121, 1963.

[11] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Transactions on Information Theory*, vol. 40, no. 3, pp. 716–720, 1994.

[12] D. Wilhelm and J. Bruck, "Stochastic switching circuit synthesis," in *International Symposium on Information Theory*, 2008, pp. 1388–1392.

[13] C. E. Shannon, "The synthesis of two terminal switching circuits," *Bell System Technical Journal*, vol. 28, pp. 59–98, 1949.

[14] H. Zhou and J. Bruck, "On the expressibility of stochastic switching circuits," in *International Symposium on Information Theory*, 2009, pp. 2061–2065.

[15] A. Mishchenko *et al.*, "ABC: A system for sequential synthesis and verification," 2007. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[16] W. Qian, M. D. Riedel, K. Barzagan, and D. Lilja, "The synthesis of combinational logic to generate probabilities," in *International Conference on Computer-Aided Design*, 2009, pp. 367–374.

Appendix A

Theorem 4
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 $C_1, \ldots, C_{n+1}$ 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 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,

$$|p_{i_{n+1}} - p^*_{i_{n+1}}| \leq \frac{1}{2}(|p^*_{i_{n+1}}| + |1 - p^*_{i_{n+1}}|).$$

Our greedy algorithm essures that $0 \leq p^*_{i_{n+1}} \leq 1$. Thus, we further have

$$|p_{i_{n+1}} - p^*_{i_{n+1}}| \leq \frac{1}{2}. \tag{18}$$

Next, we will show by induction that for all $1 \leq k \leq n$, we have

$$|w_k - p^*_{i_{n+1-k}}| \leq \frac{1}{2} \prod_{j=1}^{k} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \tag{19}$$

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 (12), 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 (18), we have

$$\begin{aligned} |w_1 - p^*_{i_n}| &\leq \frac{1}{2}(1 - p_{i_n}) \leq \frac{1}{2} \max\{p_{i_n}, 1 - p_{i_n}\} \\ &= \frac{1}{2} \prod_{j=1}^{1} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \end{aligned} \tag{20}$$

Similarly, if the first gate is an AND gate, we can also get Equation (20). Thus, the statement holds for the base case.

Inductive step: Assume that the statement holds for some $1 \leq k \leq n - 1$. 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,

$$\begin{aligned} |w_{k+1} - p^*_{i_{n-k}}| &= (1 - p_{i_{n-k}})|w_k - p^*_{i_{n+1-k}}| \\ &\leq \max\{p_{i_{n-k}}, 1 - p_{i_{n-k}}\}|w_k - p^*_{i_{n+1-k}}|. \end{aligned} \tag{21}$$

Based on the induction hypothesis, we have

$$|w_k - p^*_{i_{n+1-k}}| \leq \frac{1}{2}\prod_{j=1}^{k} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \tag{22}$$

Combining Equations (21) and (22), we have

$$|w_{k+1} - p^*_{i_{n-k}}| \leq \frac{1}{2}\prod_{j=1}^{k+1} \max\{p_{i_{n+1-j}}, 1 - p_{i_{n+1-j}}\}. \tag{23}$$

In the other three cases, we can similarly derive Equation (23). Thus, the statement holds for $k+1$. This completes the induction proof.

Note that $\{p_{i_1}, \ldots, p_{i_n}\} = \{p_1, \ldots, p_n\}$. Thus, when $k = n$, Equation (19) can be written as

$$|w_n - p^*_{i_1}| \leq \frac{1}{2}\prod_{j=1}^{n} \max\{p_j, 1 - p_j\}.$$

Based on our algorithm, the final output is either the direct output of the n-th gate or the inverted output of the n-th gate. In either case, we have

$$|w - q| = |w_n - p^*_{i_1}| \leq \frac{1}{2}\prod_{j=1}^{n} \max\{p_j, 1 - p_j\}.$$

□

APPENDIX B

Lemma 1

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 $\frac{2^i}{2^{2^n} - 1}$. □

Proof: We prove the lemma by induction on n.

Base case: When $n = 1$, by assumption, $P(x_1 = 1) = \frac{2}{3}$. 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 $P(x_n = 1) = \frac{2^{2^{n-1}}}{2^{2^{n-1}} + 1}$.

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,

$$\begin{aligned} p_{i,n} &= P(x_n = 0) \cdot p_{i,n-1} = \frac{1}{2^{2^{n-1}} + 1} \cdot \frac{2^i}{2^{2^{n-1}} - 1} \\ &= \frac{2^i}{2^{2^n} - 1}. \end{aligned} \tag{24}$$

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,

$$\begin{aligned} p_{i,n} &= P(x_n = 1) \cdot p_{i-2^{n-1},n-1} = \frac{2^{2^{n-1}}}{2^{2^{n-1}} + 1} \cdot \frac{2^{i-2^{n-1}}}{2^{2^{n-1}} - 1} \\ &= \frac{2^i}{2^{2^n} - 1}. \end{aligned} \tag{25}$$

Combining Equation (24) and (25), the statement holds for n. Thus, the statement in the lemma holds for all n. □