Applied Mathematical Sciences, Vol. 7, 2013, no. 115, 5693 - 5710
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ams.2013.38460

Computing Probability Generating Functions of Coin Flipping and Its Generalizations

Cherng-tiao Perng, Godfred Yamoah, and Abdinur Ali

Department of Mathematics
Norfolk State University
700 Park Avenue, Norfolk, VA 23504, USA
ctperng@nsu.edu, gsyamoah@nsu.edu, amali@nsu.edu

Abstract

A classical probability question asks for the expected waiting time for flipping a coin (fair or not) until a series of consecutive k heads occur. Now instead of k heads, we can ask for the expected waiting time for a prescribed string such as HTHHTT (H for 'head' and T for 'tail'), and furthermore, the following more general setting: replacing coin flipping by taking a letter, one at a time, what is the expected waiting time until a prescribed string (a series of letters) is reached? Here we allow different probabilities for the occurrence of different letters. We give an exposition to this problem by offering an elementary algorithm and implementing it to compute the corresponding probability generating function: we show that there exists a universal program taking as inputs the choice of letters with given probabilities and the prescribed string, and as output, returning the probability generating function for the waiting time. The same method is applied to solve the problem of several competing strings, which asks for the probability (or more generally the probability generating function) of one of the given strings occurring before the remaining strings. In particular, this solves the problem of finding the expectation and variance for the waiting time random variable of the first problem.

Mathematics Subject Classification: 60-04, 65C50, 60C05, 05C05

Keywords: probability generating function, coin tossing, state tree, universal program, Conway's algorithm, the game of Penney Ante

1 Introduction

The interest of the current investigation originated from a question posed by 'hotkarl' in an actuarial science discussion forum ([10]). By paraphrasing, the problem concerns solving the expectation and variance for the process of *runs* ([1]). Hotkarl knew how to compute the expectation but wondered how to find the variance, for which a somewhat tricky solution was proposed by 'Third Eye', who used the method of compound distribution ([10], [5], [14]). In this paper, a method via probability generating function was devised to solve the above particular problem, then this was generalized to solve the general cases, including the generalization to the problem of competing strings. Granted that this problem is classical and its various solutions are known, the exact method proposed in this note does not seem to appear in textbook (we figured out the results independently of the existing solutions, and we only found out later that there was an exposition in [4] similar in flavor). It is our attempt to keep the prerequisite to the minimum (for solutions from more advanced viewpoints, see [6] and [3]). By literature search we learned that this problem has interesting history (for connection with the Penney Ante type problems, we refer to [2] and [8]), and the field of applicability is wide ([12], [13], [17]; see also [15] for a long list of references). We hope the findings and the implementation of our algorithm can be of use, either pedagogically or in the applied fields, such as scan statistics, biology, and actuarial science.

In section 2, we provide an ad hoc method to solve the generating functions for the processes which are *factorizable*. This method is simple and serves as the driving force for solving the problem regarding the general patterns. In section 3, we have a simple and unified approach to the general problem. The method consists of two steps: first we associate a given problem with a state tree with proper indexing. Secondly based on the tree, we write down a system of equations. Solving the equations yields the required probability generating function. The method of section 3 is generalized in section 4 to the situation of two or more competing strings. We have implemented our algorithm in SAGE ([16]) and Maple ([7]). The codes in SAGE are provided in the Appendix.

2 Basic facts and motivating examples

We recall here some basic facts and give two motivating examples. The main result of this paper will be stated and proven in the next section.

Definition 2.1 *Let X be a discrete random variable taking only nonnegative integers. The probability generating function associated to X, written as $f_X(x)$, is defined by*

$$f_X(x) = \sum_{k=0}^{\infty} \mathrm{P}(X = k)x^k.$$

For more information of generating functions, we refer the readers to [18] and [4].

Facts 2.2 *The mean (i.e. expectation) of X defined by*

$$\mathrm{E}(X) = \sum_{k=0}^{\infty} \mathrm{P}(X = k)k$$

can be computed by $\mathrm{E}(X) = f'_X(1)$, and the variance of X defined by

$$\mathrm{Var}(X) := \mathrm{E}((X - \mathrm{E}(X))^2) = \mathrm{E}(X^2) - \mathrm{E}(X)^2$$

can be computed by $\mathrm{Var}(X) = f'_X(1) + f''_X(1) - f'_X(1)^2$. (See for example, p. 266 of [1].)

Example 2.3 *Performing coin flipping, one stops once we get a series of k consecutive heads. Let X be the random variable denoting the number of throws until one stops. What are $\mathrm{E}(X)$ and $\mathrm{Var}(X)$?*

Solution. Here we do the case of fair coin (the argument of unfair coin is similar). Observe that the process can be factored as

$$(T, HT, HHT, \cdots, \overbrace{H \cdots H}^{k-1} T) * \overbrace{H \cdots H}^{k},$$

where $*$ simply means juxtaposition and each element in the parenthesis represents a failed attempt, which may be empty, or repeated as many times as possible, and the order does not matter. Now by multinomial coefficient argument it is straightforward to see that the probability generating function for the waiting time is given by

$$f_X(x) = \sum_{n=0}^{\infty} \left\{ \frac{x}{2} + \left(\frac{x}{2}\right)^2 + \cdots + \left(\frac{x}{2}\right)^k \right\}^n \left(\frac{x}{2}\right)^k,$$

or

$$f_X(x) = \frac{1}{1 - (\frac{x}{2}) - \cdots - (\frac{x}{2})^k} \cdot \left(\frac{x}{2}\right)^k$$

Therefore by a straightforward computation, we have

$$\mathrm{E}(X) = f'_X(1) = 2^{k+1} - 2$$

and

$$\mathrm{Var}(X) = 2^{2k+2} - 2^{k+1} \cdot (2k + 1) - 2.$$

Example 2.4 *Flipping a coin until a pattern of* HHT *first occurs, what is the expected waiting time? What is the variance of the waiting time?*

Solution. For illustrative purpose, we deal only with fair coin here. Clearly in this situation, the process can be factored as

$$(T, HT) * HH * (H) * T,$$

where elements in the parentheses can be empty, or repeated as many times as possible, and are commutative. Therefore the probability generating function is

$$f_X(x) = \frac{1}{1 - \frac{x}{2} - (\frac{x}{2})^2} \cdot \left(\frac{x}{2}\right)^2 \cdot \frac{1}{1 - \frac{x}{2}} \cdot \frac{x}{2},$$

i.e.

$$f_X(x) = \frac{x^3}{(x-2)(x^2+2x-4)}.$$

By a straightforward computation, we found $\mathrm{E}(X) = 8$ and $\mathrm{Var}(X) = 24$.

The above two examples are special in that the process can be factored and hence the probability generating function can be written down by reading off the factorization. Since not every process can be factored, we need a unified approach to deal with the problem. Fortunately we can solve the problem by using analogue of state diagram: we consider several states each of which represents level of success; instead of the usual state diagram with loops, we use state tree with the same states. This observation is simple but crucial to the problem.

3 Unified Approach

Problem 3.1 *Let T be a finite set of letters with a probability distribution on it so that each letter has a positive probability. Let s be a string of length $\ell > 0$ formed by letters from T. Furthermore let X be the random variable of waiting time associated to the process of taking letters, one at a time, and each independently of the other, where one stops when a pattern of s first appear. We are interested in computing the probability generating function of X and in particular,* E(X) *and* Var(X).

Lemma 3.2 *Let $T, X,$ and s be as in Problem 3.1. Let*

$$f_X(x) = \sum_{n=0}^{\infty} P(X = n)x^n$$

be the probability generating function associated to X. Then the power series defining $f_X(x)$ has a radius of convergence greater than 1.

Proof. Let p, $0 < p < 1$, be the probability of obtaining the string s at the first ℓ attempts. First write $f_X(x) = \sum_{n=0}^{\infty} P(X = n)x^n = \sum_{n=0}^{\infty} a_n x^n$, which can be regrouped, depending on equivalent classes of the powers modulo ℓ, as follows

$$\sum_{n=0}^{\infty} a_n x^n = \sum_{m=0}^{\ell-1} \sum_{k=0}^{\infty} a_{\ell k+m} x^{\ell k+m} = \sum_{m=0}^{\ell-1} x^m \sum_{k=0}^{\infty} a_{\ell k+m} x^{\ell k}.$$

To prove the result, it suffices to find a majorant coefficient for $a_{\ell k+m}$, which can be estimated by dividing the qualified words of length $\ell k+m$ into a string of length m, and all the remaining, strings of length ℓ, as follows

$$\overbrace{*\cdots*}^{m}\overbrace{*\cdots*}^{\ell}\cdots\overbrace{*\cdots*}^{\ell}.$$

To estimate the probability we allow the first m letters to be arbitrary. We know that all the strings but the last of length ℓ in the above division cannot contain s, which shows that $1-p$ is an upper bound of the probability of that corresponding segment. Therefore we have

$$a_{\ell k+m} \leq (1-p)^{k-1} \cdot p,$$

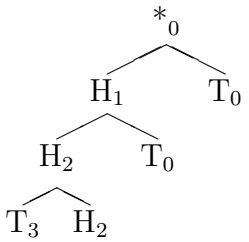
from which it follows that the original series has radius of convergence at least $\frac{1}{\sqrt[\ell]{1-p}} > 1$.

□

Definition 3.3 *A state tree for Problem 3.1 is a tree structure such that the root node is empty letter (denote as $*$ with index 0) and inductively each child node is a letter belonging to the set T with index denoting the progress in achieving the required string s, and it becomes a terminal node if the index already appears in the ancestor nodes. A node with index equal to the length of the string s indicates the least waiting time for the random variable X.*

We illustrate the above definition by working out the state tree for Example 2.4 as follows:

Example 3.4

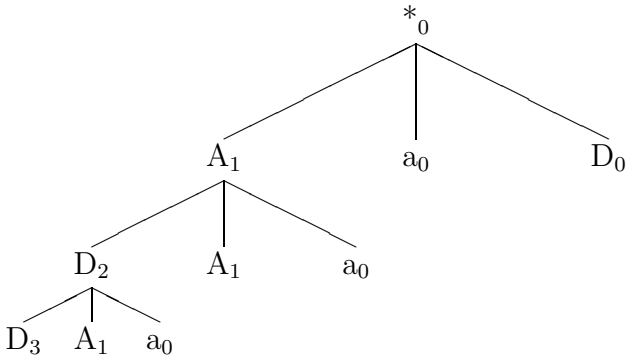


Lemma 3.5 *For Problem 3.1, there exists an algorithm taking the given set T and s as inputs, and returning a state tree similar to the above example as output.*

Proof. For convenience, we let s be represented by $A_1A_2\cdots A_\ell$. Starting with state zero, the root of the tree, descend one step depending on the input letter $A^{(1)} \in T$. If $A^{(1)}$ agrees with the first letter of s, label it as state $(A_1)_1$ which we call state 1, otherwise label it as $(A^{(1)})_0$ and prescribe it as a terminal node. In general, a node with index appearing the second time will be treated as terminal node. Now inductively, assume that for $i \leq k < \ell$, $(A_i)_i$, together with $(A^{(i)})_j$ have been defined. (As will be evident by construction, these nodes $(A^{(i)})_j, j < i$ are the terminal nodes.) Going one step down from $(A_k)_k, k < \ell$, if the input $A^{(k+1)} \in T$ agrees with the $(k+1)$-th letter of s, label it as $(A_{(k+1)})_{k+1}$ which is state $k+1$. For the other possibilities, consider the string t which differs with $s' = A_1\cdots A_kA_{k+1}$ only for the last letter, i.e. $t = A_1\cdots A_kA^{(k+1)}$ with $A^{(k+1)} \neq A_{k+1}$. It is convenient to think of the strings as digital numbers. Initially they are lined up one row below the other such that their starting letters are at the same position. Now perform the shifting procedure: shift t left by one digit, if the overlapped digits of s' and the shifted string agree, then the state of $A^{(k+1)}$ would be labeled as k, otherwise keep shifting left, until there is agreement of the digits in the overlap, or one stops right after s' and the shifted string become disjoint (i.e. no overlapped digits). Then the state of $A^{(k+1)}$ would be labeled as $(k+1)$ minus the number of shifts. Note that the case of shifting until the two strings become disjoint gives the state index of $(k+1)-(k+1) = 0$.

□

Example 3.6 *Let $T = \{A, a, D\}$ and $s = ADD$ in the above lemma. Then the corresponding state tree would be as follows:*



Definition 3.7 *Pertaining to the state tree, for each state $i, 0 \leq i \leq \ell$, we define the conditional probability generating function $f_i(x)$ for the random variable Y_i of extra stopping time conditioned on starting with state i, where the*

stopping status is when a string of s first appears. Clearly $f_0(x) = f_X(x)$*, and* $f_\ell(x) = 1$.

Lemma 3.8 *For a particular state* $i, 0 \leq i \leq \ell - 1$*, with descendants*

$$(A_1)_{j_1}, \cdots, (A_m)_{j_m},$$

where the indices j's mean the states, and we can assume as we did above that the leftmost descendant is of state $i + 1$*, while* $0 \leq j_m \leq i$*, for* $2 \leq i \leq m$*. Here by our notations, we mean* $T = \{A_1, \cdots, A_m\}$ *and for convenience we denote the probability of occurrence of* A_k *by* a_k *for* $1 \leq k \leq m$*. Then we have the following recursive formula for the (conditional) probability generating functions* $f_i(x), f_{j_1}(x) = f_{i+1}(x), \cdots, f_{j_m}(x)$*:*

$$f_i(x) = \sum_{k=1}^{m} (a_k x) f_{j_k}(x).$$

Proof. Regard Y_i as a joint distribution of $Y_{j_k}, 1 \leq k \leq m$ with required transition probability a_k, but with one step shift (hence multiplied by $a_k x$), i.e. the transition from state i to state j_k requires taking one more letter. The formula is clear.

□

Theorem 3.9 *There exists a universal (computer) program taking the inputs T with corresponding probability distribution, and s, which returns the probability generating function of the random variable X defined in the problem. In particular, this allows to compute* E(X) *and* Var(X)*.*

Proof. From Lemma 3.5, we have an algorithm to obtain a state tree with subscripts denoting the states. Due to Lemma 3.8, for each of the non-terminal nodes (i.e. the first appearance of the states 0 up to $\ell - 1$),we can write down the system of equations with unknowns $f_j(x)$: Lemma 3.2 shows that $f_0(x)$ is well-defined function which is analytic within the radius of convergence of the power series; by the relation of the system of equations, the same holds for other $f_j(x)$'s. These are ℓ equations in ℓ unknowns (note that $f_\ell(x) = 1$ is known). For convenience, we keep the indices of $f_j(x)$ as an ordering of variables so that the entries of the corresponding coefficient matrix are described by C_{ij} with $0 \leq i, j \leq \ell - 1$, and we work over the function field of one variable, e.g. $\mathbb{R}(x)$ ($\mathbb{Q}(x)$ suffices if all the assigned probabilities of the letters are rational numbers). We will show that this system of equations has a unique solution over $\mathbb{R}(x)$. For this purpose, observe that the coefficient matrix is the sum of a lower triangular matrix and a matrix whose only nonzero terms are the $(i, i + 1)$-entries. These $(i, i + 1)$-entries are of the form $\pm ax$, where a is the

transition probability from the non-terminal node with index i to the non-terminal note with index $i+1$. Similarly, it is easy to see that the diagonal entries are of the form $\pm(ax-1)$ (with $a \neq 0$) or ± 1, and the remaining entries in the lower triangular matrix are either 0 or of the form $\pm ax$. The determinant of this matrix is obviously a polynomial $f(x) \in \mathbb{R}[x] \subset \mathbb{R}(x)$. To show that the system has a unique solution, if suffices to show that $f(x) \neq 0$. But this is clear, since $f(0) = \pm 1$.

Now the universal program can be obtained by using the algorithm of finding the state tree and the corresponding system of equations, which can be solved by suitable computer algebra system.

□

Corollary 3.10 *The probability generating function associated with the problem is a rational function.*

Proof. This follows from Cramer's Rule.

□

Corollary 3.11 *With assumption as in Theorem 3.9, one can organize the matrix of the coefficient matrix, denoted M (so that $det(M) = f(x)$), in the proof such that it becomes the identity matrix when $x = 0$. Then*

$$f(1) = \mathrm{P}(X = \ell),$$

which is a product of ℓ probabilities associated with the letters of the given string s of length ℓ. In particular, $f(1) \neq 0$.

Proof. First we denote the assigned probabilities for the letters of the string s as $a_1, a_2, \cdots, a_\ell$. Under the stated conditions, the system of equations can be written in matrix form

$$M \begin{bmatrix} f_0(x) \\ \vdots \\ f_{\ell-2}(x) \\ f_{\ell-1}(x) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a_\ell x \end{bmatrix} := B,$$

where the $(i, i+1)$-entries of M are of the form $-a_1 x, \cdots, \cdots, -a_{\ell-1}x$. By Cramer's rule for solving $f_0(x)$, we have

$$f_0(x) = \frac{\det(M_{f_0})}{f(x)},$$

where M_{f_0} is the matrix obtained by replacing the first column of M by B. By direct computation, we see that $\det(M_{f_0}) = (a_1 a_2 \cdots a_\ell)x^\ell$ and we have the relation $f(x)f_0(x) = (a_1 a_2 \cdots a_\ell)x^\ell$. Setting $x = 1$ yields

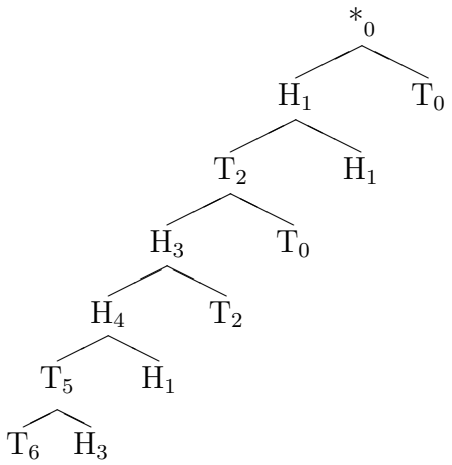
$$f(1) = a_0 a_2 \cdots a_\ell = P(X = \ell),$$

since $f_0(1) = f_X(1) = 1$.

□

Example 3.12 *Let $T = \{H, T\}$ with probability of occurrence p for H and q for T such that $p + q = 1$. What is the probability generating function for the waiting time associated to the string $HTHHTT$?*

Solution.



With a computer program accepting the string "HTHHTT", one first computes the information"010213" which is the state tree, and then the system of equations in unknowns $f_i(x), 0 \leq i \leq 5$.

$f_0(x) = q * x * f_0(x) + p * x * f_1(x)$
$f_1(x) = p * x * f_1(x) + q * x * f_2(x)$
$f_2(x) = q * x * f_0(x) + p * x * f_3(x)$
$f_3(x) = q * x * f_2(x) + p * x * f_4(x)$
$f_4(x) = p * x * f_1(x) + q * x * f_5(x)$
$f_5(x) = p * x * f_3(x) + q * x$

Finally solving these recursive formulas (with the same program), one gets

$$f_0(x) = \frac{q^3 x^6 p^3}{1 - px - qx + p^3 x^6 q^3},$$

which under the assumption that $p = q = \frac{1}{2}$ yields

$$f_X(x) = \frac{x^6}{64(1 - x + \frac{1}{64}x^6)}.$$

It is important to note that we aim not just to compute a single example. In fact, we can compute **all cases** using a **single** program!

It seems fitting to explain the relation of our generating function with that defined by Odlyzko. Following Odlyzko, let's define first the *correlation* of two strings X and Y over some given alphabet, which was first mentioned and called *leading number* by Conway ([2]).

Definition 3.13 *Let A and B be two words over some alphabet, possibly of different lengths. The correlation of A and B, denoted AB, is a binary string of the same length as A such that the i-th bit (from the left) of AB is determined by placing B under A so that the leftmost character of B is under the i-th character (from the left) of A, and checking whether the characters in the overlapping segments of A and B are identical. If they are identical, the i-th bit of AB is set equal to 1, and otherwise it is set equal to 0.*

Example 3.14 *If $A = abcab$ and $B = bcabbc$, then $AB = 100100$, as depicted below.*

A :	a	b	c	a	b					
B :	b	c	a	b	b	c				0
		b	c	a	b	b	c			1
			b	c	a	b	b	c		0
				b	c	a	b	b	c	0
					b	c	a	b	b	c	1

Note that in the above example, we have $AB = 01001$, $BA = 000000$ (so $AB \neq BA$ in general), $AA = 10010$, and $BB = 100010$.

It is useful in application to record $AB = a_n a_{n-1} \cdots a_0$ ($a_i = 0$ or 1 for $0 \leq i \leq n$) as a polynomial in z and write $AB_z := a_n z^n + a_{n-1} z^{n-1} + \cdots + a_0$, where $a_i = 0$ or 1. Then with inputs as in Example 3.14, we have

$$AB_z = z^3 + 1, BA_z = 0, AA_z = z^4 + z$$

and

$$BB_z = z^5 + z.$$

Similarly in the above examples, we can evaluate z at a given number, for example, $AB_2 = 2^3 + 1 = 9$, etc.

To relate our function and the functions defined by Odlyzko, we consider the random variable X associated with the process of getting a word A made with a finite set Ω of alphabet with cardinality $q \geq 2$. We have the probability function

$$f_X(x) = \sum_{n=0}^{\infty} P(X = n) x^n.$$

In [9], Odlyzko defined a generating function

$$G_A(z) := \sum_{n=0}^{\infty} g_A(n) z^{-n},$$

where $g_A(n)$ is the number of words of length n containing A as suffix (and subword) and no other occurrences of A as subwords (we have slightly changed the notations). By elementary argument, the following result was derived:

$$G_A(z) = \frac{1}{1 + (z - q)AA_z}. \tag{1}$$

In particular, $G_A(z)$ is a rational function in z.

Now if we assume equal probability for each letter in Ω, then it follows easily that our $f_X(x)$ is of the form

$$f_X(x) = \sum_{n=0}^{\infty} \frac{g_A(n)}{q^n} x^n = G_A\left(\frac{q}{x}\right). \tag{2}$$

Proposition 3.15 *Let X be a random variable associated with the word A consisting of alphabet Ω of size $q \geq 2$, where the occurrence of each letter in Ω has equal probability $\frac{1}{q}$. Then $E(X) = qAA_q$.*

Proof. We have by equations (1) and (2)

$$f_X(x) = G_A\left(\frac{q}{x}\right) = \frac{1}{1 + \left(\frac{q}{x} - q\right) AA_{\frac{q}{x}}}.$$

Since $f_X(1) = 1$ and $\mathrm{E}(X) = f'_X(1)$, we have by straightforward computation

$$\mathrm{E}(X) = \frac{d}{dx}\left[\frac{1}{1 + \left(\frac{q}{x} - q\right) AA_{\frac{q}{x}}}\right] |_{x=1} = qAA_q.$$

□

4 Competing Strings

Our method in the above section generalizes easily to the problem of two or more competing strings: given a finite list of strings such that none of them could contain the other strings as subword. We are interested in the problem of solving the probability of a given string occurring before the remaining strings. For simplicity we describe only the case of two competing strings, for which

we have implemented the algorithm in SAGE and Maple.

In what follows we define and explain the case of two competing strings. We omit the details of proof since it is completely analogous to the one given in the previous section.

Definition 4.1 *Let A and B be two nonempty strings consisting of letters from alphabet Ω (we assume that Ω is the union of the set of letters in A and in B). We define $X_{A\dashv B}$ to be the random variable associated to A leading B such that $P(X_{A\dashv B} = n)$ means the probability of A leading B exactly in step n. We let $P(A \dashv B)$ be the probability that A leads B. Furthermore we let*

$$f_{X_{A\dashv B}}(x) := \sum_{n=0}^{\infty} P(X_{A\dashv B} = n)x^n$$

be the probability generating function associated with the random variable $X_{A\dashv B}$.

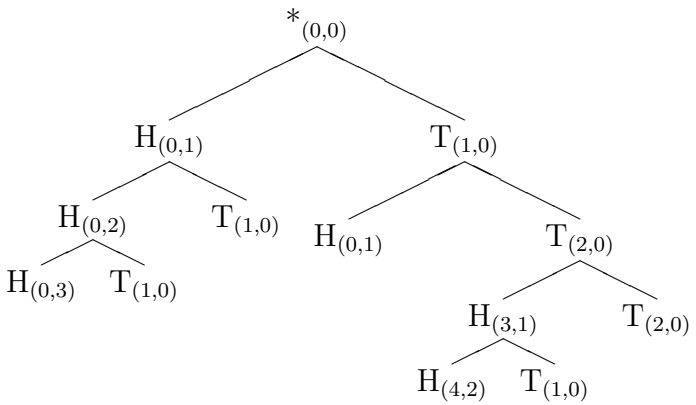
By our definition, clearly we have $P(A \dashv B) = f_{X_{A\dashv B}}(1)$.

Remark. If we define $f_{X_A}(x)$ to be the probability generating function associated with the string A as in previous sections, then clearly $f_{X_A}(x)$ is a majorant series of $f_{X_{A\dashv B}}(x)$. In particular, $f_{X_{A\dashv B}}(x)$ has a radius of convergence greater than 1.

The following example illustrates our implementation for computing $f_{A\dashv B}(x)$, based on analogous algorithm to Lemma 3.5. In the current situation, we need to keep track of indices of the form of ordered pairs, where the first component denotes the success level of the first string and the second component denotes the success level of the second string.

Example 4.2 *$TTHH$ versus HHH*

In this example, we are interested in computing the probability generating function $f_{A\dashv B}(x)$, where $A =$ "TTHH" and $B =$ "HHH". Since we want the probability that A leads B, the indices in the form of ordered pairs are chosen such that the first component reaches 4 before the second component reaches 3. The obvious terminal nodes are of the form $(4, i)$ for some $i < 3$ which corresponds to the probability generating function 1, and $(j, 3)$ for some $j < 4$ which corresponds to the probability generating function 0. Any other node becomes terminal if it appears the second time in the tree. For this example the state tree is as follows.



If we fix an ordering for the non-terminal nodes top-down and from left to right and assign the symbols $f_0, \cdots, f_5$ representing the probability generating functions starting with the given nodes, then we can write down easily the following system of equations. Note that in our program we have used p_0 for probability of "H" and p_1 for probability of "T".

$$
\begin{aligned}
f_0(x) &= p_0 * x * f_1(x) + p_1 * x * f_2(x) \\
f_1(x) &= p_1 * x * f_2(x) + p_0 * x * f_3(x) \\
f_2(x) &= p_0 * x * f_1(x) + p_1 * x * f_4(x) \\
f_3(x) &= p_1 * x * f_2(x) \\
f_4(x) &= p_1 * x * f_4(x) + p_0 * x * f_5(x) \\
f_5(x) &= p_1 * x * f_2(x) + p_0 * x
\end{aligned}
$$

Equivalently in matrix form, we have

$$
\begin{bmatrix}
1 & -p_0x & -p_1x & 0 & 0 & 0 \\
0 & 1 & -p_1x & -p_0x & 0 & 0 \\
0 & -p_0x & 1 & 0 & -p_1x & 0 \\
0 & 0 & -p_1x & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1-p_1x & -p_0x \\
0 & 0 & -p_1x & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
f_0(x) \\ f_1(x) \\ f_2(x) \\ f_3(x) \\ f_4(x) \\ f_5(x)
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ p_0x
\end{bmatrix},
$$

which can be solved in a similar manner as in the proof of Theorem 3.9, noting that by the remark following Definition 4.1 $f_0(x)$ has a radius of convergence greater than 1.

We have $f_{X_{A\neg B}}(x) = \frac{p_0^4p_1^2x^6+p_0^3p_1^2x^5+p_0^2p_1^2x^4}{p_0^2p_1^2x^4-p_0^2p_1x^3-p_0p_1x^2-p_1x+1}$, which is computed by our function "two_party_p_function" that we have implemented.

Now just for fun, we assume that $p_0 = p_1 = \frac{1}{2}$, and X_A, X_B are random variable associated with the strings A and B as in Definition 2.1. The cases of $f_{X_A}(x)$

and $f_{X_B}(x)$ can be computed by our function "general_p_function".

We have the following results:

$$f_{X_A}(x) = \frac{x^4}{x^4 - 16x + 16}, f_{X_B}(x) = \frac{-x^3}{x^3 + 2x^2 + 4x - 8}$$

$$E(X_A) = f'_{X_A}(1) = 16, E(X_B) = f'_{X_B}(1) = 14$$

$$f_{X_{A\dashv B}}(1) = \frac{7}{12} \qquad \text{and}$$

$$(BB_2 - BA_2) : (AA_2 - AB_2) = (7 - 0) : (8 - 3) = 7 : 5,$$

where the computation shows that the probability that A leads B, namely $\frac{7}{12}$, does confirm the prediction by Conway's algorithm for computing the odds that A leads B, namely $7 : (12 - 7) = 7 : 5$. This example was a problem introduced by David L. Silverman, referred to as *Penney paradox*: $TTHH$ has a waiting time of 16 and HHH has a waiting time of 14. Which of these strings is most likely to appear first and with what probability (see [2],[11] and [8] for recent development)?

To quench the curiosity why Conway's algorithm is true, we offer here a concise explanation, by relating our function with the function defined by Odlyzko. The result we use from Odlyzko is not very complicated, and we refer it to his paper.

Pertaining to the case of two competing strings A and B, we adopt the following definition of Odlyzko ([9])

$$H_A(z) = \sum_{n=0}^{\infty} h_A(n) z^{-n}, \tag{3}$$

where $h_A(n)$ is the number of strings over Ω of length n that end with A but contain no other subwords of A and B. It was shown that

$$H_A(z) = \frac{BB_z - BA_z}{(z - q)(AA_z \cdot BB_z - AB_z \cdot BA_z) + AA_z + BB_z - AB_z - BA_z} \tag{4}$$

(see formula (5.2) of [9] where $q = 2$ was used).

Assuming equal probability of occurrence of letters in Ω, it is clear by (3) and (4) that the probability of A leading B is

$$H_A(q) = \frac{BB_q - BA_q}{AA_q + BB_q - AB_q - BA_q}.$$

By symmetry, we have that the probability of B leading A is

$$H_B(q) = \frac{AA_q - AB_q}{AA_q + BB_q - AB_q - BA_q}.$$

The above argument shows that the odds that B leads A is

$$AA_q - AB_q : BB_q - BA_q, \tag{5}$$

a result obtained by Conway by a much more complicated method (see (5.3) of [9], where $q = 2$ was used).

Now to relate $H_A(z)$ with our probability generating function, note that by Definition 4.1 and (3), we have

$$f_{X_{A\neg B}}\left(\frac{q}{z}\right) = H_A(z),$$

where $f_{X_{A\neg B}}(1) = H_A(q)$ is the probability of A leading B. Hence under the assumption of equal probability for alphabet Ω, one has

$$f_{X_{A\neg B}}(x) = H_A\left(\frac{q}{x}\right). \tag{6}$$

Example 4.3 *Let $\Omega = \{a, b, c\}, A = abcab$, and $B = bcabbc$ as in Example 3.14. Then our program shows that*

$$\begin{aligned} f_{X_{A\neg B}}(x) &= \text{two\_party\_p\_function}('\text{abc}','\text{abcab}','\text{bcabbc}') \\ &= \frac{-(p_0^3p_1^4p_2^2x^9 + p_0^2p_1^2p_2x^5)}{D(p_0, p_1, p_2, x)}, \end{aligned}$$

where the denominator $D(p_0, p_1, p_2, x)$ equals

$$\begin{aligned} &p_0^3p_1^4p_2^3x^{10} - p_0p_1^3p_2^2x^6 - (p_0^3p_1^4p_2^2 + p_0^2p_1^4p_2^3)x^9 + (p_0^2p_1^3p_2^3 + (p_0^3p_1^3 + p_0^2p_1^4)p_2^2)x^8 \\ &+(p_0p_1^3p_2 + p_0p_1^2p_2^2)x^5 - p_0p_1p_2x^3 + (p_0^2p_1p_2 + p_0p_1p_2^2)x^4 + (p_0 + p_1 + p_2)x - 1. \end{aligned}$$

Assuming $p_0 = p_1 = p_2 = \frac{1}{3}$, this leads to

$$f_{X_{A\neg B}}(x) = \frac{-3(x^9 + 81x^5)}{x^{10} - 6x^9 + 27x^8 - 81x^6 + 486x^5 + 1458x^4 - 2187x^3 + 59049x - 59049}.$$

Note that $f_{X_{A\neg B}}(1) = \frac{123}{151}$ is the probability that A leads B and it is straightforward to check that

$$f_{X_{A\neg B}}(x) = H_A\left(\frac{3}{x}\right) \text{ (in agreement with (6) for } q = 3)$$

and the odds that A leads B equals $123 : (151 - 123) = 123 : 28 = 246 : 56 = (BB_3 - BA_3) : (AA_3 - AB_3)$, in agreement with (5).

Acknowledgements

Part of this work had been presented by the first author at an MAA sectional meeting in 2009; he thanks the organizer for inclusion of the talk. We thank Dr. Pogany for pointing out the references [6] and [12].

References

[1] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd edition, John Wiley & Sons, Inc., 1968.

[2] M. Gardner, *Time Travel And Other Mathematical Bewilderments,* W.H. Freeman and Company, New York, 1988

[3] H. Gerber, and S.-Y. Li, The Occurrence of Sequence Patterns in Repeated Experiments and Hitting Times in A Markov Chain, *Stochastic Process and their Applications* 11 (1981) 101-108, North-Holland Publishing Company

[4] R.L. Graham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics*, Second Edition, Addison-Wesley Publishing Company, 1994

[5] S.A. Klugman, H.H. Panjer, and G.E. Willmot, *Loss Models*, Second Edition, John Wiley & Sons, Inc., 2004.

[6] S.-Y. Li, A Martingale Approach to the Study of Occurrence of Sequence Patterns in Repeated Experiments, *The Annals of Probability* (1980) Vol. 8, No. 6, 1171-1176

[7] Maple 13, Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.

[8] Y. Nishiyama, Pattern Matching Probabilities and Paradoxes as a New Variation on Penney's Coin Game, *International Journal of Pure and Applied Mathematics,* Vol. 59 No. 3, 2010, 357-366.

[9] A. M. Odlyzko, Enumeration of strings, pp. 205-228 in *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil (eds.), Springer, 1985.

[10] http://www.actuarialoutpost.com/actuarial_discussion_forum/archive/index.php/t-113470.html

[11] W. Penney, A Problem Given in Journal of Recreational Mathematics, October 1969, p.241.

[12] V. Pozdnyakov, J. Glaz, M. Kulldorff, and J. Steele, A Martingale Approach to Scan Statistics, *Ann. Inst. Statist. Math.* Vol. 57, No. 1, 21-37 (2005)

[13] V. Pozdnyakov and M. Kulldorff, Waiting Times for Patterns and a Method of Gambling Teams, *The American Mathematical Monthly,* Vol. 113, No. 2 (Feb., 2006), 134-143

[14] S.D. Promislow, *Fundamentals of Actuarial Mathematics*, John Wiley & Sons, Ltd, 2006

[15] V.T. Stefanov and W. Szpankowski, Waiting Time Distributions for Pattern Occurrence in a Constrained Sequence Discrete Mathematics and Theoretical Computer Science, 9(1), 305-320.

[16] William A. Stein et al. Sage Mathematics Software (Version 4.6), The Sage Development Team, 2013, http://www.sagemath.org.

[17] M. Uchida, On Generating Functions of Waiting Time Problems for Sequence Patterns of Discrete Random Variables, *Ann. Inst. Statist. Math.* Vol. 50, No. 4, 655-671 (1998)

[18] H. S. Wilf, *Generatingfunctionology*, Academic Press, 2nd edition, 1994

Received: August 19, 2013

Appendix

See next page for the codes in SAGE (tested on Version 4.6 and Version 5.4) for computing probability generating functions.

```
def general_p_function(Alphabet, Word): # for example, Alphabet = 'HT', Word = 'HTHHTT'
    A = list(Alphabet)
    W = list(Word)
    p = list(var('p_%d' %i) for i in range(len(A)))
    f = list(var('f_%d' %i) for i in range(len(W)))
    L = [ ]
    for i in range(0, len(W)):
        M = 0
        for j in range(0, len(A)):
            if general_tree_index(Word, i, A[j]) < len(W):
                M = M+p[j]*x*f[general_tree_index(Word, i, A[j])]
            else:
                M = M+p[j]*x
        L = L + [f[i]==M]
    s = solve(L,f)
    return s[0][0].right()

def general_tree_index(Word, i, char):
    L = list(Word)
    if i==0:
        if L[0]==char:
            return 1
        else:
            return 0
    K = L[0:i+1]
    if K[i]==char:
        return i+1
    else:
        K[i]=char
        j = 0
        while(j<i+1):
            if L[0:i+1-j]==K[j:i+1]:
                break
            j=j+1
        return len(K[j:i+1])

def two_party_p_function(Alphabet, Word1, Word2): # for example, Alphabet = 'abc', Word1 = 'abcab', Word2 = 'bcabbc'
    A = list(Alphabet)
    W1 = list(Word1)
    W2 = list(Word2)
    V = variable_index(Alphabet, Word1, Word2)
    p = list(var('p_%d' %i) for i in range(len(A)))
    f = list(var('f_%d' %i) for i in range(len(V)))
    L = [ ]
    for i in range(0, len(V)):
        M = V[i]
        N = 0
        for j in range(0, len(A)):
            k = general_tree_index(Word1, M_x(M),A[j])
            l = general_tree_index(Word2, M_y(M),A[j])
            if l < len(Word2):
                if k==len(Word1):
                    N = N+p[j]*x
                else:
                    for m in range(0, len(V)):
                        if (k,l)==V[m]:
                            N = N+p[j]*x*f[m]
                            break
        L = L+[f[i]==N]
    s = solve(L,f)
    return s[0][0].right()

def variable_index(Alphabet, Word1, Word2):
    A = list(Alphabet)
    W1 = list(Word1)
    W2 = list(Word2)
    L = [(0,0)]
    M = L
    while(len(M)!=0):
        N = [ ]
        for i in range(0, len(M)):
            for j in range(0, len(A)):
                k = general_tree_index(Word1, M_x(M[i]),A[j])
                l = general_tree_index(Word2, M_y(M[i]),A[j])
                if k < len(Word1) and l < len(Word2) and (k,l) not in Set(L):
                    N = N + [(k,l)]
        L = L + N
        M = N
    return L

def M_x((x,y)):
    return x

def M_y((x,y)):
    return y
```