Contents

Foreword

The title of this project is called "Generating Distributions", and this being a project in computer science, our means of generating distributions would of course be computer programs. To write programs that are capable of generating distributions we need a programming language that contains probabilistic choice as a construct. We shall investigate several small examples of programs written in the Probabilistic Guarded Command Language ($pGCL$), a probabilistic extension of the Guarded Command Language of Dijkstra[1]. One of the goals of this project is to find small programs that generate interesting probability distributions. Another important goal is to find a theory that allows us to reason rigourously about the distributions generated with style and ease close to what we have in the wp-semantics for standard GCL programs.

We shall make use of the expectation-transformer model of McIver and Morgan[12, 13] as our main tool of analysis, complemented by the Probabilistic Dynamic Logic (**Pr**DL) of Feldman and Harel[3] when we analyse programs with loops.

This report is divided into five chapters. Chapter 1 contains a brief introduction of $pGCL$ and the expectation-transformer model and several small examples. Chapter 2 is a discussion of two different approaches for reasoning about loops, where we introduce the Probabilistic Dynamic Logic. Chapter 3 contains two more complicated examples and we show how they could be analysed with the tools developed in chapter 2. Chapter 4 is quite different from the rest of the report. It is an attempt to study a family of algorithms for simulating an unbiased source of randomness using a biased source. Chapter 5 is a conclusion of the whole project.

I must thank my supervisor Dr. J.W. Sanders for sacrificing many hours for discussing the project with me. Those discussions were enlightening and help me realize the inadequecies of many of my ideas and also help me to solve many problems I faced. He also tirelessly read the many different drafts of this report, and reminded me to finish the project on time. Of course he is in no way responsible for any mistakes or problems in this report.

I hope you would enjoy reading it.

Chapter 1

The Probabilistic Guarded Command Language and Expectation Transformers

1.1 Introduction

In this chapter we describe the programming language $pGCL$, a probabilistic extension of the Guarded Command Language of Dijkstra[1]. It is the main language of this report and all the programs are going to be written in it. Readers who are familiar with the Guarded Command Language should find it easy to understand. For the time being we consider only loop-free programs. The loop rules, and programs with loops will be discussed in the next chapter.

1.2 Basic syntax and semantics of $pGCL$, and some notes on notations

We follow the syntax and semantics presented in McIver and Morgan[12, 13], except that we are not going to make much use of non-deterministic choice in the programs in this report. So our $pGCL$ is very similiar to that presented in Jones[9], except that we are going to make use of the syntax and notations of McIver and Morgan.

1.2.1 Syntax

The syntax of $pGCL$ is as follows:

$$\begin{aligned} prog \ := \ & \mathbf{abort} \mid \mathbf{skip} \mid x := E \mid prog; prog \mid \\ & prog \ {}_p\oplus \ prog \mid prog \sqcap prog \mid \\ & (\mu X \cdot C) \end{aligned} \tag{1.1}$$

The syntax of $pGCL$ is basically the same as GCL, with the addition of a probabilistic choice operator ${}_p\oplus$.

1.2.2 The probabilistic choice operator

First of all we introduce the probabilistic choice operator ${}_p\oplus$.

$$x := 0 \ {}_p\oplus \ x := 1 \tag{1.2}$$

We interprete the program as "choose the left branch $x := 0$ with probability p and the right branch $x := 1$ with probability $1 - p$". We sometimes use

$$x := 1 \oplus_p x := 0 \tag{1.3}$$

as another way of writing the above program. Yet another variation of the above program is:

$$x := 0 \ {}_p\oplus \ 1 \tag{1.4}$$

1.2.3 Boolean embedding

For any predicate P, we write $[P]$ as the characteristic function of P, i.e., $[P] = 1$ when P is true and 0 otherwise. For example, $[x = 0]$ is 1 if $x = 0$ and 0 for other values of x.
Below are some rules for the manipulation of characteristic functions for predicates:

$$\begin{aligned} [\mathbf{true}] &= 1 \ \text{ when it is not enclosed by wp or program free} \\ [\mathbf{false}] &= 0 \\ [P \wedge Q] &= [P][Q] \\ [P \vee Q] &= [P]\mathbf{max}[Q] \\ [P] + [\neg P] &= [\mathbf{true}] \end{aligned}$$

1.2.4 Probabilistic wp semantics of $pGCL$

Below are some of the transformation rules for the wp semantics of $pGCL$, with many of them generalized from standard GCL:

$wp.\mathbf{abort}.postE := 0$
$wp.\mathbf{skip}.postE := postE$
$wp.(x := expr).postE := postE(x \mapsto expr)$
$wp.(prog; prog').postE := wp.prog.(wp.prog'.postE)$
$wp.(prog \sqcap prog').postE := wp.prog.postE \ \mathbf{min} \ wp.prog' postE$
$wp.(prog \oplus_p prog').postE := (1-p) \cdot wp.prog.postE + p \cdot wp.prog'.postE$
$wp.(\mathbf{if} \ B \ \mathbf{then} \ prog \ \mathbf{else} \ prog').postE := [B] \cdot wp.prog.postE + [\neg B] \cdot wp.prog'.postE$

Figure 1.1: wp semantics of $pGCL$

1.2.5 Syntatic sugar

We are going to use extensively the operation of choosing a number randomly from the interval $[1..N]$ (i.e. each with probability $1/N$). It could be written using the probabilistic choice operator ${}_p\oplus$ as follows:

$$x := 1_{\frac{1}{N}}\oplus(2_{\frac{1}{N-1}}\oplus(\cdots{}_{\frac{1}{3}}\oplus(N-1_{\frac{1}{2}}\oplus N))\cdots)) \tag{1.5}$$

It will be convenient to have a shorthand for this and so we define

$$x :\in [1..N] \tag{1.6}$$

to be equivalent to expression (1.5).
As a first exercise in $pGCL$ we try to prove the following result using the newly introduced tranformation rules:
Lemma: Let N be a positive integer, P be a predicate. Then

$$wp.(x :\in [1..N]).[P] = \frac{1}{N}\sum_{i=1}^{N}[P(x := i)] \tag{1.7}$$

where i is not free in the predicate P. This result will be used frequently in the remainder of this report and is proved as follows:

$$\begin{aligned}
& wp.(x :\in [1..N]).[P] \\
= \quad & \{\text{definition of } :\in [1..N]\} \\
& wp.(x := 1_{\frac{1}{N}} \oplus (2_{\frac{1}{N-1}} \oplus (\cdots{}_{\frac{1}{3}} \oplus (N-1_{\frac{1}{2}} \oplus N))\cdots)).[P] \\
= \quad & \{\text{definition of probabilistic choice}\} \\
& \frac{1}{N}wp.(x := 1).[P] + \frac{N-1}{N}wp.(x := 2_{\frac{1}{N-1}} \oplus (\cdots{}_{\frac{1}{3}} \oplus (N-1_{\frac{1}{2}} \oplus N))\cdots)).[P] \\
= \quad & \{\text{assignment; definition of probabilistic choice}\} \\
& \frac{1}{N}[P(x := 1)] + \frac{N-1}{N}\frac{1}{N-1}wp.(x := 2).[P] \\
& + \frac{N-1}{N}\frac{N-2}{N-1}wp.(x := 3_{\frac{1}{N-2}} \oplus (\cdots{}_{\frac{1}{3}} \oplus (N-1_{\frac{1}{2}} \oplus N))\cdots)).[P] \\
= \quad & \cdots
\end{aligned}$$

$$
\begin{aligned}
= & \quad \{\text{continue unfolding}\} \\
& \frac{1}{N}[P(x := 1)] + \frac{N-1}{N}\frac{1}{N-1}[P(x := 2)] + \cdots + \frac{N-1}{N}\frac{N-2}{N-1}\cdots\frac{2}{3}\frac{1}{2}[P(x := N)] \\
= & \frac{1}{N}\sum_{i=1}^{N}[P(x := i)], \text{ for a variable } i \text{ that is not free in } P
\end{aligned}
\tag{1.8}
$$

In this chapter we use program variables such as x, y to denote random variables directly instead of following common practice of using capital letters X, Y as in probability theory. In our programs it should be clear from the context whether a certain program variable is a random variable or not.

1.3 A first example

Consider the following program:

```
{Program 1.1}
|[var x;
   x := 0;
   x := x ⊕_p x + 1;
   x := x ⊕_p x + 1;
   x := x ⊕_p x + 1.
]|
```

What is the distribution of x?

It is not difficult to see that x is a sum of 3 independent identically distributed Bernoulli random variables with parameter p. So x is Binomial$(3, p)$ in the notation of elementary probability theory.

The distribution can be calculated as follows, in a step by step manner starting from the last assignment.
For $0 \leq r \leq 3$,

$$
\begin{aligned}
& wp.(x := x \oplus_p x + 1).[x = r] \\
= & \quad \{\text{definition of probabilistic choice}\} \\
& p[x + 1 = r] + (1 - p)[x = r] \\
= & \quad \{\text{arithmetic}\} \\
& p[x = r - 1] + (1 - p)[x = r]
\end{aligned}
\tag{1.9}
$$

Then we calculate the effect of the next assignment:

$$
\begin{aligned}
& wp.\big(x := x \oplus_p x+1\big).\big(p[x=r-1]+(1-p)[x=r]\big) \\
= & \qquad \{\text{definition of probabilistic choice}\} \\
& p^2[x+1=r-1]+p(1-p)[x+1=r]+p(1-p)[x=r-1]+(1-p)^2[x=r] \\
= & \qquad \{\text{arithmetic}\} \\
& p^2[x=r-2]+2p(1-p)[x=r-1]+(1-p)^2[x=r]
\end{aligned} \tag{1.10}
$$

Lastly for the first probabilistic choice:

$$
\begin{aligned}
& wp.\big(x := x \oplus_p x+1\big).\big(p^2[x=r-2]+2p(1-p)[x=r-1]+(1-p)^2[x=r]\big) \\
= & \qquad \{\text{definition of probabilistic choice}\} \\
& p^3[x+1=r-2]+2p^2(1-p)[x+1=r-1]+p(1-p)^2[x+1=r] \\
& +p^2(1-p)[x=r-2]+2p(1-p)^2[x=r-1]+(1-p)^3[x=r] \\
= & \qquad \{\text{arithmetic}\} \\
& p^3[x=r-3]+3p^2(1-p)[x=r-2]+3p(1-p)^2[x=r-1]+(1-p)^3[x=r]
\end{aligned} \tag{1.11}
$$

As x is initialised to be 0,

$$
\begin{aligned}
& wp.\big(x := 0\big).\big(p^3[x=r-3]+3p^2(1-p)[x=r-2] \\
& +3p(1-p)^2[x=r-1]+(1-p)^3[x=r]\big) \\
= & \qquad \{\text{assignment}\} \\
& p^3[0=r-3]+3p^2(1-p)[0=r-2]+3p(1-p)^2[0=r-1]+(1-p)^3[0=r] \\
= & \qquad \{\text{arithmetic}\} \\
& p^3[r=3]+3p^2(1-p)[r=2]+3p(1-p)^2[r=1]+(1-p)^3[r=0] \\
= & \qquad \{\text{arithmetic}\} \\
& \sum_{k=0}^{3}\binom{3}{k}p^k(1-p)^{n-k}[r=k]
\end{aligned} \tag{1.12}
$$

Thus the distribution of x is indeed Binomial$(3,p)$.
This program is rather inelegant and we shall look at another program that generates the binomial distribtutions in the next chapter.

1.4 A second example, a monotone distribtution

Now let's look at a less obvious example:

{Program 1.2}
var i;
|[**var** n;
    $n :\in [1..N]$;
    $i :\in [1..n]$;
]| **rav** n

What is the distribution of i after hiding n?

We are seeking the marginal pdf (or more precisely, pmf, probability mass function) $f_i(r)$ of i from the joint distribution of i and n.
If $1 \leq r \leq N$,

$$
\begin{aligned}
& f_i(r) \\
= \; & wp.(n :\in [1..N]; i :\in [1..n])[i = r] \\
= \; & \qquad \{\text{Rule for uniform choice (1.7)}\} \\
& wp.(n :\in [1..N])(\frac{1}{n}\sum_{x=1}^{n}[x = r]) \\
= \; & \qquad \{\text{Rule for uniform choice (1.7)}\} \\
& \frac{1}{N}\sum_{y=1}^{N}\frac{1}{y}\sum_{x=1}^{y}[x = r] \\
= \; & \qquad \{\text{arithmetic}\} \\
& \frac{1}{N}\sum_{y=1}^{N}\frac{1}{y}[1 \leq r \leq y] \\
= \; & \qquad \{\text{arithmetic}\} \\
& \frac{1}{N}\sum_{y=r}^{N}\frac{1}{y}[1 \leq r \leq N] \\
= \; & \qquad \{\text{arithmeitc}\} \\
& \frac{1}{N}(H_N - H_{r-1})[1 \leq r \leq N]
\end{aligned}
\tag{1.13}
$$

where H_k is the kth Harmonic number$(= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k})$, and we define H_0 to be 0. The distribution is monotonic decreasing in r.

The advantage of using wp semantics to calculate distributions is that one does not have to worry about whether the distribution is well-defined at a particular execution point of the program, for example, when a new variable is declared but not yet initialised. Such a calculation in a relation model would be difficult because it is clumsy to specify completely the state space during every step in the calculation. The wp function gives the probability distribution function of the distribution generated after the program and the post-expectation are supplied. Moreover, the calculation is almost the same if we want to calculate the joint distribution instead: we only need to change the post-expectation from $[i = r]$ to $[i = r \wedge n = s]$ to obtain the joint probability distribution function $f_{i,n}(r, s)$:

$$
\begin{array}{cl}
 & wp.(n :\in [1..N]; i :\in [1..n])[i = r \wedge n = s] \\
= & \qquad \{\text{Rule for uniform choice (1.7)}\} \\
 & wp.(n :\in [1..N])(\frac{1}{n}\sum_{x=1}^{n}[x = r \wedge n = s]) \\
= & \qquad \{\text{Rule for uniform choice (1.7)}\} \\
 & \frac{1}{N}\sum_{y=1}^{N}\frac{1}{y}\sum_{x=1}^{y}[x = r \wedge y = s] \\
= & \qquad \{\text{Property of characteristic functions}\} \\
 & \frac{1}{N}\sum_{y=1}^{N}\frac{1}{y}\sum_{x=1}^{y}[x = r][y = s] \\
= & \qquad \{\text{arithmetic}\} \\
 & \frac{1}{N}\sum_{y=1}^{N}\frac{1}{y}[y = s]\sum_{x=1}^{y}[x = r] \\
= & \qquad \{\text{arithmetic}\} \\
 & \frac{1}{N}\frac{1}{s}[1 \le s \le N]\sum_{x=1}^{s}[x = r] \\
= & \qquad \{\text{arithmetic}\} \\
 & \frac{1}{Ns}[1 \le s \le N][1 \le r \le s] \\
= & \qquad \{\text{Property of characteristic functions}\} \\
 & \frac{1}{Ns}[1 \le r \le s \le N]
\end{array}
\tag{1.14}
$$

i.e., $f_{i,n}(r, s) = 1/Ns$ when $1 \le r \le s \le N$, and 0 for all other pairs of value of r and s.

1.5 A bitonic distribution

We've produced a monotone distribution. Can we generate a bitonic (monotone increasing until reaching a peak and then monotone decreasing from there onward) distribution with a simple program?
Let's look at the distribution of k in the following program.

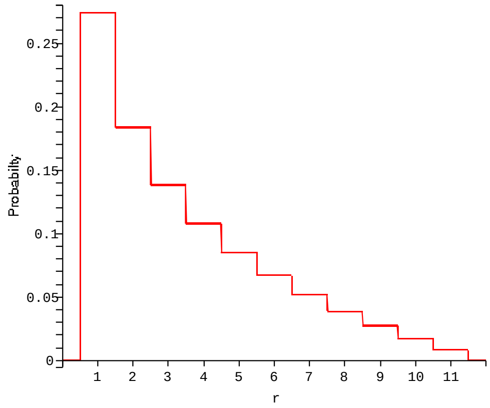

Figure 1.2: Distribution of r in Program 1.1 for N=11

$$
\begin{array}{|l|}
\hline
\{\text{Program 1.3}\} \\
\textbf{var } k; \\
|[\ \textbf{var } i, j; \\
\quad i :\in [1..N]; \\
\quad j :\in [1..N]; \\
\quad \textbf{if } i > j \ \ \rightarrow k :\in [j..i] \\
\quad \square\ i = j \ \ \rightarrow k := i \\
\quad \square\ i < j \ \ \rightarrow k :\in [i..j] \\
\quad \textbf{fi} \\
]|\ \textbf{rav} \\
\hline
\end{array}
$$

Intuitively k should have a higher probability of being around the centre $N/2$, and so the distribution generated should be bitonic. To obtain the exact distribution we calculate:

$$
\begin{array}{ll}
 & wp.(IF).[k = r] \\
= & \qquad \{\text{definition of if}\} \\
 & [i > j]wp.(k :\in [j..i]).[k = r] + [i = j]wp.(k := i).[k = r] \\
 & + [i < j]wp.(k :\in [i..j]).[k = r] \\
= & \qquad \{\text{Rule for uniform choice (1.7)}\} \\
 & \dfrac{1}{i - j + 1}[i > j]\displaystyle\sum_{z=j}^{i}[z = r] + [i = j][i = r] + \dfrac{1}{j - i + 1}[i < j]\displaystyle\sum_{z=i}^{j}[z = r]
\end{array}
\tag{1.15}
$$

Then we calculate the effect of the two assignments from a uniform distribution:

$$wp.\big(i :\in [1..N]; j :\in [1..N]\big).\big(\frac{1}{i-j+1}[i > j]\sum_{z=j}^{i}[z = r] + [i = j][i = r]$$

$$+ \frac{1}{j-i+1}[i < j]\sum_{z=i}^{j}[z = r]\big)$$

$=$ {Rule for uniform choice (1.7)}

$$wp.\big(i :\in [1..N]\big).\big(\frac{1}{N}\sum_{y=1}^{N}\frac{1}{i-y+1}[i > y]\sum_{z=y}^{i}[z = r] + \frac{1}{N}\sum_{y=1}^{N}[i = y][i = r]$$

$$+ \frac{1}{N}\sum_{y=1}^{N}\frac{1}{y-i+1}[i < y]\sum_{z=i}^{y}[z = r]\big)$$

$=$ {Rule for uniform choice (1.7)}

$$\frac{1}{N}\sum_{x=1}^{N}\frac{1}{N}\sum_{y=1}^{N}\frac{1}{x-y+1}[x > y]\sum_{z=y}^{x}[z = r] + \frac{1}{N}\sum_{x=1}^{N}\frac{1}{N}\sum_{y=1}^{N}[x = y][x = r]$$

$$+ \frac{1}{N}\sum_{x=1}^{N}\frac{1}{N}\sum_{y=1}^{N}\frac{1}{y-x+1}[x < y]\sum_{z=x}^{y}[z = r] \tag{1.16}$$

Now we have to deal with this horrific-looking sum:

$$\frac{1}{N}\sum_{x=1}^{N}\frac{1}{N}\sum_{y=1}^{N}\frac{1}{x-y+1}[x > y]\sum_{z=y}^{x}[z = r] + \frac{1}{N}\sum_{x=1}^{N}\frac{1}{N}\sum_{y=1}^{N}[x = y][x = r]$$

$$+ \frac{1}{N}\sum_{x=1}^{N}\frac{1}{N}\sum_{y=1}^{N}\frac{1}{y-x+1}[x < y]\sum_{z=x}^{y}[z = r]$$

$=$ {By symmetry of x, y in the first and third terms}

$$\frac{2}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}\frac{1}{x-y+1}[x > y]\sum_{z=y}^{x}[z = r] + \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x = y][x = r]$$

$=$ {Splitting off using the characteristic function}

$$\frac{2}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}\frac{1}{x-y+1}[x \geq y]\sum_{z=y}^{x}[z = r] - \frac{2}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}\frac{1}{x-y+1}[x = y]\sum_{z=y}^{x}[z = r]$$

$$+ \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x = y][x = r]$$

$$
\begin{aligned}
= & \quad \{\text{Manipulation of characteristic functions}\} \\
& \frac{2}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}\frac{1}{x-y+1}[y \le r \le x] - \frac{2}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] + \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] \\
= & \quad \{\text{arithmetic}\} \\
& \frac{2}{N^2}\sum_{x=r}^{N}\sum_{y=1}^{r}\frac{1}{x-y+1}[1 \le r \le N] - \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] \\
= & \quad \{\text{arithmetic}\} \\
& \frac{2[1 \le r \le N]}{N^2}\sum_{x=r}^{N}(H_x - H_{x-r}) - \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] \\
= & \quad \{\text{arithmetic}\} \\
& \frac{2[1 \le r \le N]}{N^2}(\sum_{x=r}^{N}H_x - \sum_{x=r}^{N}H_{x-r}) - \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] \\
= & \quad \{\text{arithmetic; change of summation index}\} \\
& \frac{2[1 \le r \le N]}{N^2}(\sum_{x=1}^{N}H_x - \sum_{x=1}^{r-1}H_x - \sum_{x=1}^{N-r}H_x) - \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] \\
= & \quad \{\text{Identity: } \textstyle\sum_{1\le k<n} H_k = nH_n - n\} \\
& \frac{2[1 \le r \le N]}{N^2}\big((N+1)H_N - N - rH_{r-1} + (r-1) - (N+1-r)H_{N-r} + (N-r)\big) \\
& - \frac{1}{N^2}\sum_{x=1}^{N}\sum_{y=1}^{N}[x=y=r] \\
= & \quad \{\text{arithmetic}\} \\
& \frac{2[1 \le r \le N]}{N^2}\big((N+1)H_N - rH_{r-1} - (N+1-r)H_{N-r} - 1\big) - \frac{1}{N^2}[1 \le r \le N] \\
= & \quad \{\text{arithmetic}\} \\
& \frac{2[1 \le r \le N]}{N^2}\big((N+1)H_{N+1} - 1 - rH_r + 1 - (N+1-r)H_{N+1-r} + 1 - 1\big) - \frac{1}{N^2}[1 \le r \le N] \\
= & \quad \{\text{arithmetic}\} \\
& \frac{2\big((N+1)H_{N+1} - rH_r - (N+1-r)H_{N+1-r}\big) - 1}{N^2}[1 \le r \le N]
\end{aligned} \tag{1.17}
$$

This distribution is symmetric in the sense that if you substitute $N+1-r$ for r, the expression is invariant. This property is the same as the one for binomial coefficients. Now we want to check that the expression does describe a bitonic distribution. We do this by calculating the difference between successive terms:

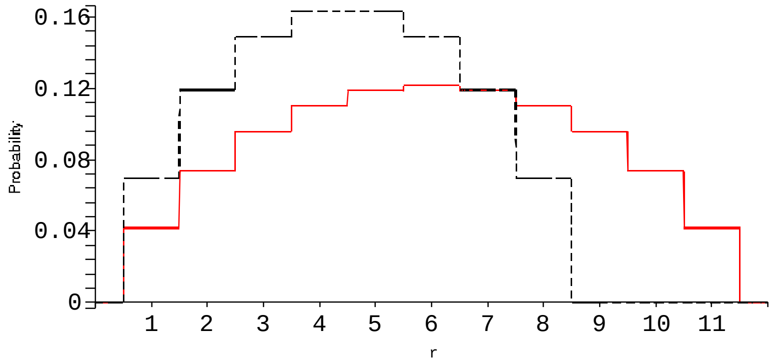

Figure 1.3: Distribution of r in Program 1.2 for N=8 (dashes) and N=11 (solid line)

Call the above expression E_r, and let $1 \leq r \leq N-1$.

$$
\begin{aligned}
& E_{r+1} - E_r \\
= & \frac{2}{N^2}\big(rH_r + (N+1-r)H_{N-r+1} - (r+1)H_{r+1} - (N-r)H_{N-r}\big) \\
= & \frac{2}{N^2}\big(-H_r - 1 + 1 + H_{N-r}\big) \\
= & \frac{2}{N^2}\big(H_{N-r} - H_r\big)
\end{aligned}
\tag{1.18}
$$

which is positive when $r \leq \lfloor \frac{N}{2} \rfloor$, and negative otherwise.

1.6 Summary

The three simple examples in this chapter illustrate how the probabilistic *wp* semantics can be used to calculate the underlying distributions generated by different programs. The method is very similiar to standard GCL and is as powerful. But instead of dealing with predicates we have to manipulate real numbers and characteristic functions of predicates, and quite often we have to deal with some non-trivial sums such as the one in the last example. In the next chapter we are going to consider programs with loops and see what kinds of distribution we can get out of them.

Chapter 2

Two Approaches for Probabilistic Loops

2.1 Introduction

In this chapter we discuss how to reason about probabilistic loops. We present and compare two different approaches. The first follows from our discussion on the expectation transformer model in the last chapter and shows how it can be used to reason about deterministic loops. Then we present another approach using invariants by introducing the probabilistic dynamic logic **Pr**DL of Feldman and Harel[3], with emphasis on the inference rules they gave for loops. These two approaches have their analogs in the standard reasoning of *GCL* programs.

2.2 The wp semantics for the loop construct

In his book *A Discipline of Programming*[1] Dijkstra gave the formal definition of the weakest precondition of a loop construct, and we quote him as follows:

> Let "DO" be the name of the statement
>
> $$\textbf{do } B_1 \to SL_1 \square B_2 \to SL_2 \square \cdots \square B_n \to SL_n \textbf{ od} \tag{2.1}$$
>
> and let "IF" be the name of the statement formed by embracing the same guarded command set by the bracket pair "**if**...**if**". The conditions $H_k(R)$ are given by
>
> $$H_0(R) = R \wedge \neg(\exists j : 1 \leq j \leq n : B_j) \tag{2.2}$$
>
> and for $k > 0$:
>
> $$H_k(R) = wp.IF.H_{k-1}(R) \vee H_0(R) \tag{2.3}$$

then

$$wp.DO.R = (\exists k : k \geq 0 : H_k(R)) \tag{2.4}$$

Here the intuitive understanding of $H_k(R)$ is: the weakest precondition such that the **do** $-$ **od**-construct will terminate after at most k selections of a guarded command, leaving the system in a final state satisfying the postcondition R.

Note that for deterministic programs, we can change the definition of $H_k(R)$ to:

$$H_k(R) = wp.IF.H_{k-1}(R) \tag{2.5}$$

by dropping the second disjunct $H_0(R)$. The meaning of $H_k(R)$ changes to the weakest precondition such that the **do** $-$ **od** construct terminates after *exactly* k selections of a guarded command. The definition and meaning of $wp.DO.R$ remain unchanged since the law

$$wp.prog.(P \vee Q) = wp.prog.P \vee wp.prog.Q \tag{2.6}$$

holds for deterministic programs.

We can generalize from this to obtain a wp semantics for the loop construct in $pGCL$, keeping in mind that we are dealing with expectations instead of predicates.

Definition Let "DO" be the name of the statement

$$\mathbf{do}\ B_1 \rightarrow SL_1 \Box B_2 \rightarrow SL_2 \Box \cdots \Box B_n \rightarrow SL_n\ \mathbf{od} \tag{2.7}$$

where the guards $B_1, B_2, \cdots B_n$ are disjoint and the commands $SL_1, SL_2, \cdots SL_n$ are deterministic. Let "IF" be the name of the statement formed by embracing the same guarded command set by the bracket pair "**if**...**if**". The expectations $J_k([R])$ are given by

$$J_0([R]) = [R \wedge \neg(\exists j : 1 \leq j \leq n : B_j)] \tag{2.8}$$

and for $k > 0$:

$$J_k([R]) = wp.IF.J_{k-1}([R]) \tag{2.9}$$

Then

$$wp.DO.R = (\sum k : k \geq 0 : J_k([R])) \tag{2.10}$$

In the definition $J_k([R])$ is the weakest pre-expectation such that **DO** will terminate after *exactly* k selections of a guarded command, leaving the system in a final state satisfying the postcondition R. We need to sum up the pre-expectations because it is possible for the **DO** construct to achieve the postcondition R after different numbers of iterations.

2.3 The Binomial Distribution using the wp approach

In chapter 1 we promised a cleaner program that generates a Binomial(N, p) distribution, and now here it is:

{Program 2.1}
$|[$**var** $x, n := 0, 0;$
 do $n < N \rightarrow$
  $x := x \oplus_p x + 1;$
  $n := n + 1;$
 od
$]|$

This is a good program to be a first test of our loop rule. The structure of the loop is standard, the only ingredient that make it probabilistic is the probabilistic assignment in the loop body. It would be too bad if our generalized wp semantics for **DO** could not handle a simple loop like this one. We want to find out the distribution of x, or $\mathbf{Pr}(x = r)$, for some $r \geq 0$.

Following (2.8) we define:

$$J_0([x = r]) = [x = r][n \geq N] \tag{2.11}$$

Using (2.9),

$$
\begin{array}{ll}
 & J_1([x = r]) \\
\equiv & \quad \{\text{definition}\} \\
 & wp.IF.J_0([x = r]) \\
\equiv & \quad \{\text{if statement}\} \\
 & [n < N]wp.\big(x := x \oplus_p x + 1; n := n + 1\big).\big([x = r][n \geq N]\big) \qquad (2.12) \\
\equiv & \quad \{\text{assignment; probabilistic choice}\} \\
 & [n < N]\big(p[x + 1 = r][n + 1 \geq N] + (1 - p)[x = r][n + 1 \geq N]\big) \\
\equiv & \quad \{n < N \wedge n + 1 \geq N \Rightarrow n = N\} \\
 & [n + 1 = N]\big(p[x + 1 = r] + (1 - p)[x = r]\big)
\end{array}
$$

In general, we find

$$J_k([x = r]) = \sum_{i=0}^{k} \binom{k}{i} p^i (1 - p)^{k-i} [x + i = r][n + k = N] \tag{2.13}$$

To show invariance we need to show

$$wp.IF.J_{k-1}([x = r]) = J_k([x = r]) \tag{2.14}$$

This is a straightforward calculation:

$$
\begin{array}{ll}
& wp.IF.J_{k-1}([x=r]) \\
\equiv & \quad \{\text{if statement}\} \\
& [n<N]wp.(x:=x\oplus_p x+1; n:=n+1). \\
& \bigl(\sum_{i=0}^{k-1}\binom{k-1}{i}p^i(1-p)^{k-1-i}[x+i=r][n+k-1=N]\bigr) \\
\equiv & \quad \{\text{assignment; probabilistic choice}\} \\
& [n<N]\bigl(p\sum_{i=0}^{k-1}\binom{k-1}{i}p^i(1-p)^{k-1-i}[x+1+i=r][n+k=N] \\
& +(1-p)\sum_{i=0}^{k-1}\binom{k-1}{i}p^i(1-p)^{k-1-i}[x+i=r][n+k=N]\bigr) \\
\equiv & \quad \{\text{arithmetic; } n+k=N \Rightarrow n<N\} \\
& [n+k=N]\bigl(\sum_{i=0}^{k-1}\binom{k-1}{i}p^{i+1}(1-p)^{k-1-i}[x+1+i=r] \\
& +\sum_{i=0}^{k-1}\binom{k-1}{i}p^i(1-p)^{k-i}[x+i=r]\bigr) \\
\equiv & \quad \{\text{arithmetic; change of summation index}\} \\
& [n+k=N]\bigl(p^k+\sum_{i=1}^{k-1}\binom{k-1}{i-1}p^i(1-p)^{k-i}[x+i=r] \\
& +\sum_{i=1}^{k-1}\binom{k-1}{i}p^i(1-p)^{k-i}[x+i=r]+(1-p)^k\bigr) \\
\equiv & \quad \{\text{arithmetic}\} \\
& [n+k=N]\bigl(p^k+\sum_{i=1}^{k-1}(\binom{k-1}{i-1}+\binom{k-1}{i})p^i(1-p)^{k-i}[x+i=r]+(1-p)^k\bigr) \\
\equiv & \quad \{\text{arithmetic}\} \\
& [n+k=N]\sum_{i=0}^{k}\binom{k}{i}p^i(1-p)^{k-i}[x+i=r] \\
\equiv & \quad \{\text{definition}\} \\
& J_k([x=r])
\end{array}
\tag{2.15}
$$

Thus by (2.10) we can conclude

$$wp.DO.[x=r]=\sum_{k\geq 0}J_k([x=r]) \tag{2.16}$$

Lastly we calculate the effect of initialization. Notice that in the above infinite sum all the terms vanish except for $J_N([x = r])$ because $[0 + k = N] = 1$ if and only if $k = N$. So we have as our pre-expectation:

$$
\begin{aligned}
& wp.(x, n := 0, 0).J_N([x = r]) \\
\equiv & \qquad \{\text{assignment}\} \\
& \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} [i = r]
\end{aligned} \tag{2.17}
$$

This is the distribution we want.

Remark For this example we can eliminate k from the definition of $J_k([x = r])$ to obtain an invariant that is independent of the number of iterations:

$$
\begin{aligned}
& J_k([x = r]) \\
\equiv & \qquad \{\text{definition}\} \\
& [n + k = N] \sum_{i=0}^{k} \binom{k}{i} p^i (1-p)^{k-i} [x + i = r] \\
\equiv & \qquad \{\text{since } k = N - n\} \\
& \sum_{i=0}^{N-n} \binom{N-n}{i} p^i (1-p)^{N-n-i} [x + i = r]
\end{aligned} \tag{2.18}
$$

This expression can easily be shown to be invariant. This type of invariant can be used to give a more succinct proof of the distribution since it does not involve the number of iterations k. Sometimes it is difficult to guess such an invariant directly, but it can be derived using our methods from the definitions of $J_k([R])$.

2.4 The Geometric Distribution using the wp approach

After having successfully treated the binomial distribution let's test our loop rules on a slightly more difficult distribution, the geometric distribution. Unlike the binomial distribution, the number of iterations of the loop body is not fixed. This make things more interesting:

```
{Program 2.2}
|[var x, k := 0, 0;
  do x = 0 →
    x := 0 ⊕_p 1;
    k := k + 1
  od
|]
```

We want to find the distribution of n.

Define

$$J_0([n = r]) = [x \neq 0][n = r] \tag{2.19}$$

and for $k > 0$,

$$J_k([n = r]) = p(1 - p)^{k-1}[x = 0][n + k = r] \tag{2.20}$$

To prove the invariance of $J_k([n = r])$ we need to show:

1. $wp.IF.J_{k-1}([n = r]) = J_k([n = r])$ and
2. $wp.IF.J_0([n = r]) = J_1([n = r])$, since $J_0([n = r])$ is of a slightly different form

For the general case we reason

$$
\begin{aligned}
& wp.IF.J_{k-1}([n = r]) \\
\equiv \quad & \{\text{if statement}\} \\
& [x = 0]wp.\big(x := 0 \oplus_p 1; n := n + 1\big).\big(p(1 - p)^{k-2}[x = 0][n + k - 1 = r]\big) \\
\equiv \quad & \{\text{assignment; probabilistic choice}\} \\
& [x = 0]\big(p(1 - p)^{k-1}[0 = 0][n + k = r] + p^2(1 - p)^{k-2}[1 = 0][n + k = r]\big) \\
\equiv \quad & \{\text{arithmetic}\} \\
& p(1 - p)^{k-1}[x = 0][n + k = r] \\
\equiv \quad & \{\text{definition}\} \\
& J_k([n = r])
\end{aligned}
\tag{2.21}
$$

Next for $J_0([n = r])$:

$$
\begin{aligned}
& wp.IF.J_0([n = r]) \\
\equiv \quad & \{\text{if statement}\} \\
& [x = 0]wp.\big(x := 0 \oplus_p 1; n := n + 1\big).\big([x \neq 0][n = r]\big) \\
\equiv \quad & \{\text{assignment; probabilistic choice}\} \\
& [x = 0]\big(p[1 \neq 0][n + 1 = r] + (1 - p)[0 \neq 0][n + 1 = r]\big) \qquad (2.22) \\
\equiv \quad & \{\text{arithmetic}\} \\
& p[x = 0][n + 1 = r] \\
\equiv \quad & \{\text{definition}\} \\
& J_1([n = r])
\end{aligned}
$$

Hence $J_k([n = r])$ satisfies the conditions in the wp semantics of the loop construct, and we can conclude that the weakest pre-expectation of the loop is the infinite sum given in (2.10), with $R = [n = r]$. After putting in the initial conditions $(x, n := 0, 0)$ it is easy to see that all the pre-expectations $J_k([x = r])$ are 0 except for $k = r$, if $r \geq 1$ ($J_0([n = r])$ is always 0 because it does not satisfy the initial conditions, indicating that it is impossible for n to be 0 when the program terminates). Hence there is only one non-zero term in the infinite sum and the weakest pre-expectation for the whole program is:

$$
\begin{aligned}
& J_r([x = r])(x, n := 0, 0) \\
\equiv\; & p(1 - p)^{r-1} \quad \text{for } r \geq 1 \qquad (2.23) \\
\equiv\; & p(1 - p)^{r-1}[r \geq 1]
\end{aligned}
$$

Following from the remark in the last section we can eliminate k from the expression of $J_k([n = r])$ to obtain:

$$
p(1 - p)^{r-n-1}[x = 0][n < r] + [x \neq 0][n = r] \qquad (2.24)
$$

as invariant. This is the invariant used in Morgan and McIver[12], where they treat the case $p = 1/2$. Our derivation have provided a method for "guessing" it.

Remark It is not always possible to obtain an invariant that is independent of the number of iterations k. For example, if we change the post-expectation from $[n = r]$ to $[n > r]$ in the last example, we cannot obtain an invariant that is free of k via substituting k with expressions like $N - n$. However, we can still use $J_k([n > r])$ and the wp semantics for the loop construct to find out the weakest pre-expectation.

2.5 An alternative approach using the invariance theorem

Recall that in standard GCL when we reason about loops, we usually do not use the formal definition of **DO**. Instead we use the method of invariants: a certain invariant is established and maintained. Then if the loop terminates, it is guaranteed to terminate in a state satisfying the invariant. In program development it is much easier to think about how incrementally to reach a goal state from the initial condition, rather than thinking about how to go back to the initial state from the goal state, as required by the formal wp semantics of **DO**. Although we are not doing any program development in $pGCL$, it would be useful to have an approach that is similar to the invariance theorem in GCL.

In GCL when we prove the invariant is maintained we frequently need to use logical implication $\Rightarrow$. However in the expectation transformer model there is no good generalization of this: arguments using pointwise domination $\Rrightarrow$ are expected to lose some probabilistic mass unless it is exact equivalence $\equiv$ each time. As a result we would get subdistributions instead of distributions that sum to 1.

We want to be able to use logical implication effectively to reason about the maintainence of the invariants properly. The probabilistic dynamic logic **Pr**DL of Feldman and Harel[3] provides such a tool.

2.6 A Brief Description of PrDL

We give below a brief description of **Pr**DL. For details please see Feldman and Harel[3].

2.6.1 Syntax

Following the approach of Ramshaw[15] the basic concept of the language is frequency instead of probability. Frequency is a general finite measure that is almost the same as probability except that it is not normalized. We use $\mathbf{Fr}(P)$ to denote the frequency of the predicate P is true. We define **Pr** in terms of **Fr** as:

$$\mathbf{Pr}(P) = if\ \mathbf{Fr}(\mathbf{true}) \neq 0\ then\ \frac{\mathbf{Fr}(P)}{\mathbf{Fr}(\mathbf{true})}\ else\ 0 \tag{2.25}$$

which normalizes $\mathbf{Fr}(P)$ to a probability measure.

The syntax of **Pr**DL is derived from Pratt's first order dynamic logic, which is based on modal logic. The statement $\{\alpha\}P$, where α is a program, P a predicate, is true when P holds after the execution of the program α. The Hoare triple $\{P\}Q\{R\}$ can be expressed in **Pr**DL as:

$$P \Rightarrow \{Q\}R \tag{2.26}$$

The following are some examples of formulae in **Pr**DL:

1. The fact that a program α ends with certainty is expressed by the **Pr**DL formula
$$\mathbf{Fr}(true) = 1 \Rightarrow \mathbf{Fr}(true) = 1 \tag{2.27}$$

2. Partial correctness of α relative to P and Q is given by
$$\mathbf{Pr}(P) = 1 \Rightarrow \{\alpha\}(\mathbf{Pr}(Q) = 1 \vee \mathbf{Fr}(true) = 0) \tag{2.28}$$

3. This program halts with probability at least v:
$$\mathbf{Fr}(true) = 1 \Rightarrow \{\alpha\}\mathbf{Fr}(true) \geq v \tag{2.29}$$

4. The first head will occur at the nth toss with probability 2^{-n}:
$$\mathbf{Fr}(true) = 1 \Rightarrow \{i := 1; x := random; \mathbf{while}\ x < \frac{1}{2}\ \mathbf{do}\ (x := random; \\ i := i + 1)\}\mathbf{Fr}(true) = 1 \wedge \forall n(\mathbf{nat}(n) \wedge n \geq 1 \Rightarrow \mathbf{Fr}(i = n) = 2^{-n}) \tag{2.30}$$

2.6.2 The axiomatic proof system

Below we transcribe the axioms presented in Feldman and Harel:

Axioms

A0. All axioms schemes of the first-order predicate calculus (using the extended definitions of free occurrences of variables and substitutability)

A1. $(\{\alpha\}P) \equiv P$ for a program-free formula P conatining no occurences of **Fr** (and hence no program variables either).

A2. $\{\alpha\}(P \vee Q) \equiv \{\alpha\}P \vee \{\alpha\}Q$.

A3. $\{\alpha\}\neg P \equiv \neg\{\alpha\}P$.

A4. $\{\alpha\}\exists vP \equiv \exists v\{\alpha\}P$ if $v \notin \alpha$.

A5. $\{x := t\}\mathbf{Fr}(P) = v \equiv \mathbf{Fr}(P[x/t]) = v$.

A6. $\{x :=?\}\mathbf{Fr}(P) = v \equiv \mathbf{Fr}(P[x/random_i(1)]) = v$ for any constant i such that the subterm $random_i(t)$ does not appear elsewhere in P for any program term t

A7. $\{\alpha;\beta\}P \equiv \{\alpha\}\{\beta\}P$

For the next axiom we introduce an abbreviation: for an event R, the program $R!$ will be thought of as standing for

$$\mathbf{while}\ \neg R\ \mathbf{do\ skip} \tag{2.31}$$

The program $R!$ eliminates all mass which does not satisfy R; it changes a probability measure to the unnormalized conditional probability $\mathbf{Fr}(\cdot|R)$. Below we supply a special axiom for this program.

A8. $\{\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta\}\mathbf{Fr}(Q) = v$
$\equiv\ \exists u(\{P!;\alpha\}\mathbf{Fr}(Q) = u \wedge \{\neg P!;\beta\}\mathbf{Fr}(Q) = v - u)$.

A9. $\{P!\}\mathbf{Fr}(Q) = v\ \equiv\ \mathbf{Fr}(P \wedge Q) = v$.

Inference Rules

R1. Modus Ponens

$$\frac{P, P \Rightarrow Q}{Q} \tag{2.32}$$

R2. Generalization

$$\frac{P}{\{\alpha\}P};\ \frac{P}{\forall v_i P} \tag{2.33}$$

R3. **while** Rule

$$\frac{\begin{array}{r}\forall s \forall n \mathbf{nat}(n) \Rightarrow (Q(n,s) \Rightarrow \{P!;\alpha\}Q(n+1,s)) \\ \forall s \forall n \mathbf{nat}(n) \Rightarrow (Q(n,s) \Rightarrow \mathbf{Fr}(\neg P \wedge R) = s(n))\end{array}}{\exists s(\sum_{n=0}^{\infty} s(n) = v \wedge Q(0,s)) \Rightarrow \{\mathbf{while}\ P\ \mathbf{do}\ \alpha\}\mathbf{Fr}(R) = v} \tag{2.34}$$

where s is thought of as a free function variable in Q representing a real sequence and the infinite sum is shorthand for the appropriate formula.

Many of these axioms and inference rules are related to the wp semantics of $pGCL$ in the expectation transformer model.

2.7 The Binomial Distribution using the invariant approach

Let us see how the proof system works by working a few examples. We are not going to do a completely formal proof in the axiomatic proof system since it is too tedious and not particularly instructive. Moreover the underlying programming language of **Pr**DL is not $pGCL$, so it would be impossible to do a complete proof in the system without some translation. Instead we are going to be inspired by the syntax and the idea in the transformation rules in the axioms to provide a proof as it was in standard GCL.

Recall the binomial distribution program in the previous section. We can use it as a first example of reasoning with invariants.

Define the invariant I as:

$$I \equiv \mathbf{Pr}(x = r) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} [k = r] \wedge 0 \leq n \leq N \tag{2.35}$$

The invariant is established upon initialization:

$$\begin{aligned} & I(x, n := 0, 0) \\ \equiv\ & \mathbf{Pr}(0 = r) = \sum_{k=0}^{0} \binom{0}{k} p^k (1-p)^{0-k} [k = r] \wedge 0 \leq 0 \leq N \\ \equiv\ & \mathbf{Pr}(r = 0) = [r = 0] \\ \equiv\ & \mathbf{True} \end{aligned} \tag{2.36}$$

$\mathbf{Pr}(r = 0) = [r = 0]$ evaluates to **True** as r is a constant (e.g. $\mathbf{Pr}(0 = 0) = [0 = 0] = 1; \mathbf{Pr}(1 = 0) = [1 = 0] = 0$).

Now we need to show $I \wedge G \Rightarrow I(x := x \oplus_p x + 1; n := n + 1)$.

$$I(x := x \oplus_p x + 1; n := n + 1)$$

$\equiv$ {definition}

$$\mathbf{Pr}(x = r) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} [k = r] \wedge (0 \leq n \leq N) \big(x := x \oplus_p x + 1; n := n + 1\big)$$

$\equiv$ {assignment}

$$\mathbf{Pr}(x = r) = \sum_{k=0}^{n+1} \binom{n+1}{k} p^k (1-p)^{n+1-k} [k = r] \wedge (0 \leq n + 1 \leq N) \big(x := x \oplus_p x + 1\big)$$

$\equiv$ {probabilistic choice}

$$p\mathbf{Pr}(x + 1 = r) + (1-p)\mathbf{Pr}(x = r) = \sum_{k=0}^{n+1} \binom{n+1}{k} p^k (1-p)^{n+1-k} [k = r] \wedge (0 \leq n + 1 \leq N)$$

$\equiv$ $\{I \Rightarrow \mathbf{Pr}(x = r) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} [k = r]\}$

$$\begin{aligned} & p \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} [k = r - 1] + (1-p) \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} [k = r] \\ & = \sum_{k=0}^{n+1} \binom{n+1}{k} p^k (1-p)^{n+1-k} [k = r] \wedge (0 \leq n + 1 \leq N) \end{aligned}$$

$\equiv$ {arithmetic}

$$\sum_{k=0}^{n+1} \binom{n+1}{k} p^k (1-p)^{n+1-k} [k = r] = \sum_{k=0}^{n+1} \binom{n+1}{k} p^k (1-p)^{n+1-k} [k = r] \wedge (0 \leq n + 1 \leq N)$$

$$
\begin{aligned}
\equiv\quad & \{\text{arithmetic}\} \\
& 0 \le n+1 \le N \\
\equiv\ & \{I \wedge G \Rightarrow 0 \le n < N\} \\
& \textbf{True}
\end{aligned}
\tag{2.37}
$$

Hence $\{I, G\} \vdash I(x := x \oplus_p x+1; n := n+1)$ and by the Deduction Theorem, we have proved $I \wedge G \Rightarrow I(x := x \oplus_p x+1; n := n+1)$.

We conclude upon termination (termination can be argued using standard variant methods):

$$
\begin{aligned}
& I \wedge \neg G \\
\equiv\quad & \{\text{definition}\} \\
& \mathbf{Pr}(x = r) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} [k = r] \wedge (0 \le n \le N) \wedge (n \ge N) \\
\equiv\quad & \{(0 \le n \le N \wedge n \ge N) \Rightarrow n = N\} \\
& \mathbf{Pr}(x = r) = \sum_{k=0}^{N} \binom{N}{k} p^k (1-p)^{N-k} [k = r]
\end{aligned}
\tag{2.38}
$$

which is the distribution we want.

Now we are in the paradise of reasoning with invariants again.

2.8 The Geometric Distribution as a further illustration of the invariant method

The style of argument we used for the binomial distribution in the last section was very much like the way we use invariants to argue about the correctness of programs in standard *GCL*. However, such simplicity is somehow misleading because the program we treated was not much different from standard *GCL* program: the guard is standard $(n < N)$, and the loop could only terminate at the Nth iteration.

This is not the case for many interesting probabilistic programs. For example, the following geometric distribution program (Program 2.2) we have discussed in the previous section.

We want to find out the the distribution of k. To be more precise, for $r \ge 1$, what is $\mathbf{Pr}(k = r)$?

From the inference rule for **while** loops **R3** in **Pr**DL, we know that the invariant should contain two parts. One part should describe the states that stay inside the loop from iteration to iteration, while the other should describe the the probability of the states that escape from the loop at each iteration. We guess the invariant:

$$I(n) = \begin{pmatrix} x = 0 \\ k = 0 \end{pmatrix} \vee \begin{pmatrix} n \geq 1 \\ \mathbf{Fr}(x = 0, k = n) = (1-p)^n \\ \mathbf{Fr}(x = 1, k = n) = p(1-p)^{n-1} \end{pmatrix} \tag{2.39}$$

The invariant is established upon initialization because the first disjunct holds. Next we need to show the invariant is maintained:

$$\begin{array}{ll}
& I(n+1)(x := 0 \oplus_p 1; k := k+1) \\
\equiv & \{\text{assignment}\} \\
& \begin{pmatrix} x = 0 \\ k+1 = 0 \end{pmatrix} \vee \begin{pmatrix} n+1 \geq 1 \\ \mathbf{Fr}(x = 0, k+1 = n+1) = (1-p)^{n+1} \\ \mathbf{Fr}(x = 1, k+1 = n+1) = p(1-p)^n \end{pmatrix} (x := 0 \oplus_p 1) \\
\equiv & \{\text{probabilistic choice}\} \\
& \begin{pmatrix} x = 0 \\ k+1 = 0 \end{pmatrix} \vee \begin{pmatrix} n+1 \geq 1 \\ (1-p)\mathbf{Fr}(0 = 0, k = n) + p\mathbf{Fr}(1 = 0, k = n) = (1-p)^{n+1} \\ (1-p)\mathbf{Fr}(0 = 1, k = n) + p\mathbf{Fr}(1 = 1, k = n) = p(1-p)^n \end{pmatrix} \\
\equiv & \{\text{the first disjunct is false as } k \geq 0\text{; arithmetic}\} \\
& \begin{pmatrix} n \geq 0 \\ \mathbf{Fr}(k = n) = (1-p)^n \\ \mathbf{Fr}(k = n) = (1-p)^n \end{pmatrix} \\
\equiv & \{\text{arithmetic}\} \\
& \begin{pmatrix} n \geq 0 \\ \mathbf{Fr}(x = 0, k = n) + \mathbf{Fr}(x \neq 0, k = n) = (1-p)^n \end{pmatrix} \\
\equiv & \{\text{guard implies } \mathbf{Fr}(x \neq 0) = \mathbf{Fr}(x \neq 0, k = n) = 0\} \\
& \begin{pmatrix} n \geq 0 \\ \mathbf{Fr}(x = 0, k = n) = (1-p)^n \end{pmatrix} \\
\Leftarrow & I(n) \quad \text{if } n \geq 1
\end{array} \tag{2.40}$$

If $n = 0$ (i.e., at initialization), the invariant is still maintained because

$$(n = 0) \wedge (x = 0, k = 0) \Rightarrow n \geq 0 \wedge \mathbf{Fr}(x = 0, k = n) = 1 \tag{2.41}$$

Thus we have shown $I(n) \wedge G \Rightarrow I(n+1)$.

Up till now it is still very much like a standard invariant argument. However the next part is different. Clearly

$$\begin{array}{l} I(0) \Rightarrow \mathbf{Fr}(x = 0, k = 0) = 0 \\ I(n) \Rightarrow \mathbf{Fr}(x = 1, k = n) = p(1-p)^{n-1} \quad \text{for } n \geq 1 \end{array} \tag{2.42}$$

Now following the inference rule for **while** loop **R3** we define

$$
\begin{aligned}
&s(0) = 0 \\
&s(n) = p(1-p)n-1[n=r] \quad \text{for } n \geq 1
\end{aligned} \tag{2.43}
$$

Then certainly

$$s(n) = \mathbf{Fr}(x = 1, k = n, k = r) \tag{2.44}$$

Hence we can infer from **R3**

$$
\begin{aligned}
&\mathbf{Fr}(k = r) \\
\equiv &\sum_{n=0}^{\infty} s(n) \\
\equiv &\sum_{n=1}^{\infty} p(1-p)^{n-1}[n = r] \\
\equiv &p(1-p)^{r-1}[r \geq 1]
\end{aligned} \tag{2.45}
$$

Now as

$$1 \geq \mathbf{Fr}(\mathbf{true}) \geq \mathbf{Fr}(k \geq 1) = \sum_{r=1}^{\infty} p(1-p)^{r-1} = 1 \tag{2.46}$$

Therefore $\mathbf{Fr}(\mathbf{true}) = 1$ and we can conclude $\mathbf{Pr}(k = r) = p(1-p)^{r-1}$ by the definition of $\mathbf{Pr}$.

2.9 Summary

We have presented two different methods for reasoning about loops in $pGCL$, illustrated by the two of the most common discrete distributions in probability theory, the binomial distribution and geometric distribution. In the next chapter we are going to analyse a few further examples using these two methods in order to evaluate them as methods.

Chapter 3

The Careful Treatment of Two Examples

3.1 Introduction

Having introduced the *wp* method and the invariant method in the last chapter, we devote this chapter to the careful treatment of two more examples. We would argue that the latter method is better for treating more complicated loops. Our purpose is twofold: firstly we want to demonstrate how to analyse a loop and guess the invariant; secondly we hope to convince you the examples are interested in themselves.

3.2 The maximum of M uniform random variables

Consider the following scenario: a group of M children have a new video game and they all want to be the first one to play it. They decide to determine who is going to be the first one by throwing a die in turn to pick out the one with the largest number; if there were more than one winner they repeat this process among the winners. The question is: what is the distribution of the greatest number they have after the first round of throws?

This is basically a question of determining the distribution of the maximum of M uniform random variables in the range $[1..6]$, and can be modelled easily using a $pGCL$ program. But instead of the range $[1..6]$ we treat the more general case $[1..N]$ (we assume a fair N-sided die, which could be a coin when $N = 2$, the corresponding Platonic solids when $N = 4, 6, 8, 12, 20$, and perhaps in the form of an N-sided regular prism for all other N).
Consider the following program:

```
{Program 3.1}
|[var n, w := 0, 0;
  do n < M →
      x :∈ [1..N];
      if x > w → w := x
      □ x ≤ w → skip
      fi;
      n := n + 1
  od
]|
```

In the above program, w stores the maximum of the n throws so far while x is a temporary store of the result of the throw in each iteration. We want to find the distribution of w, but it is not clear what the invariant should be. For standard *GCL* programs we have a whole set of heuristics to try (taking conjuncts as invariants, replacing constants by variables, etc; for details see Gries[6], Kaldewaij[10]).

Not having any particularly good idea how to proceed, we can always try simple examples first. We would first use the wp approach to find out the distribution of w for $M = 2, 3$. Obviously for $M = 1$, the distribution is:

$$\mathbf{Pr}(w = k) \;=\; \frac{1}{N}[1 \leq k \leq N] \tag{3.1}$$

since the distribution for the maximum of one uniform random variable is the distribution of the uniform random variable itself.

The case for $M = 2$ is more interesting. We call the loop in the above program **DO**, and the corresponding selection statement of **DO** defined in the last chapter **IF**. We call the selection statement inside the loop **if**. Following the notation in the last chapter, we define:

$$J_0([w = k]) = [n \geq 2][w = k] \tag{3.2}$$

and reason

$$\begin{array}{ll} & J_1([w = k]) \\ \equiv & \quad \{\text{definition}\} \\ & wp.\mathbf{IF}.J_0([w = k]) \\ \equiv & \quad \{\text{assignment}\} \\ & wp.\big(x :\in [1..N]; \mathbf{if}\big).\big([n + 1 \geq 2][w = k]\big) \\ \equiv & \quad \{\text{if statement}\} \\ & wp.\big(x :\in [1..N]\big).\Big(\big([x > w][x = k] + [x \leq w][w = k]\big)[n + 1 \geq 2]\Big) \end{array} \tag{3.3}$$

$$\begin{aligned}
&\equiv \quad \{\text{Rule for uniform choice (1.7)}\}\\
&\Big(\frac{1}{N}\sum_{j=1}^{N}[j > w][j = k] + \frac{1}{N}\sum_{j=1}^{N}[j \le w][w = k]\Big)[n+1 \ge 2]\\
&\equiv \quad \{\text{arithemtic}\}\\
&\Big(\frac{1}{N}[w < k][1 \le k \le N] + \frac{k}{N}[w = k]\Big)[n+1 \ge 2]
\end{aligned} \tag{3.4}$$

Now for the second iteration:

$$\begin{aligned}
&J_2([w = k])\\
&\equiv \quad \{\text{definition}\}\\
&wp.\mathbf{IF}.J_1([w = k])\\
&\equiv \quad \{\text{assignment}\}\\
&wp.\big(x :\in [1..N]; \mathbf{if}\big).\Big(\Big(\frac{1}{N}[w < k][1 \le k \le N] + \frac{k}{N}[w = k]\Big)[n+2 \ge 2]\Big)\\
&\equiv \quad \{\text{if statement; Rule for uniform choice (1.7)}\}\\
&\Big(\frac{1}{N}\sum_{j=1}^{N}\frac{1}{N}[j > w][j < k][1 \le k \le N] + \frac{1}{N}\sum_{j=1}^{N}\frac{1}{N}[j \le w][w < k][1 \le k \le N]\\
&+\frac{1}{N}\sum_{j=1}^{N}\frac{k}{N}[j > w][j = k] + \frac{1}{N}\sum_{j=1}^{N}\frac{k}{N}[j \le w][w = k]\Big)[n+2 \ge 2]\\
&\equiv \quad \{\text{arithmetic}\}\\
&\Big(\frac{k-w-1}{N^2}[w+1 < k][1 \le k \le N] + \frac{w+k}{N^2}[w < k][1 \le k \le N] + \frac{k^2}{N^2}[w = k]\Big)[n+2 \ge 2]
\end{aligned} \tag{3.5}$$

Finally we put in the initial conditions:

$$\begin{aligned}
&wp.(n, w := 0, 0).J_2([w = k])\\
&\equiv \quad \{\text{assignments}\}\\
&\frac{2k-1}{N^2}[1 \le k \le N]
\end{aligned} \tag{3.6}$$

It is easy to check that the distribution adds up to 1.

Now we try the case $M = 3$. Instead of starting the calculation from scratch, we can follow from our last calulation of $J_2([w = k])$. Since we know that the program terminates after three iterations and k must lie between 1 and N, we drop the characteristic functions $[n + 2 \le 3]$ and $[1 \le k \le N]$ to make the calculations look tidier.

$$
\begin{aligned}
& J_3([w=k]) \\
\equiv \quad & \{\text{if statement}\} \\
& wp.\big(x :\in [1..N]\big).\Big(\frac{k-x-1}{N^2}[x>w][x+1<k] + \frac{k-w-1}{N^2}[x \le w][w+1<k] \\
& + \frac{x+k}{N^2}[x>w][x<k] + \frac{w+k}{N^2}[x \le w][w<k] \\
& + \frac{k^2}{N^2}[x>w][x=k] + \frac{k^2}{N^2}[x \le w][w=k]\Big) \\
\equiv \quad & \{\text{Rule for uniform choice (1.7)}\} \\
& \frac{1}{N}\sum_{j=1}^{N}\frac{k-j-1}{N^2}[j>w][j+1<k] + \frac{1}{N}\sum_{j=1}^{N}\frac{k-w-1}{N^2}[j \le w][w+1<k] \\
& + \frac{1}{N}\sum_{j=1}^{N}\frac{j+k}{N^2}[j>w][j<k] + \frac{1}{N}\sum_{j=1}^{N}\frac{w+k}{N^2}[j \le w][w<k] \\
& + \frac{1}{N}\sum_{j=1}^{N}\frac{k^2}{N^2}[j>w][j=k] + \frac{1}{N}\sum_{j=1}^{N}\frac{k^2}{N^2}[j \le w][w=k]
\end{aligned}
\tag{3.7}
$$

Now we put in the initial conditions to make subsequent simplifications easier:

$$
\begin{aligned}
& wp.(n, w := 0, 0).J_3([w=k]) \\
\equiv \quad & \{\text{assignment}\} \\
& \frac{1}{N}\sum_{j=1}^{N}\frac{k-j-1}{N^2}[j>0][j+1<k] + \frac{1}{N}\sum_{j=1}^{N}\frac{j+k}{N^2}[j>0][j<k] + \frac{1}{N}\sum_{j=1}^{N}\frac{k^2}{N^2}[j>0][j=k] \\
\equiv \quad & \{\text{arithmetic}\} \\
& \frac{(k-2)(k-1)}{N^3} - \frac{1}{N^3}\sum_{j=1}^{k-2} j + \frac{k(k-1)}{N^3} + \frac{1}{N^3}\sum_{j=1}^{k-1} j + \frac{k^2}{N^3} \\
\equiv \quad & \{\text{arithmetic}\} \\
& \frac{3k^2-3k+1}{N^3}[1 \le k \le N]
\end{aligned}
\tag{3.8}
$$

It is easy to check that the distribution adds up to one.

The last calculation was much more tedious than the case for $M = 2$ because the number of terms in the expressions $J_i([w=k])$ increases steadily as M increases. This could be expected because the wp approach is a rather brute-force method. The expressions $J_i([w=k])$ have to describe all the possible paths for w to increase from 0 to k in i steps, and no wonder they get so much more complicated as M increases. It is very difficult to see if there were any general pattern in $J_i([w = k])$ for us to do an inductive proof on.

The two examples in the last chapter could be attacked using the *wp* approach because if you draw out their computation trees, they are highly "linear" (the binomial distribution program does not contain any branches while the geometric program has just one branch to exit the loop during each iteration). An inductive form of $J_i([R])$ was relatively easy to find. This is not the case for our current program, which contains a lot of branches during each iteration.

Since we are not particularly interested in the computation paths used to achieve the postcondition, it would be much better for us to find out a method for obtaining the post-expectation without considering all such computation paths. Another problem with the *wp* approach is that the "invariants" which we obtain from substituting the number of iterations k with expressions like $N - n$ in the last chapter are rather unintuitive: it is difficult to see what the expression actually means if we did not show how it is converted from $J_k([R])$. Perhaps just like the standard *GCL*, the formal *wp* semantics for **DO** is difficult to use, and does not give any insight into the program.

3.2.1 Switching to the invariant approach

At this point we should perhaps try the other approach using invariants. Guessing the invariant requires some ingenuity, but our calculations in the last section provide us with many hints. The expressions for the two distributions look like some binomial expansions without the leading term. If we are still unsure we can calculate the case for $M = 4$ using *wp* (an even more tedious calculation) to obtain $(4k^3 - 6k^2 + 4k - 1)/N^4$. So we guess

$$\mathbf{Pr}(w = k) = \frac{k^M - (k-1)^M}{N^M} \tag{3.9}$$

Let us prove it using the invariant method. Define

$$I \equiv (\forall k : 1 \le k \le N : \mathbf{Pr}(w = k) = \frac{k^n - (k-1)^n}{N^n}) \wedge (0 \le n \le M) \tag{3.10}$$

The invariant is satisfied at initialization:

$$\begin{aligned}
& I \ (n, w := 0, 0) \\
\equiv\ & (\forall k : 1 \le k \le N : \mathbf{Pr}(0 = k) = \frac{k^0 - (k-1)^0}{N^0}) \wedge (0 \le 0 \le M) \\
\equiv\ & (\forall k : 1 \le k \le N : \mathbf{Pr}(k = 0) = 0) \\
\equiv\ & \mathbf{True}
\end{aligned} \tag{3.11}$$

Next we want to prove $I \wedge G \Rightarrow I\ (x :\in [1..N]; \mathbf{if}; n := n+1)$.

$$
\begin{aligned}
& I(n+1)\ (x :\in [1..N]; \mathbf{IF}; n := n+1) \\
\equiv\quad & \{\text{definition; assignment}\} \\
& (\forall k : 1 \le k \le N : \mathbf{Pr}(w = k) = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M)\ \big(x :\in [1..N]; \mathbf{IF}\big) \\
\equiv\quad & \{\text{the law for if statement in } \mathbf{Pr}\text{DL (A8)}\} \\
& (\forall k : 1 \le k \le N : \{(x \le w)!\}\mathbf{Pr}(w = k) + \{(x > w)!\}\mathbf{Pr}(x = k) \\
& = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M)\ \big(x :\in [1..N]\big) \\
\equiv\quad & \{\text{the law for if statement in } \mathbf{Pr}\text{DL (A9)}\} \\
& (\forall k : 1 \le k \le N : \mathbf{Pr}(x \le w \wedge w = k) + \mathbf{Pr}(x > w \wedge x = k) \\
& = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M)\ \big(x :\in [1..N]\big) \\
\equiv\quad & \{\text{Rule for uniform choice (1.7)}\} \\
& (\forall k : 1 \le k \le N : \frac{1}{N}\sum_{j=1}^{N} \mathbf{Pr}(j \le w \wedge w = k) + \frac{1}{N}\sum_{j=1}^{N} \mathbf{Pr}(j > w \wedge j = k) \\
& = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M) \\
\equiv\quad & \{\text{arithmetic}\} \\
& (\forall k : 1 \le k \le N : \frac{k}{N}\mathbf{Pr}(w = k) + \frac{1}{N}\mathbf{Pr}(w < k) = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M) \\
\equiv\quad & \{\text{arithmetic}\} \\
& (\forall k : 1 \le k \le N : \frac{k}{N}\mathbf{Pr}(w = k) + \frac{1}{N}\sum_{i=1}^{k-1} \mathbf{Pr}(w = i) = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M) \\
\equiv\quad & \{\text{assume } I \text{ as inductive hypothesis}\} \\
& (\forall k : 1 \le k \le N : \frac{k}{N}\frac{k^n - (k-1)^n}{N^n} + \frac{1}{N}\sum_{i=1}^{k-1} \frac{i^n - (i-1)^n}{N^n} = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \\
& \wedge (0 \le n+1 \le M) \\
\equiv\quad & \{\text{telescoping sum}\} \\
& (\forall k : 1 \le k \le N : \frac{k^{n+1} - k(k-1)^n}{N^{n+1}} + \frac{(k-1)^n}{N^{n+1}} = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M) \\
\equiv\quad & \{\text{arithmetic}\} \\
& (\forall k : 1 \le k \le N : \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}} = \frac{k^{n+1} - (k-1)^{n+1}}{N^{n+1}}) \wedge (0 \le n+1 \le M) \\
\equiv\ & 0 \le n+1 \le M \\
\equiv\quad & \{\text{the guard } G \text{ implies } n+1 \le M\} \\
& \mathbf{True}
\end{aligned}
\tag{3.12}
$$

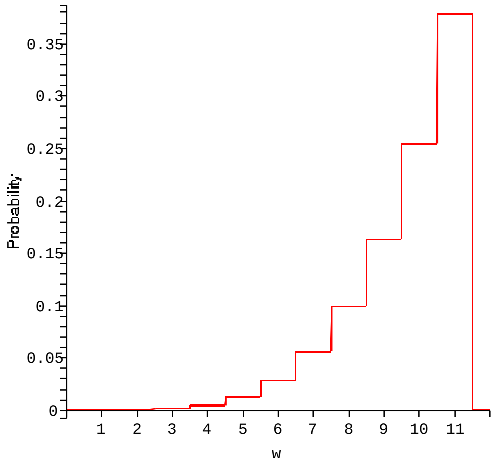

Figure 3.1: Distribution of w in Program 3.1 with N=11 and M=5

Thus upon termination we have:

$$\begin{aligned}
&I \wedge \neg G \\
\equiv\ & (\forall k : 1 \le k \le N : \mathbf{Pr}(w = k) = \frac{k^n - (k-1)^n}{N^n}) \wedge (0 \le n \le M) \wedge (n \ge M) \\
\equiv\ & (\forall k : 1 \le k \le N : \mathbf{Pr}(w = k) = \frac{k^M - (k-1)^M}{N^M})
\end{aligned} \tag{3.13}$$

This is the distribution of the greatest number the M children obtained after the first round of throwing N-sided die. In this form it is very easy to see this telescoping sum adds up to one. The distribution is also monotonically increasing, indicating a higher probability of getting a large number from the range $[1..N]$. This makes good sense as we are taking the maximum of M uniform random variables.

3.2.2 Relation to the continuous case

Now we would like to show how the distribution we obtained relates to the random variable $max(X_1, X_2, \cdots, X_M)$, where X_1, X_2, $\ldots$, X_M are independent identically distributed $U[0,1]$ random variables, where $U[0,1]$ denotes the continous uniform distribution on the interval $[0,1]$.

The cumulative distribution function of X, where X is a discrete uniform random variable on [1..N], is:

$$F_X(x) = \sum_{k=1}^{N} \frac{k}{N} \chi_{(k-1,k]}(x) \tag{3.14}$$

Imagine we squeeze all the points from [1..N] onto the interval $[0,1]$ by defining a random variable Y by $Y = X/N$. The cumulative distribution function of Y is:

$$F_Y(y) = \sum_{k=1}^{N} \frac{k}{N} \chi_{(\frac{k-1}{N},\frac{k}{N}]}(y) \tag{3.15}$$

If we let $N \to \infty$, the right hand side of the above equation converges pointwise to $y\chi_{[0,1]}(y)$. Hence Y converges in distribution to $U(0,1)$.

Now let $Z = \mathbf{max}(X_1, X_2, \cdots, X_M)$, where X_i's are i.i.d. random variables having a discrete uniform distribution on [1..N]. According to previous calculations the c.d.f of Z is:

$$F_Z(z) = \sum_{k=1}^{N} \frac{k^M}{N^M} \chi_{(k-1,k]}(z) \tag{3.16}$$

If we apply the same operations and squeeze all the points onto the interval $[0,1]$ and let N tend to infinity, we have:

$$\lim_{N\to\infty} F_{\frac{Z}{N}}(z) = z^M \chi_{[0,1]}(z) \tag{3.17}$$

This is the c.d.f. of $\mathbf{max}(U_1, U_2, \cdots, U_M)$, where U_1,U_2,...,U_M are i.i.d. random variables having the distribution $U(0,1)$ because

$$\begin{aligned} &P(max(U_1, U_2, \cdots, U_M \leq z) \\ &= P(U_1 \leq z, U_2 \leq z, \cdots, U_M \leq z) \\ &= P(U_1 \leq z)P(U_2 \leq z)\cdots P(U_M \leq z) \\ &= z^M \end{aligned} \tag{3.18}$$

3.2.3 A simple variation

Suppose now we are interested in the distribution of the smallest number the children obtained after the first round of throws. It is easily obtained via the following duality relation:

$$x \ \mathbf{max} \ y = z \equiv (N+1-x) \ \mathbf{min} \ (N+1-y) = N+1-z \tag{3.19}$$

for $1 \leq x, y, z \leq N$. The distribution of the minimum of the M uniform random variables would be the distribution of the maximum from the last section reflected along the line $y = N/2$. To be more precise,

$$\mathbf{Pr}(w = k) = \frac{(N+1-k)^M - (N-k)^M}{N^M} \tag{3.20}$$

3.3 The Gambler's Ruin in a Fair Casino

Now let's look at one of the simplest random processes in probability theory: the symmetric random walk. It can be expressed as a very simple *pGCL* program.

$$
\boxed{
\begin{array}{l}
\{\text{Program 3.2}\} \\
|[\mathbf{var}\ x := 1; \\
\quad \mathbf{do}\ x \neq 0 \rightarrow \\
\quad\quad x := x - 1 \ {}_{\frac{1}{2}}\oplus\ x + 1 \\
\quad \mathbf{od} \\
]|
\end{array}
}
$$

Suppose we want to find the probability of $x = 0$ at termination. Again it would be disaster to try to solve it by brute force using the *wp* approach, as the expectations $J_k([x = 0])$ must describe all the possible paths from 1 back to 0 in k turns. What makes this worse than the last example is that the number of paths is infinite in this case. The loop rules described in McIver and Morgan do not apply to this example because their expectation is "unbounded". However we can still use the invariant method based on **Pr**DL to treat this example.

It would not be particularly helpful for us to try out a few cases to guess the invariant, since there are no cases to try. We could put a barrier N on the random walk, but it is not useful as an unbounded random walk is essentially different from a bounded one, no matter how large the finite bound N is. We need to seek help from probability theory.

By the "Ballot Theorem" proved using the "Reflection Principle" for counting paths (see Feller[4]), for $n > 0$, $k > 0$, the probability of a particle x, starting from 0, to reach k in n turns without returning to the origin is:

$$\mathbf{Pr}(x = k) = \frac{1}{2^n}\frac{k}{n}\binom{n}{\frac{1}{2}(n+k)}[n + k \equiv 0 \mod 2] \tag{3.21}$$

The characteristic function $[n + k \equiv 0 \mod 2]$ expresses the fact that the random walk is periodic with period 2. Since our program starts from 1, we need to modify the above expression to obtain the invariant:

$$\begin{aligned} I(n) \equiv &(\forall k : k > 0 : \mathbf{Fr}(x = k) = \frac{1}{2^n}\frac{k}{n+1}\binom{n+1}{\frac{1}{2}(n+1+k)}[n + 1 + k \equiv 0 \mod 2]) \\ &\wedge \mathbf{Fr}(x = 0) = \frac{1}{2^n}\frac{1}{n}\binom{n}{\frac{1}{2}(n+1)}[n > 0][n + 1 \equiv 0 \mod 2] \end{aligned} \tag{3.22}$$

At initialization $x = 1$, we have

$$
\begin{aligned}
& I(0)\ (x := 1) \\
\equiv\ & (\forall k : k > 0 : \mathbf{Fr}(1 = k) = k \binom{1}{\frac{1}{2}(1+k)} [1 + k \equiv 0 \mod 2]) \\
& \wedge \mathbf{Fr}(1 = 0) = 0 \\
\equiv\ & \mathbf{Fr}(1 = 1) = 1 \wedge (\forall k : k > 1 : \mathbf{Fr}(1 = k) = 0) \wedge \mathbf{Fr}(1 = 0) = 0 \\
\equiv\ & \textbf{True}
\end{aligned}
\tag{3.23}
$$

Thus the invariant is established upon initialization.

Next we need to show the invariance of I. Note that the invariant is a conjunction of statements about the frequencies of x at position k during the nth iteration, so instead of tackling the whole invariant, we can split it into three cases for $k > 1$, $k = 1$ and $k = 0$. The characteristic function $[n + 1 + k \equiv 0 \mod 2]$ in the invariant expresses the fact that the random walk has period 2, and is not used in the calculation below most of the time. So we decide to drop it to make the calculations look tidier.

$$
\begin{aligned}
& \mathbf{Fr}(x = k) = \frac{1}{2^{n+1}} \frac{k}{(n+1)+1} \binom{(n+1)+1}{\frac{1}{2}((n+1)+1+k)} (x := x - 1 \ {}_{\frac{1}{2}}\!\oplus\ x + 1) \\
\equiv\ & \qquad \{\text{assignment}\} \\
& \frac{1}{2}\mathbf{Fr}(x = k+1) + \frac{1}{2}\mathbf{Fr}(x = k-1) = \frac{1}{2^{n+1}} \frac{k}{n+2} \binom{n+2}{\frac{1}{2}(n+2+k)} \\
\equiv\ & \qquad \{\text{assume inductive hypothesis } I(n) \text{ and } k - 1 > 0\} \\
& \frac{1}{2}\frac{1}{2^n}\frac{k+1}{n+1}\binom{n+1}{\frac{1}{2}(n+2+k)} + \frac{1}{2}\frac{1}{2^n}\frac{k-1}{n+1}\binom{n+1}{\frac{1}{2}(n+k)} = \frac{1}{2^{n+1}}\frac{k}{n+2}\binom{n+2}{\frac{1}{2}(n+2+k)} \\
\equiv\ & \qquad \{\text{arithmetic}\} \\
& \frac{k}{n+1}(\binom{n+1}{\frac{1}{2}(n+2+k)} + \binom{n+1}{\frac{1}{2}(n+k)}) + \frac{1}{n+1}(\binom{n+1}{\frac{1}{2}(n+2+k)} - \binom{n+1}{\frac{1}{2}(n+k)}) \\
& = \frac{k}{n+2}\binom{n+2}{\frac{1}{2}(n+2+k)} \\
\equiv\ & \qquad \{\text{arithmetic}\} \\
& \frac{k}{n+1}\binom{n+2}{\frac{1}{2}(n+2+k)} + \frac{1}{n+1}\frac{k}{n+2}\binom{n+2}{\frac{1}{2}(n+2+k)} \\
& = \frac{k}{n+2}\binom{n+2}{\frac{1}{2}(n+2+k)}
\end{aligned}
\tag{3.24}
$$

$$\begin{aligned}
&\equiv \quad \{\text{arithmetic}\}\\
&\quad \frac{k}{n+2}\binom{n+2}{\frac{1}{2}(n+2+k)} = \frac{k}{n+2}\binom{n+2}{\frac{1}{2}(n+2+k)}\\
&\equiv \textbf{True}
\end{aligned} \tag{3.25}$$

For $k = 1$, we can start from the second line of the above calculations and substitute $k = 1$:

$$\begin{aligned}
&\frac{1}{2}\mathbf{Fr}(x = 1+1) + \frac{1}{2}\mathbf{Fr}(x = 1-1) = \frac{1}{2^{n+1}}\frac{1}{n+2}\binom{n+2}{\frac{1}{2}(n+2+1)}\\
\equiv &\quad \{\text{guard implies } \mathbf{Fr}(x=0)=0\}\\
&\frac{1}{2}\mathbf{Fr}(x=2) = \frac{1}{2^{n+1}}\frac{1}{n+2}\binom{n+2}{\frac{1}{2}(n+3)}\\
\equiv &\quad \{\text{inductive hypothesis } I(n)\}\\
&\frac{1}{2}\frac{1}{2^n}\frac{2}{n+1}\binom{n+1}{\frac{1}{2}(n+3)} = \frac{1}{2^{n+1}}\frac{1}{n+2}\binom{n+2}{\frac{1}{2}(n+3)}\\
\equiv &\quad \{\text{arithmetic}\}\\
&\frac{2}{n+1}\binom{n+1}{\frac{1}{2}(n+3)} = \frac{1}{n+2}\binom{n+2}{\frac{1}{2}(n+1)}\\
\equiv &\quad \{\text{arithmetic}\}\\
&\frac{2}{n+1}\binom{n+1}{\frac{1}{2}(n+3)} = \frac{1}{n+2}\frac{n+2}{\frac{1}{2}(n+1)}\binom{n+1}{\frac{1}{2}(n-1)}\\
\equiv &\quad \{\text{arithmetic}\}\\
&\frac{2}{n+1}\binom{n+1}{\frac{1}{2}(n+3)} = \frac{2}{n+1}\binom{n+1}{\frac{1}{2}(n+3)}\\
\equiv &\ \textbf{True}
\end{aligned} \tag{3.26}$$

The invariance of $k = 0$ is even easier:

$$\begin{aligned}
&\frac{1}{2}\mathbf{Fr}(x = 0+1) + \frac{1}{2}\mathbf{Fr}(x = 0-1) = \frac{1}{2^{n+1}}\frac{1}{n+1}\binom{n+1}{\frac{1}{2}((n+1)+1)}[n+1>0]\\
\equiv &\quad \{\mathbf{Fr}(x=-1)=0,\ \text{inductive hypothesis } I(n)\}\\
&\frac{1}{2}\frac{1}{2^n}\frac{1}{n+1}\binom{n+1}{\frac{1}{2}(n+2)} = \frac{1}{2^{n+1}}\frac{1}{n+1}\binom{n+1}{\frac{1}{2}((n+1)+1)}\\
\equiv &\ \textbf{True}
\end{aligned} \tag{3.27}$$

Since the postcondition we want is exactly the guard, following the inference rule for the **while** loop **R3**, we can define

$$\begin{aligned}
&s(0) = 0\\
&s(n) = \frac{1}{2^n}\frac{1}{n}\binom{n}{\frac{1}{2}(n+1)}[n+1 \equiv 0 \mod 2] \quad \text{for } n \geq 1,
\end{aligned} \tag{3.28}$$

clearly the probability of termination at the nth iteration implied by the invariant.

Lastly to get the post-expectation we need to sum up all the frequencies:

$$
\begin{aligned}
&\sum_{n=1}^{\infty} \frac{1}{2^n}\frac{1}{n}\binom{n}{\frac{1}{2}(n+1)}[n+1 \equiv 0 \mod 2] \\
=&\sum_{n=0}^{\infty} \frac{1}{2^{2n+1}}\frac{1}{2n+1}\binom{2n+1}{n+1} \\
=&1
\end{aligned}
\tag{3.29}
$$

which sums up to 1 according to the theory of random walks. Hence the program terminates with probability 1, i.e., $\mathbf{Pr}(x = 0) = 1$.

It probably looks slightly unsatisfactory that we have to quote results from the theory of random walks all the time in this example. However, given that there are so many interesting but difficult problems in random processes and the fact that many of them can be represented as a *pGCL* program, our semantics can only help us to transfer a question about the behaviour of a program in *pGCL* to one in probability theory, and we can only hope that the tools from probability theory can carry us through the remaining analysis.

The loop rule of **Pr**DL requires us to add up all the probabilities of achieving the postcondition at each iteraton. However in this particular case we know that the probability of achieving the negation of the guard $(x = 0)$ at termination must be one if there is a positive probability of termination. The above calculations seem too heavy unless we want to find out the probability of escaping the loop at the nth iteration. We can formalize the above intuition by proving that for all guards G and loop bodies P,

$$\{\mathbf{while}\ G\ \mathbf{do}\ P\}\mathbf{Fr}(true) > 0 \ \Rightarrow\ \mathbf{Pr}(G) = 1 \tag{3.30}$$

are theorems of **Pr**DL. This "meta-theorem" about theorems in **Pr**DL can then be applied directly to obtain the probability of achieving $x = 0$ given termination.

3.4 Summary

We have treated two examples in detail, and in the process we have discovered that the invariant approach in general seems to be a much better way than the *wp* approach for reasoning with loops. We also notice that as we look at more and more complicated examples, the main work in the analysis of *pGCL* programs shifts from the semantics to the manipulation of sums and borrowing results from probability theory.

Chapter 4

On the Simulation of Coins and Dice

4.1 Introduction

In the previous chapters we made frequent use of uniform random choice ($:\in [1..N]$) in our programs. However, this might not be the primitive source of randomness available in many situations. In this chapter we explore the question of simulating different sources of randomness from one another, in particular the question of simulating the uniform random choice $:\in [1..N]$ from the binary probabilistic choice ${}_p\oplus$ with fixed unknown p. In this chapter we use UNIFORM(N) to denote a uniform random choice from the range $[1..N]$ rather than the notation $:\in [1..N]$, which is more suitable with program text.

4.2 Simulating Fair Choice using Biased Choice

4.2.1 Von Neumann's Method

There is a method due to von Neumann that allows us to simulate a fair coin using a biased coin. The algorithm is very simple and clever.

Algorithm B(2)

1. Throw the biased coin twice
2. If the outcome is HT then output H. Else if the outcome is TH output T.
3. If the outcome is HH or TT, then go back to step 1 and repeat the experiment

The algorithm relies on the assumption that the biased coin has a constant probability p of showing a head no matter how many times the experiment is repeated. The events HT and TH have the same probability $(p(1-p))$ of happening. By turning this algorithm into a $pGCL$ program we can simulate $\oplus_{\frac{1}{2}}$ using $\oplus_p$ for any $p \in (0,1)$. Let P,Q be any two programs. Then

$$
\begin{aligned}
P \oplus_{\frac{1}{2}} Q \equiv\ & x, y := 0, 0; \\
& \mathbf{do}\ x = y \rightarrow \\
& \quad x := 0 \oplus_p 1; \\
& \quad y := 0 \oplus_p 1 \\
& \mathbf{od}; \\
& \mathbf{if}\ x = 1\ \ \rightarrow P \\
& \square\ x \neq 1\ \ \rightarrow Q \\
& \mathbf{fi}
\end{aligned}
\tag{4.1}
$$

Remark Note that the program on right hand side can diverge with probability 0 while we would expect the program on the left to always terminate. However, our semantics does not differentiate between these two cases.

To prove the above program correct, we need to show $\mathbf{Pr}(x = 1) = 1/2$. Using the methods we introduced in the last two chapters and drawing experience from the geometric distribution program (Program 2.2), we guess the invariant:

$$
I(n) \equiv \left(\begin{array}{c} \mathbf{Fr}(x = 1, y = 0) = (p^2 + (1-p)^2)^{n-1} p(1-p) \\ \mathbf{Fr}(x = 0, y = 1) = (p^2 + (1-p)^2)^{n-1} p(1-p) \\ \mathbf{Fr}(x = y) = (p^2 + (1-p)^2)^n \end{array} \right)
\tag{4.2}
$$

$I(n)$ can be easily shown to be invariant using our methods, and we can define:

$$
\begin{aligned}
& s(0) = 0 \\
& s(n) = p(1-p)(p^2 + (1-p)^2)^{n-1} \quad \text{for } n \geq 1
\end{aligned}
\tag{4.3}
$$

where $s(n)$ represents $\mathbf{Fr}(x \neq y, x = 1)$ according to the constructions required by the loop rules of $\mathbf{Pr}$DL. We obtain as conclusion:

$$
\begin{aligned}
\mathbf{Fr}(x = 1) &= \sum_{n=0}^{\infty} s(n) \\
&= p(1-p) \sum_{n=1}^{\infty} \big(1 - 2p(1-p)\big)^{n-1} \\
&= p(1-p) \frac{1}{1 - (1 - 2p(1-p))} \\
&= \frac{1}{2}
\end{aligned}
\tag{4.4}
$$

Using similar methods we can show $\mathbf{Fr}(x = 0) = 1/2$, hence $\mathbf{Fr}(\mathbf{true}) = 1$, and we can conclude $\mathbf{Pr}(x = 1) = 1/2$.

Now if we want to find out the expected number of probabilistic choices $\oplus_p$ needed during the run of the program we can add a counter variable k that increments by 2 during each execution of the loop. Then we can adopt very similar reasoning to the geometric distribution in Chapter 2 to get

$$\begin{aligned} & \frac{2}{1 - p^2 - (1-p)^2} \\ = & \frac{1}{p(1-p)} \end{aligned} \tag{4.5}$$

as the expected value of k, and hence the expected number of probabilistic choices $\oplus_p$ needed.

4.2.2 A simple generalization

The key idea in the above algorithm is that the events HT and TH are equiprobable. Now suppose we throw the biased coin three times; we would obtain the following possible outcomes:

Probablity	p^3	$p^2(1-p)$	$p(1-p)^2$	$(1-p)^3$
Events	HHH	HTH	HTT	TTT
		THH	THT	
		HHT	TTH	

Observe that {HTH,HTT}, {THH,THT}, and {HHT,TTH} are three sets of disjoint equiprobable events. This motivates the following algorithm for simulating UNIFORM(3):

Algorithm B(3)

1. Throw the biased coin thrice
2. If the outcome is in the set:
   (a) {HTH,HTT} $\rightarrow$ output 0
   (b) {THH,THT} $\rightarrow$ output 1
   (c) {HHT,TTH} $\rightarrow$ output 2
3. If the outcome is HHH or TTT, go back to step 1 and repeat the experiment

The expected runtime of this algorithm is:

$$
\begin{aligned}
& \frac{3}{1-p^3-(1-p)^3} \\
= & \frac{3}{3p^2(1-p)+3p(1-p)^2} \\
= & \frac{1}{p(1-p)}
\end{aligned}
\tag{4.6}
$$

More interestingly, note that the expected number of random choices needed to choose uniformly between 2 objects and among 3 objects equals $1/(p(1-p))$ using both schemes. On average we do not need any extra coin throws to choose randomly from 3 objects. Isn't that nice?

Algorithm B(3) actually suggests a possible generalization to a family of algorithms for choosing uniformly from $[1..N]$, that is especially efficient when N is prime. This is due to the fact that the binomial coefficient $\binom{N}{k}$ is divisible by N when $0 < k < N$ for prime N. Indeed

$$
\binom{N}{k} = \frac{N!}{k!(N-k)!} \tag{4.7}
$$

The condition $0 < k < N$ implies the denominator is a product of integers less than N, while N prime implies it is not divisible by any of those factors in the denominator. So $\binom{N}{k}$ is divisible by N.

Hence we can throw the biased coin N times, and only have to reject the two sequences which gives all heads or all tails. We can divide the remaining sequences into N equiprobable events, and map each to one of the natural numbers from 0 to $N-1$. For example when $N = 5$, we can divide the sequences as follows:

Probablity	$p(1-p)^4$	$p^2(1-p)^3$	$p^3(1-p)^2$	$p^4(1-p)$
Events	00001	00011	00111	01111
		00101	01011	
	00010	00110	01101	10111
		01001	01110	
	00100	01010	10011	11011
		01100	10101	
	01000	10001	10110	11101
		10010	11001	
	10000	10100	11010	11110
		11000	11100	

The expected runtime of this algorithm is $5/(1-p^5-(1-p)^5)$.

Obviously we can do this to other primes N as well. We record it as another algorithm, which contains as special cases Algorithm B(2) and Algorithm B(3).

Algorithm B(N) (for N prime)

1. Generate N random bits using $\oplus_p$ to give a bit sequence s
2. If the sequence s is not all 0's or all 1's, then let it contain k 1's. There are $\binom{N}{k}$ sequences that contain k 1's. Let $r(s)$ be the rank of s among all the sequences containing k 1's when all such sequences are ordered by the binary number they represent in ascending order (just like the above example for $N = 5$). Output
$$r(s) \textbf{ div } \frac{1}{N}\binom{N}{k} \tag{4.8}$$
3. If the sequence s is constant (all 0's or all 1's) then go back to step 1 and repeat the whole experiment.

Step 2 of the above algorithm is just a deterministic procedure of dividing the sequences into N equiprobable disjoint sets. Certainly there are many other ways of dividing up the sequences.

The expected runtime for this algorithm is
$$\frac{N}{1 - p^N - (1-p)^N} \tag{4.9}$$
(N throws for each experiment with a success probability of $1-p^N-(1-p)^N$ for each experiment.)

Algorithm B(N) can be expressed in $pGCL$ as follows:

```
var i, j, k, b : Int,  s : array [0..N) of Int
j := N;
do j = N →
    {generate the sequence}
    i := 0;
    do i < N →
        s[i] := 0 ⊕_p 1;
        i := i + 1
    od;

    {checks if the sequence is constant}
    j, b := 1, s[0];
    do (j < N) ∧ (s[j] = b) →
        j := j + 1
    od
od;

{count the number of 1's}
k := 0;
j := 0;
do j < N →
    if s[j] = 1 → k := k + 1
    □ s[j] = 0 → skip
    fi;
    j := j + 1
od;

output := rank(s) div (1/N)·C(N,k)
```

The procedure **rank** takes an array of bits as a sequence and returns the equiprobable set to which it belongs (identified by $0, 1, 2, ..N-1$) according to the rule described (alphabetical order) in the description of algorithm $\mathrm{B}(N)$ above. It can be implemented using the "next higher permutation" program discussed in chapter 13 of [1] to count from the lowest permutation with k 1's until the sequence is reached. This implementation is not particularly efficient, and indeed it is an interesting problem to find an efficient procedure to partition the sequences into N disjoint equiprobable sets. But we are not going to concentrate on this problem because our main measure of cost in this chapter is the expected number of probabilistic choices (or coin throws) used in various simulations.

4.2.3 The case for general N

For composite N we can of course imitate the above algorithm to simulate UNIFORM(N). But when N is composite we need to reject sequences other than those which are all 0's or all 1's. For example, for $N = 6$, the binomial coefficients are 1,6,15,20,15,6,1, none of which is divisible by 6 except for the two which represent sequences that contain only one head or one tail. Thus we can imagine such a procedure is not going to be particularly effective. There are at least two different ways to improve this. Stout and Warren[16] suggested taking mod N of the binomial coefficients, resulting in a family of $O(\log N)$ algorithms.

Our method of simulating UNIFORM(N), which is also in $O(\log N)$, is based on the idea of generate and test and uses the algorithms in the previous section. First of all consider the following algorithm:

Algorithm C(2) (Preliminary version)

1. Generate $\lceil \log_2 N \rceil$ random bits using $\oplus_{\frac{1}{2}}$
2. Interpret the resulting bit string s as a binary number x. If $x < N$, then output x.
3. If $x \geq N$, then go back to step 1 and repeat the experiment

Informally, the number of expected invocations of $\oplus_{\frac{1}{2}}$ in Algorithm C(2) is

$$\lceil \log_2 N \rceil \frac{2^{\lceil \log_2 N \rceil}}{N} \tag{4.10}$$

(geometric with success probability $N/2^{\lceil \log_2 N \rceil}$).

If the only source of randomness available is ${}_p\oplus$ (a biased coin), we can then use algorithm B(2) to simulate the fair choice ${}_{\frac{1}{2}}\oplus$ for use in step 1 of the above algorithm.

However there is an important improvement to the above algorithm in step 2. Observe that we do not need to generate all the $\lceil \log_2 N \rceil$ bits to determine whether the number is smaller than N. For example, take $N = 9$, if the first two random bits are 11, then we know that there is no point in wasting time to generate the remaining two bits as $11XX > 1001$, where 1001 is the binary representation of 9. This motivates the following improvement to the algorithm. We use $N_2[i]$ to denote the ith bit in the binary representation of N counting from the most significant bit, and $0 \leq i < \lceil \log_2 N \rceil$.

Algorithm C(2) (Final version)

1. Set i to 0
2. Generate a random bit using $\oplus_{\frac{1}{2}}$ and store it in $s[i]$. Increment i by 1.
3. If $s[i] = (N-1)_2[i]$ and i is less than $\lceil \log_2 N \rceil$ then go back to step 2. If $s[i] > (N-1)_2[i]$ then go back to step 1. Otherwise go to step 4.

4. While $i < \lceil \log_2 N \rceil$, generate a random bit using $\oplus_{\frac{1}{2}}$, store it in $s[i]$ and increment i by 1.

5. Interpret the resulting bit string s as a binary number x. If $x \leq N-1$, then output x.

6. If $x > N-1$, then go back to step 1 and repeat the experiment

The algorithm can be represented as a $pGCL$ program as follows:

$$
\begin{array}{l}
\textbf{var } i, n, x, y : Int;\ b, s : \textbf{array } [0..\lceil \log_2 N \rceil) \textbf{ of } Int \\
i, n, y := 0, \lceil \log_2 N \rceil, N-1; \\
\\
\{\text{store the bit representation of } N-1 \text{ in array b}\} \\
\textbf{do } i < n \rightarrow \\
\quad y, b[n-1-i] := y \textbf{ div } 2, y \textbf{ mod } 2; \\
\quad i := i+1 \\
\textbf{od} \\
\\
i := 0; \\
\textbf{do } i \neq n \rightarrow \\
\quad i := 0; \\
\quad s[i] := 0 {}_{\frac{1}{2}}\!\oplus 1; \\
\quad \{\text{step 2 and 3}\} \\
\quad \textbf{do } s[i] = b[i] \wedge i < n-1 \rightarrow \\
\quad\quad i := i+1; \\
\quad\quad s[i] := 0 {}_{\frac{1}{2}}\!\oplus 1 \\
\quad \textbf{od}; \\
\\
\quad \{\text{step 4}\} \\
\quad \textbf{if } i < n \wedge s[i] < b[i] \rightarrow \\
\quad\quad \textbf{do } i < n-1 \rightarrow \\
\quad\quad\quad i := i+1; \\
\quad\quad\quad s[i] := 0 {}_{\frac{1}{2}}\!\oplus 1 \\
\quad\quad \textbf{od} \\
\quad \Box\ otherwise \rightarrow \textbf{ skip} \\
\quad \textbf{fi} \\
\textbf{od};
\end{array}
$$

```
{convert sequence s into the number it represents}
x, i := 0, n;
do i > 0 →
    i := i − 1;
    x := 2 * x + s[i]
od;

return(x)
```

The expected number of probablistic choices $_{\frac{1}{2}}\oplus$ used in the above program is more difficult to analyse than the simple preliminary version of algorithm C(2). It is not monotone increasing and depends on the exact bit representation of the number $N-1$. As an example let us find out the expected number of probabilistic choices needed to simulate UNIFORM(10). The bit representation of $10-1=9$ is 1001, and according to our algorithm we only have to start over again when the initial segment is 11 or 101. Let X_{10} be the expected number of probabilistic choices needed to simulate UNIFORM(10); we have:

$$E(X_{10}) = \frac{10}{2^4} \cdot 4 + \frac{1}{2^2}(2 + E(X_{10})) + \frac{1}{2^3}(3 + E(X_{10})) \tag{4.11}$$

and hence

$$E(X_{10}) = \frac{27}{5} \tag{4.12}$$

which is smaller than that needed by the preliminary version of algorithm C(2) (which requires $4 \cdot 16/10 = 32/5$).

In general we have

$$E(X_N) = \lceil \log_2 N \rceil + \frac{2^{\lceil \log_2 N \rceil}}{N} \sum_{k=1}^{\lceil \log_2 N \rceil} \frac{k}{2^k} \left[(N-1)_2[k-1] = 0 \right] \tag{4.13}$$

This function is not smooth but can be easily computed using a mathematics package. A plot of the above function for $N = 2$ to 1000 is shown in Figure 4.1. The sudden jump in the value of the expected number of probabilistic choices is caused by moving from 2^n to $2^n + 1$. If N is of the form 2^n, then it would be very efficient to simulate UNIFORM(N) because we do not have to reject any sequence at all: we are guaranteed to stop after exactly n probabilistic choices. On the other hand, UNIFORM($2^n + 1$) is most inefficient as we need to reject almost half of all the sequences generated. For N in the range $2^n < N < 2^{n+1}$, the expected number in general decreases as N increases because the length of the sequence remains the same $(n + 1)$ while the number of sequences need to be rejected decreases. So the expected number is rather like the reactivity of the elements in the periodic table: it decreases as you go across the same period, but increases as you go from one

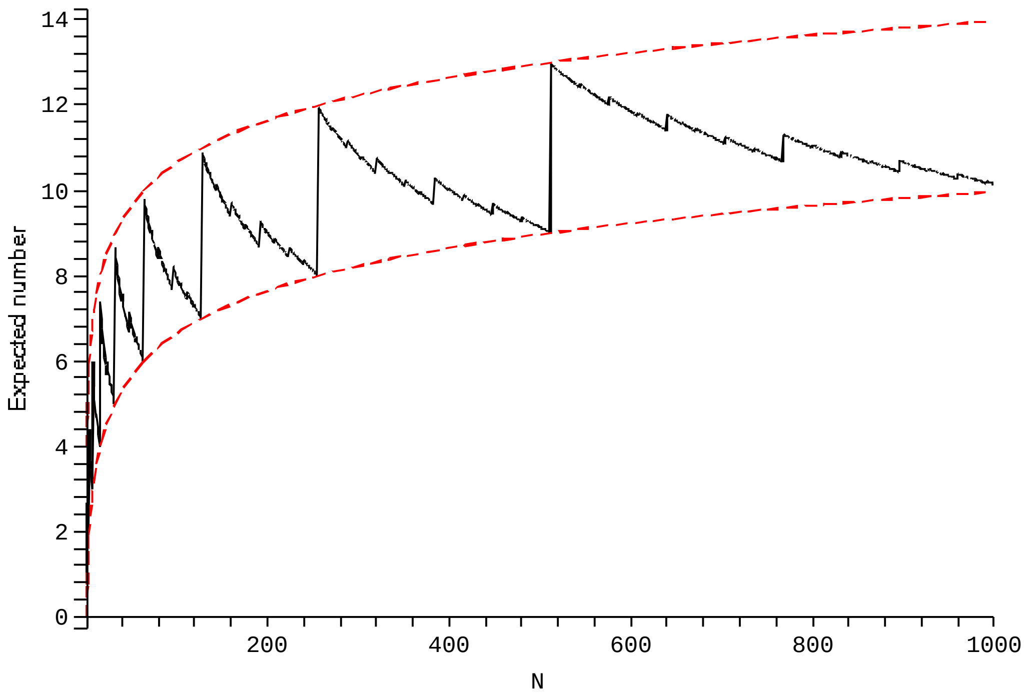

Figure 4.1: Expected number of probabilistic choice versus N (base 2)

period down to the next (except that here we have infinitely many periods with increasing length, and there are some local irregularities as well).

Now we shall investigate (4.13) to obtain some bounds on $E(X_N)$. Certainly

$$\frac{2^{\lceil \log_2 N \rceil}}{N} < 2 \tag{4.14}$$

and

$$\begin{aligned} & \sum_{k=1}^{\lceil \log_2 N \rceil} \frac{k}{2^k} [(N-1)_2[k-1] = 0] \\ < & \{\text{since } (N-1)_2[0] = 1 \text{ always}\} \\ & \sum_{k=2}^{\infty} \frac{k}{2^k} \\ = & \frac{3}{2} \end{aligned} \tag{4.15}$$

Hence we have for all N,

$$\lceil \log_2 N \rceil \leq E(X_N) < \lceil \log_2 N \rceil + 3 \tag{4.16}$$

As a result

$$\begin{aligned} E(X_{N+1}) - E(X_N) &< \lceil \log_2 (N+1) \rceil + 3 - \lceil \log_2 N \rceil \\ &\leq 4 \end{aligned} \tag{4.17}$$

The maximum jump of the expected number of probabilistic choice from $N = 2^n$ to $N = 2^n + 1$ is therefore at most 4. Indeed the value of the jump tends to 4 as n tends to infinity. The two envelopes in figure 4.1 are $\log_2 N$ and $\log_2 N + 4$ respectively.

4.2.4 A further generalization

As before we can generalize algorithm C(2) from base 2 to other bases like 3, 4, 5, etc. This gives a familiy of algorithms C(b). We use $N_b[i]$ to denote the value of the ith leftmost symbol in the base b representation of N, where $0 \leq i < \lceil \log_b N \rceil$.

Algorithm C(b)

1. Set i to 0
2. Generate a random digit using UNIFORM(b) and store it in $s[i]$. Increment i by 1.
3. If $s[i] = (N-1)_b[i]$ and i is less than $\lceil \log_b N \rceil$ then go back to step 2. If $s[i] > (N-1)_b[i]$ then go back to step 1. Otherwise go to step 4.

4. While $i < \lceil \log_b N \rceil$, generate a random digit using UNIFORM(b), store it in $s[i]$ and increment i by 1.

5. Interpret the resulting string s as a base b number x. If $x \leq N-1$, then output x.

6. If $x > N-1$, then go back to step 1 and repeat the experiment

The results we obtained in the last section can be generalized in an obvious way. For example, let $E_b(X_N)$ be the expected number of probabilistic choice using UNIFORM(b) to simulate UNIFORM(N); then

$$E_b(X_N) = \lceil \log_b N \rceil + \frac{b^{\lceil \log_b N \rceil}}{N} \sum_{k=1}^{\lceil \log_b N \rceil} \frac{k}{b^k}(b - 1 - (N-1)_b[k-1]) \quad (4.18)$$

and $E_b(X_N)$ is bounded by:

$$\lceil \log_b N \rceil \leq E_b(X_N) < \lceil \log_b N \rceil + 2 + \frac{1}{b-1} \quad (4.19)$$

Another useful bound on $E_b(X_N)$ is:

$$E_b(X_N) \leq \lceil \log_b N \rceil \frac{b^{\lceil \log_b N \rceil}}{N} \quad (4.20)$$

This bound is generalized from the expected runtime of the preliminary version of algorithm C(2) (equation (4.10)). It is valid because our improved final version is always more efficient then the preliminary version. In some occasions it could be tighter bound then the one in (4.19), especially when N is small.

4.3 The analysis of some specific examples

Now we have gathered some results to investigate the main question of this chapter: given the probabilistic choice operator ${}_p\oplus$ as the only source of randomness, where p is unknown, how can we efficiently simulate UNIFORM(N) for arbitrary N? Alternatively, the question can be rephrased as: given a biased coin with unknown bias p, how can we efficiently simulate an N-sided die?

An obvious algorithm to do this is to combine algorithm B(b) and C(b) together. Use algorithm C(b) to simulate UNIFORM(N), and apply algorithm B(b) in step 2 and 4 of algorithm C(b) whenever a uniform choice in the range $[0..b)$ is needed. The expected number of probabilistic choice ${}_p\oplus$ needed is:

$$\frac{b}{1 - p^b - (1-p)^b} E_b(X_N) \quad (4.21)$$

The question that remains is: what base b should we use for different values of N? Obviously if we use a larger base b $E_b(X_N)$ becomes smaller as $E_b(X_N)$ is in $\Theta(\log_b N)$ (a consequence of (4.19)). However at the same time the first factor $b/(1-p^b-(1-p)^b)$ increases as b increases. So our choice of b would depend on the interaction of these two effects for different values of N. Moreover, the range of p can also affect the choice of b.

We are going to explore the interactions of these factors by considering specific examples. Unfortunately the results we gain from these examples are not conclusive enough to give a guideline to the choice of base b for different values of N, but they illustrate the interactions of the above factors with respect to the efficiency of the simulation.

To facilitate our discussion below, we would like to call the expression for the expected runtime of algorithm B(b) $f_b(p)$, i.e., we define

$$f_b(p) = \frac{b}{1-p^b-(1-p)^b} \tag{4.22}$$

4.3.1 Example 1: Simulating UNIFORM(8)

Let's try to simulate UNIFORM(8). 8 is a power of 2, so base 2 is an obvious thing to try first.

$$\begin{aligned} & \text{Expected runtime using base 2} \\ = & \frac{2}{1-p^2-(1-p)^2}\lceil \log_2 8 \rceil \\ = & \frac{3}{p(1-p)} \end{aligned} \tag{4.23}$$

The above runtime is tight and we haven't used any inequalities because 8 is a power of 2. Now let's try radix 3 and see if we can do better:

$$\begin{aligned} & \text{Expected runtime using base 3} \\ \le & \{\text{upper bound from (4.20)}\} \\ & \frac{3}{1-p^3-(1-p)^3}\lceil \log_3 8 \rceil \frac{9}{8} \\ = & \frac{9}{4}\frac{1}{p(1-p)} \\ < & \frac{3}{p(1-p)} \end{aligned} \tag{4.24}$$

So it is more efficient to use base 3 rather than base 2 to simulate UNIFORM(8) even though 8 is a power of 2, for any $p \in (0,1)$. This is due to fact that $f_2(p)$ is equal to $f_3(p)$. Indeed we can consider the more general case of simulating UNIFORM(N):

$$
\begin{aligned}
& \frac{\text{Expected runtime using base 3}}{\text{Expected runtime using base 2}} \\
= \; & \frac{f_3(p)E_3(X_N)}{f_2(p)E_2(X_N)} \\
= \; & \{f_3(p) = f_2(p) \text{ for all } p \in (0,1)\} \\
& \frac{E_3(X_N)}{E_2(X_N)} \\
< \; & \{\text{upper bounds from (4.19)}\} \\
& \frac{\lceil \log_3 N \rceil + 2 + \frac{1}{2}}{\lceil \log_2 N \rceil} \\
\leq \; & \frac{\log_3 N + \frac{7}{2}}{\log_2 N} \\
= \; & \log_3 2 + \frac{7}{2 \log_2 N} \\
< \; & 1
\end{aligned} \tag{4.25}
$$

for N greater than, say, 1000.

So base 3 is asymptotically a better choice than base 2 for simulating UNIFORM(N).

4.3.2 Example 2: Simulating UNIFORM(31)

We know that when N is prime, we can apply algorithm B(N) directly to simulate UNIFORM(N) without using any of the algorithms C. However, the runtime of algorithm B(N) is linear in N while algorithm C(b) is logarithmic (O($\log_b N$) for base b). Hence it is not a good idea to use algorithm B(N) to simulate UNIFORM(N) directly when N is a large prime. As an example, we would like to investigate the relative efficiency of simulating UNIFORM(31) using base 2 and base 31 respectively:

$$
\begin{aligned}
& \text{Expected runtime using base 2} \\
\leq \; & \frac{1}{p(1-p)} \lceil \log_2 31 \rceil \frac{2^{\lceil \log_2 31 \rceil}}{31} \\
= \; & \frac{160}{31} \frac{1}{p(1-p)}
\end{aligned} \tag{4.26}
$$

$$
\begin{aligned}
& \text{Expected runtime using base 31} \\
= \; & \frac{31}{1 - p^{31} - (1-p)^{31}}
\end{aligned} \tag{4.27}
$$

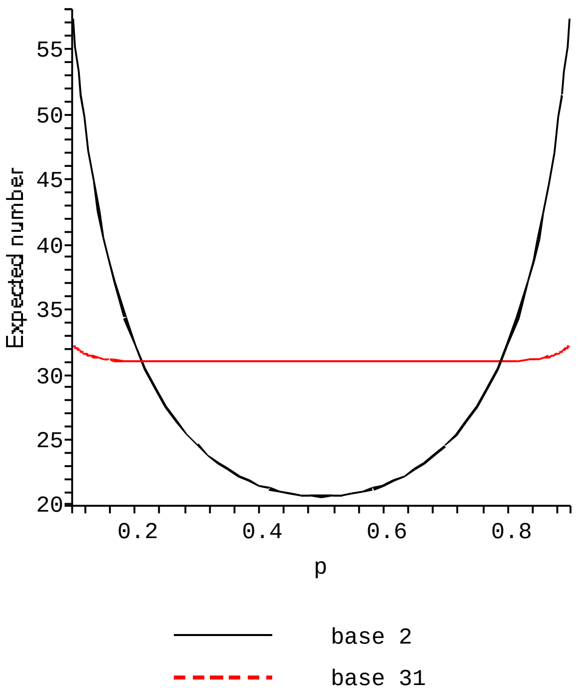

Figure 4.2: Expected number of probabilistic choice versus p

Figure 4.2 shows a plot of the expected runtime of using base 2 and base 31 against different values of p. We notice that base 2 is superior to base 31 for a large range of p. The only exception is the case when p is very close to either 0 or 1, i.e., the coin is very biased. In such a case the power 31 in the denominator of $f_{31}(p)$ helps to keep the expected runtime small.

4.3.3 Example 3: Simulating UNIFORM(5^k)

We have seen in the last example that for a large range of p it is more efficient to use small base b to simulate UNIFORM(N) when N is large. So in the following examples we shall investigate the relative efficiencies of simulating UNIFORM(N) using different small bases.

From example 1 we know that base 3 is asymptotically better than base 2 for simulating UNIFORM(N), but how would base 3 compare with base 5? Instead of investigating the general N directly, we shall look at $N = 5^k$, which should be efficient using a base 5 simulation.

Before continuing we would like to remark on the quotient $f_m(p)/f_n(p)$. It is a bounded rational function defined in the whole range of $p \in [0, 1]$. For $m < n$, we have

$$\frac{f_m(\frac{1}{2})}{f_n(\frac{1}{2})} \leq \frac{f_m(p)}{f_n(p)} \leq 1 \tag{4.28}$$

for all $p \in [0, 1]$. This result is going to be useful in the following analysis.

$$\begin{aligned}
&\frac{\text{Expected runtime using base 3}}{\text{Expected runtime using base 5}} \\
= &\frac{f_3(p)E_3(X_{5^k})}{f_5(p)E_5(X_{5^k})} \\
\geq &\{\text{bound from (4.19)}\} \\
&\frac{f_3(p)}{f_2(p)} \frac{\lceil \log_3 5^k \rceil}{5k} \\
\geq &\{\text{bound from (4.28)}\} \\
&\frac{3}{5} \frac{1 - (\frac{1}{2})^5 - (\frac{1}{2})^5}{1 - (\frac{1}{2})^3 - (\frac{1}{2})^3} \log_3 5 \\
= &\frac{3}{4} \log_3 5 \\
> &1
\end{aligned} \tag{4.29}$$

Hence it is always more efficient to use base 5 to simulate UNIFORM(5^k) than base 3. So base 3 does not dominate base 5 asymptotically. A similar analysis would show that it is always more efficient to use base 5 than base 2 to simulate UNIFORM(5^k).

4.4 Further improvements and discussion of other works

The family of algorithms we analysed in the last section for simulating UNIFORM(N) are by no means optimal and there are many possible improvements. We chose to investigate them because they are general enough (independent of bias p) and at the same time relatively easy to analyse.

We suggest below some possible improvements to the algorithms in the last section:

1. Recall the very first algorithm B(2) by von Neumann in this chapter. Given that we have failed during the first trial, i.e., we get HH or TT, if we fail again on the second trial as well and the sequence we get is HHTT or TTHH, we can terminate the algorithm and return H if we get HHTT and T if we get TTHH. We can do this because the conditional probabilities satifiy:

   $$\mathbf{Pr}(\text{HHTT}|\text{First trial failed}) = \mathbf{Pr}(\text{TTHH}|\text{First trial failed}) = \frac{1}{2}$$

   This improvement can be generalised to B(N) for other N.

   The algorithms can be improved further. Hoeffding and Simons[8] gave an in-depth analysis of such algorithms by treating them as random walks on integral lattice points. They described a family of algorithms, which they called "procedures of type k", for simulating UNIFORM(k). Their idea is to stop at all sequences which contain i heads and j tails such that $\binom{i+j}{i}$ is divisible by k. For example, their procedure of type 2, referred to as Q_2, has an expected runtime of 3.40 at $p = 1/2$ while the original von Neumann algorithm has an expected runtime of 4. The only tradeoff is that for larger k, these algorithms need to keep track of more information about the sequences to determine which value to output.

2. We have remarked that we do not need to generate the remaining bits of a sequence when the prefix indicates the whole sequence is going to be rejected. Similarly, we do not have to generate all the bits if the prefix generated so far is sufficient to ensure the sequence is going to be accepted and to determine which value to output. For example, in the algorithm B(3), we output 0 for the sequences {HTH,HTT}, and 1 for {THH,THT}. This can be improved by outputting 0 when we have HT as the result of the first two throws, and 1 when we have TH, since we know the third throw would not affect the result. We can make such improvements to B(N) in general by some clever arrangement of the equiprobable sequences.

3. Instead of using a simple base we can combine several bases together. For example to simulate UNIFORM(6) we can simulate UNIFORM(3) using algorithm B(3) to get a result $x \in \{0, 1, 2\}$ and simulate UNIFORM(2) using algorithm B(2) to get a result $y \in \{0, 1\}$. Then we can obtain a z uniformly distributed in $\{0, 1, 2, 3, 4, 5\}$ by putting $z = 3y + x$. This method applies to any N. These algorithms can be represented as the tree algorithms of Stout and Warren[16] by, for example, attaching copies of the B(2) trees to the termination nodes of B(3) trees, or the other way round. They cannot be represented as lattice algorithms as their respective set of non-continuation points cannot be combined without destroying the lattice properties.

4. We still have not answered the question of whether there is any optimal algorithm over all values of p in $(0, 1)$. Hoeffding and Simons conjectured that there is no such algorithm for B(2). Stout and Warren proved that no such optimal algorithms exist for the class of tree algorithms.

   However, given any two algorithms A,B for simulating UNIFORM(N), we can obtain an algorithm C by running A and B in parallel on the same sequence of coin tosses and stop when one of them terminates. This is especially useful when we have two algorithms that work best in different ranges of bias p. For example, we can combine the two algorithms using base 2 and base 31 respectively in example 2 in the last section to obtain an algorithm that is better than both of the original algorithms for all values of p. This method of combining algorithms can be expressed easily on a nondeterministic Turing machine, but the resulting algorithm does not lie in the class of tree algorithms, since there are more than two possible moves at each tree node. It is not known whether any optimal algorithm exists in this class, but we suspect the answer is no.

4.5 Summary

We have investigated in this chapter a particular family of algorithms and their efficiencies for simulating a discrete uniform distribution on $[0..N)$ with the probabilistic choice operator ${}_p\oplus$. Hoeffding and Simons[8] have treated the problem of simulating an unbiased coin with a biased coin in depth under the framework of random walk on the non-negative lattice points on the plane. Stout and Warren[16] treated the same problem using another framework which they called "tree algorithms". Elias[2] considered the problem of generating sequences of unbiased output values, using the expected number output values per input value as his measure of efficiency, and described an algorithm that achieves the information theoretical bound when the sequence length tends to infinity.

Bibliography

[1] E.W. Dijkstra *A Discipline of Programming.* Prentice Hall International, 1975.

[2] P. Elias *The Efficient Construction of an Unbiased Random Sequence* The Annals of Mathematical Statistics, 1972, Vol.43, No.3, 865-870.

[3] Y.A. Feldman and D. Harel *A Probabilistic Dynamic Logic.* J. Comput. Syst. Sci., 28:193–215, 1984.

[4] W. Feller *An Introduction to Probability Theory and Its Applications, volume 1.* Wiley 2nd Edition, 1965.

[5] R.L. Graham, D.E. Knuth, O. Patashnik *Concrete Mathematics: A Foundation for Computer Science, 2nd Ed.* Addison-Wesley, 1994.

[6] D. Gries *The Science of Programming.* Springer, 1981.

[7] G.H. Hardy, E.M. Wright *An Introduction to the Theory of Numbers, 5th Ed.* Oxford, 1979.

[8] W. Hoeffding, G. Simons *Unbiased Coin Tossing with a Biased Coin.* The Annals of Mathematical Statistics, 1970, Vol 41, No.2, 341-352.

[9] C. Jones *Probabilistic Nondeterminism.* Monograph ECS-LFCS-90-105, Edinburgh University, 1990.

[10] A. Kaldewaij *Programming: The Derivation of Algorithms.* Prentice Hall International, 1990.

[11] D. Kozen *Semantics of probabilistic programs.* Journal of Computer and System Sciences, 22:325-350, 1981.

[12] A.K. McIver, C.C. Morgan *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer, *To Appear*

[13] C.C. Morgan, A.K. McIver, and K. Seidel *Probabilistic predicate transformer.* ACM Transactions on Programming Languages and Systems, 18(3):325-353, May 1996.

[14] R. Motwani, P. Raghavan *Randomized Algorithms.* Cambridge, 1995.

[15] L.H. Ramshaw *Formalizing the Analysis of Algorithms.* Ph.D. thesis, Stanford University Also technical report, XEROX PARC, 1981.

[16] Q.F. Stout, B. Warren *Tree Algorithms for Unbiased Coin Tossing with a Biased Coin.* The Annals of Probability, Vol 12, No.1, 212-222.