Homeworks # 1

due Feb. 26, 2016 (11:59pm Hawaii Time)

1. **Synthesizing Stochastic Logic**

   In class, we discussed the paradigm of logical computation on stochastic bit streams It 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.

   In this paradigm, since we are mapping probabilities to probabilities, we can only implement functions that map the unit interval $[0, 1]$ onto the unit interval $[0, 1]$. Based on the constructs for multiplication and scaled addition shown in Figures 1 and 2, we can readily implement polynomial functions of a specific form, namely polynomials with non-negative coefficients that sum up to a value no more than one:

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

   where, for all $i = 0, \ldots, n$, $a_i \geq 0$ and $\sum_{i=0}^{n} a_i \leq 1$.

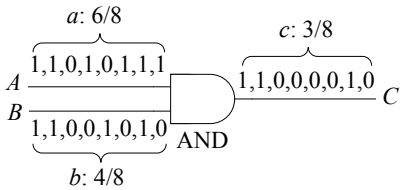


Figure 1: Multiplication on stochastic bit streams with an AND gate. Here the inputs are 6/8 and 4/8. The output is $6/8 \times 4/8 = 3/8$, as expected.

   For example, suppose that we want to implement the polynomial $g(t) = 0.3t^2 + 0.3t + 0.2$ through logical computation on stochastic bit streams.

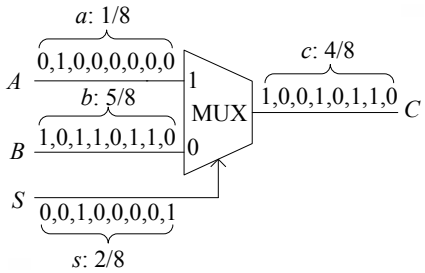


Figure 2: Scaled addition on stochastic bit streams, with a multiplexer (MUX). Here the inputs are $1/8, 5/8$, and $2/8$. The output is $2/8 \times 1/8 + (1 - 2/8) \times 5/8 = 4/8$, as expected.

We first decompose it in terms of multiplications of the form $a \cdot b$ and scaled additions of the form $sa + (1 - s)b$, where s is a constant:

$$g(t) = 0.8(0.75(0.5t^2 + 0.5t) + 0.25 \cdot 1).$$

Then, we reconstruct it with the following sequence of multiplications and scaled additions:

$$\begin{aligned} w_1 &= t \cdot t, \\ w_2 &= 0.5w_1 + (1 - 0.5)t, \\ w_3 &= 0.75w_2 + (1 - 0.75) \cdot 1, \\ w_4 &= 0.8 \cdot w_3. \end{aligned}$$

The circuit implementing this sequence of operations is shown in Figure 3. In the figure, the inputs are labeled with the probabilities of the bits of the corresponding stochastic streams. Some of the inputs have fixed probabilities and the others have variable probabilities t. Note that the different lines with the input t are each fed with *independent* stochastic streams with bits that have probability t.

What if the target function is a polynomial that is not decomposable this way? Suppose that it maps the unit interval onto the unit interval but it has some coefficients less than zero or some greater than one. For instance, consider the polynomial $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$. It is not apparent how to construct a network of stochastic multipliers and adders to implement it.

In class, we discsused a general method for synthesizing arbitrary univariate polynomial functions on stochastic bit streams. A necessary

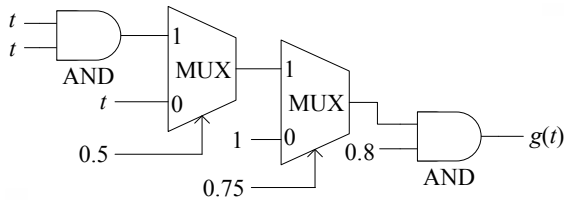


Figure 3: Computation on stochastic bit streams implementing the polynomial $g(t) = 0.3t^2 + 0.3t + 0.2$.

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 illustrate the basic steps of our synthesis method with the example of $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$.

(a) Convert the polynomial into a Bernstein polynomial with all coefficients in the unit interval:

$$g(t) = \frac{3}{4} \cdot [(1-t)^2] + \frac{1}{4} \cdot [2t(1-t)] + \frac{1}{2} \cdot [t^2].$$

Note that the coefficients of the Bernstein polynomial are $\frac{3}{4}$, $\frac{1}{4}$ and $\frac{1}{2}$, all of which are in the unit interval.

(b) Implement the Bernstein polynomial with a multiplexing circuit, as shown in Figure 4. The block labeled "+" counts the number of ones among its two inputs; this is either 0, 1, or 2. The multiplexer selects one of its three inputs as its output according to this value. Note that the inputs with probability t are each fed with *independent* stochastic streams with bits that have probability t.

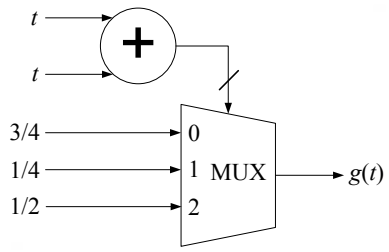


Figure 4: A generalized multiplexing circuit implementing the polynomial $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$.

Problem

Implement the following polynomial

$$\frac{1}{3}\left(1 - t + t^2 - t^3 + t^4 - t^5\right)$$

this way.

Demonstrate how the circuit works on the following input values:

- $X = 0$
- $X = 0.25$
- $X = 0.5$
- $X = 0.75$
- $X = 1$

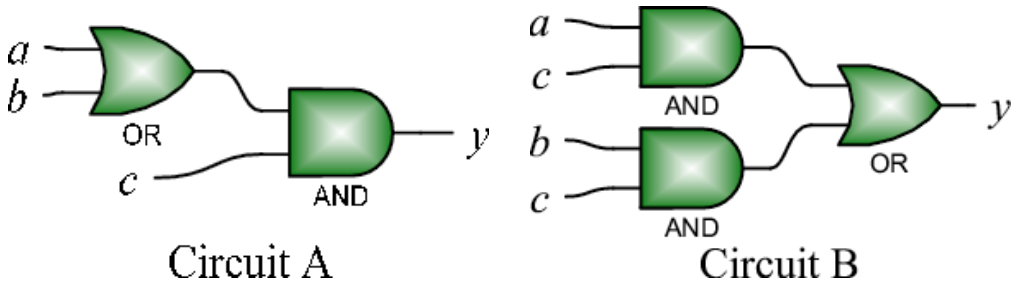
In each case, track probabilities through the circuit that you drew.

2. **Input-Dependent Probability of Failure**

   A pervasive problem for digital nano-circuitry is coping with defects and failures. For most nanoscale processes, the devices and components are inherently unreliable. They may exhibit significant variations in their operating parameters. Worse, they may fail intermittently or permanently.

   In this problem, you will investigate analysis techniques for digital circuits characterized by logic gates that compute probabilistically: with some probability, each gate produces the incorrect result; with one minus this probability, it produces the correct result.

   Consider the two circuits:



   Note that both circuits implement the same Boolean function:

$$(a+b)c$$

   Suppose that each gate produces the incorrect result (i.e., the complement of the correct Boolean value) with probability ϵ. The probability that the circuits produce the incorrect results are:

Failure Probability

$$\begin{cases} P_A = \varepsilon - (2\varepsilon^2 - \varepsilon)c \\ P_B = 3\varepsilon - 5\varepsilon^2 + 2\varepsilon^3 - (\varepsilon - 2\varepsilon^2)(a+b)c + (2\varepsilon^2 - 4\varepsilon^3)abc \end{cases}$$

   The failure probability depends on the specific input combination. For some input combinations, Circuit A has a lower probability of failure; for others Circuit B does. The following table gives the failure probabilities for a specific value of ϵ.

$$\varepsilon = 0.05$$

a	b	c	P_A	P_B	Better
0	0	0	0.050	0.138	*A*
0	0	1	0.095	0.138	*A*
0	1	0	0.050	0.138	*A*
0	1	1	0.095	0.093	*B*
1	0	0	0.050	0.138	*A*
1	0	1	0.095	0.093	*B*
1	1	0	0.050	0.138	*A*
1	1	1	0.095	0.052	*B*

Problem

For the circuit in Figure 5, suppose that each gate produces the incorrect result (i.e., the complement of the correct Boolean value) with the same probability ϵ. For each input assignment of x, y and z compute the probability of obtaining an incorrect result at the outputs c and s.

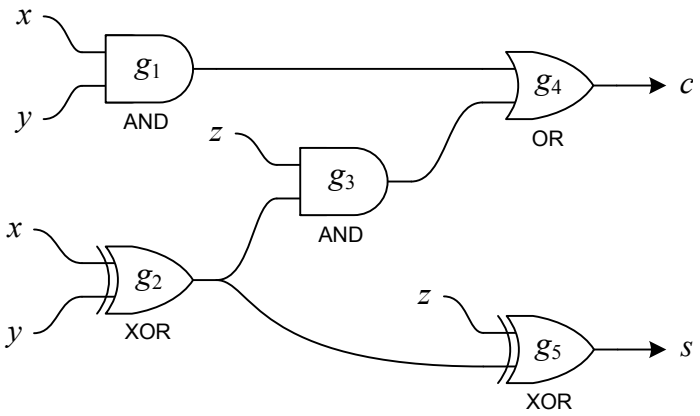


Figure 5: A full adder.

3. **Transforming Probabilities**

   A premise for stochastic computing is the availability of bit streams with the requisite probabilities. Such streams can either be generated from physical random sources or with pseudo-random constructs such as linear feedback shift registers.

   If the system exploits a physical mechanism, the random source may be cheap but the constant value may be expensive to implement. In schemes that we have looked at, each constant value 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.

   This problem presents a synthesis strategy to mitigate this issue: we will discuss 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.

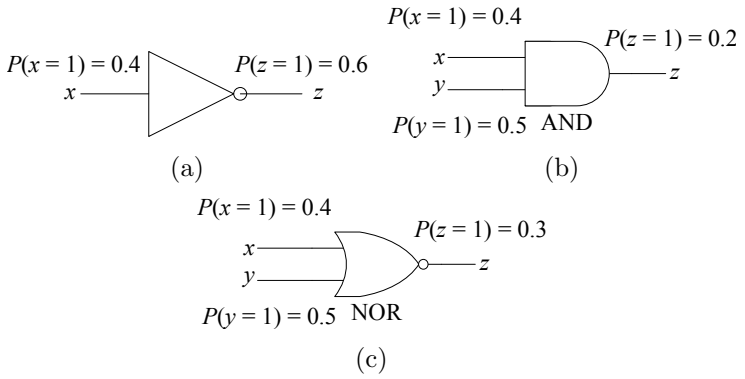


Figure 6: An illustration of transforming a set of source probabilities into new probabilities with logic gates. (a): An inverter implementing $p_z = 1 - p_x$. (b): An AND gate implementing $p_z = p_x \cdot p_y$. (c): A NOR gate implementing $p_z = (1 - p_x) \cdot (1 - p_y)$.

   (a) Suppose that we have a set of source probabilities $S = \{0.4, 0.5\}$. As illustrated in Figure 6, we can transform this set into new probabilities:

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

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

ii. 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

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

iii. 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 $\{0.4, 0.5\}$. □

In fact, all probabilities can be generated from $\{0.4, 0.5\}$. Figure 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.

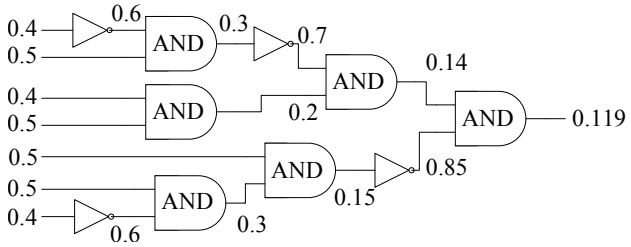


Figure 7: A circuit synthesized by our algorithm to realize the decimal output probability 0.119 from the input probabilities 0.4 and 0.5.

Problem

Implement the following probabilities, starting with $\{0.4, 0.5\}$, using AND and NOT gates.

i. 0.6555

ii. 0.6666

iii. 0.1111

(b) Now suppose that we have a set of source probabilities $S = \{0.5\}$. Suppose that the target probability is given as a probability expressed in binary.

Problem

Implement the following probabilities with logic.

- $\frac{1}{4} = 0.010_2$.
- $\frac{3}{4} = 0.110_2$.
- $\frac{5}{16} = 0.0101_2$.
- $\frac{11}{16} = 0.1011_2$.
- $\frac{27}{64} = 0.100101_2$.

(c) **Problem**

Describe a general method for implementing probabilities, given in binary, starting from $\frac{1}{2}$.