# 27

# Binary Arithmetic:

# An Application to Programming

In the previous section we looked at the binomial distribution. The binomial distribution is essentially the mathematics of repeatedly flipping a coin (and there is no requirement that the coin be unbiased). These coin flips are known as *Bernoulli trials*. Anytime you repeat an experiment with two possible outcomes and with your experiments independent of each other, you are performing Bernoulli trials. Frequently we desire to simulate such experiments on the computer.[1]

| | | |
|---|---|---|
| 10 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | + |

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | × |

**Formula 1** Binary Addition and Multiplication Tables

It is fitting and logical that since a Bernoulli trial has two outcomes that binary arithmetic can be quite useful in the simulation of repeated Bernoulli trials. I assume here that you have seen binary arithmetic before. I am repeating the essentials merely for the purpose of review.

[1]One application of simulated Bernoulli experiments is to bootstrap statistics. Given a binomial experiment with a lot of trials one might statistically analyze it by simulation rather than by approximating the binomial distribution with the normal distribution. Until recently the cost of computing made this impractical.

| Decimal Number | Binary Number |
|---|---|
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| 16 | 10000 |
| 17 | 10001 |
| 18 | 10010 |
| 19 | 10011 |
| 20 | 10100 |

# The Application of Binary Arithmetic to Bernoulli Simulation

Suppose that we want to simulate tosses of a coin until three heads occur. At any time we are only interested in the last three throws. There are $2^3 = 8$ possible outcomes to three throws. These are:

- TTT
- TTH
- THT
- THH
- HTT
- HTH
- HHT
- HHH

We will refer to the last three throws as the *state* of the program. Notice, that we can number these eight states in any way that we like. However, it makes good sense to use a method of enumeration that has an underlying logic. The most reasonable such scheme is to use binary arithmetic. Consider each H as a 1 and each T as a 0 (it is equally valid to do the opposite and replace each H by a 0 and each T by a 1). Then the above list gives the binary representation of the numbers 0 through 7 in the usual counting order.

The computer program we are considering is going to model the flips of our coin until we get three heads. The logic of the program is as follows:

1 If the last three throws are all heads Then stop.
2 Flip coin.
3 Return to first line.

To flesh out this program we need to use the representation of the state. The program starts to look like this:

1 State ← 0.

2 Flip coin and update state.

3 **If** State = 7 **Then** Stop.

4 *Goto* line 2.

Our question is how do we update the state? The following algorithm is based on binary arithmetic and is very efficient:

1 State ← 0.

2 State ← State∘2.

3 State ← State mod 8.

4 Toss coin. **If** outcome is heads **Then** State ← State + 1.

5 If State = 7 Then stop

6 **Goto** line 2.

## Why does this work?

We start with State = 0 because we have thrown no heads so far. Consider the binary representation of the state. When we multiply by 2, we are moving all the digits one place to the left and we are adding a 0 on the right end. For example if State = 5, 5 corresponds to 101 in binary notation. 5∘2 becomes 101∘10 which is 1010. When we toss the coin we can change that last digit to a 1 (heads) simply by adding a 1. Otherwise we keep that last digit at 0 which corresponds to tails. However, we are only interested in the last three digits. If we have four digits and the first digit on the left is a 0 then we can ignore it. If the first digit on the left is a 1, that 1 is in the eight's place. We could get rid of it simply by subtracting 8 from the current state number or we could replace the state by the state mod 8. (You can convince yourself that this all works simply by doing an example). The advantage of State ← State mod 8, is in the case the first digit on the left is a 0. That is, if State is already within the range 0 through 7, replacing State by State mod 8 changes nothing. Normally when we simulate a binomial experiment there

are other aspects to the program. Frequently, we want to count the number of flips until we are finished. These other elements are usually easy to add to the algorithm.

☐ **Exercise 1** Rewrite the above algorithm without the *go to* statement.

☐ **Exercise 2** Imagine that you do a sequence of flips getting, from left to right, 10010110111. What are the corresponding states?

## How to Use Decimal Arithmetic to Represent Binary Numbers

Notice that our states are represented in decimal notation but the algorithm is based upon their corresponding binary representations. Another interesting approach to the problem is to use decimal arithmetic but to make it look like the binary notation. That is, to represent the state corresponding to HTT we would use the **decimal** number 100. The algorithm then looks as follows:

1 State ← 0.

2 State ← State∘10.

3 State ← State mod 1000.

4 Toss coin. **If** outcome is heads **Then** State ← State + 1.

5 **If** State = 111 **Then** stop.

6 **Goto** line 2.

☐ **Exercise 3** Imagine that you do a sequence of flips getting, from left to right, 10010110111. What are the corresponding states in terms of this latest algorithm?

1. State ← 0.
   Repeat until State = 7.
   State ← State∘2.
   State ← State mod 8.
   Toss coin.
   If outcome is heads Then State ← State + 1.

2. If the throws are 10010110111 (left to right). Then your successive states will be: 0, 1, 2, 4, 1, 2, 5, 3, 6, 5, 3, 7.

3. If the throws are 10010110111 (left to right). Then your successive states will be: 000, 001, 010, 100, 001, 010, 101, 011, 110, 101, 011, 111.