CONTINUAL PATTERN REPLICATION

by

JAMES IAN MUNRO

B. A., University of New Brunswick, 1968

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Computer Science

We accept this thesis as conforming to the

required standard.

The University of British Columbia

August, 1969

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree tha permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver 8, Canada

Date September, 1969

ABSTRACT

This thesis continues the studies of A. Waksman (1969) in the repeated generation of finite strings in a one dimensional array of finite automata. Waksman handles thisppproblem by the use of a "modulo arithmetic" algorithm. This is shown to be very restrictive with regard to the number of characters permitted in the output string. In fact, it is shown that unless the length of the string which is to be repeated is of the form p^{α}, where p is prime, only one output character is permitted. This of course makes the process quite meaningless.

For this reason, a new algorithm is developed. This is referred to as the wheel algorithm, since there is an obvious analogy between it and a wheel, with the output string on its circumference rolling alongathe array and leaving the imprint of the characters in the string behind it in the same way that a wheel leaves tire tracks. The number of states required for such an algorithm is large and so the binary wheel algorithm is introduced. By using this algorithm, in which an output state is represented as a string of bits in several cells, the number of states required, in addition to the n output states, can be reduced to about $4 \log_2 n$.

Both the wheel and binary wheel algorithms are then extended to the two dimensional and finally the d-dimensional cases.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I wish to thank Dr. R. S. Rosenberg for the guidance and assistance which he extended to me in the preparation of this thesis and Dr. A. Mowshowitz for his helpful remarks in preparing the final manuscript.

The financial assistance of the National Research Council of Canada is also gratefully acknowledged.

I INTRODUCTION

In a recent paper, Waksman (1969) uses a one-dimensional array of finite state machines to model the continual replication of a sequence of k symbols represented by the states of the cells of the array. By continual replication of a sequence of length k we mean that after an appropriate length of time, dependent on m, the subsets of cells $\{ik+1, ik+2,\ldots(i+1)k\}$ for $i=1,2,\ldots,m$ will each be replicas of the desired sequence of k symbols. Thus if our alphabet were {0,1} and we wanted to reproduce the string of length 4 (0111) continually, after an appropriate length of time we would have

{0111011101110111...}

The principal restriction on the transformation to be performed is that the state of any cell at time (t+1) is a function of the states of that cell and its two immediate neighbours at time t. Waksman shows a method of generating continually strings of length k in the previously described manner provided the characters from which the string is constructed are chosen from the ring of integers mod g_k, $\mathbb{Z}g_k$; where

$$g_k = \gcd\{\binom{k}{i}\} \qquad i=1,2,\ldots,k-1. \qquad (1.1)$$

That is, g_k is the greatest common division of the set of binomial coefficients $\binom{k}{i}$ other than those which are 1.

The problem is formulated by Waksman as follows:
Suppose we want to generate continually the string $\{a_i\}$ $i=1,2,\ldots,k$; then, we start with the initial configuration consisting of cell 0 in a transition state, P, the next k cells in states b_i $i=1,2,\ldots,k$ and all other cells in the quiescent state, Q. It can be seen that the relation

between the strings a_i and b_i is given by

$$a_i = \sum_{j=1}^{i} \binom{i-1}{j-1} b_j \qquad (1.2)$$

The actual method of calculating $\{b_i\}$ given $\{a_i\}$ will be explained later. It is to be understood that all arithmetic will be modulo g_k.

At each time step one and only one cell is in the transition state, P. At time t, cell t only will be in state P. The process for any cell is basically broken into two segments, the time before it enters state P and the time after it enters state P. At each time step the cell in state P and its right hand neighbour exchange states. After this has happened to a cell, it retains this state. The interesting part of the process is, then, what happens before a cell enters the transition state. If cell i is in state $a_{t,i}$ at time t, with the initial condition $a_{0,i}=b_i$ for $i=1,2,\ldots,k$; we define $a_{t+1,i}$ by

$$a_{t+1,i} = a_{t,i} + a_{t,i-1} , \qquad (1.3)$$

where a cell in the quiescent state is considered to be in state 0.

To redefine the function in a more formal manner we may follow Table 1 which indicates the transformation up to the time of entry into the terminal state. To rigorously define the function we may think of the cells as having an additional flip flop which is off until the right neighbour of a cell enters state P. At that time the flip flop is turned on and no further change can occur in the state of the cell.

TABLE 1*

Flip flop in position 0
Q = quiescent state
P = transition state
Φ = no cell present (left neighbour of cell 0)
a_i = an element of $\mathbb{Z}g_k$

* The output given in this and all similar tables is at time t+1.

cell state at time t		Right neighbour a_i	Right neighbour Q	Right neighbour a_j
Left neighbour	a_k	a_i+a_k		a_i+a_k
	P	P	P	P

Left neighbour	Q	Right neighbour Q
	Q	Q
	a_k	a_k

Left neighbour	P	Right neighbour a_j
	Φ	a_j
	a_k	a_j

No other configurations can occur. Thus if we initialize the array as previously mentioned and follow the transition of Table 1, cell 0 will enter the terminal state a_1, cell 1, $b_1+b_2 = a_2$ and in general cell i-1 enters terminal state

$$\sum_{j=1}^{i} \binom{i-1}{j-1} b_j = a_i \qquad \text{(by equation 1.2)}$$

at time t=i. Our problem is now to calculate $\{b_i\}$ for a given set $\{a_i\}$. Waksman found that $\{b_i\}$ could be generated from $\{a_i\}$ in the following manner. Let $a_{1i}=a_i$ $\quad i=1,2,\ldots,k$ and write $a_{11}\ a_{12} \ldots a_{1k}$. Then

complete a k x k matrix by letting

$$a_{mj} = a_{m-1,j} + a_{m-1,j+1} \qquad \begin{matrix} m=2,3,\ldots,k \\ j=1,2,\ldots,k-1 \end{matrix}$$

and $$a_{mk} = a_{1k} = a_k \qquad (1.4)$$

The arithmetic is of course mod g_k. Waksman shows that the process is continued for m = k+1, k+2,..., and that the matrix is repeated, or that $a_{m+\alpha k,j} = a_{m,j}$ α = any tue positive integer. This property is due to the fact that the arithmetic is done modulo g_k. This process is essentially what happens in the actual generating machine before the P state is entered. The only difference is that the k-tuple remains stationary instead of moving to the right.

After the k x k matrix has been formed the first column will be called the first transposition column. This column is taken as an initializing row for a second matrix formed in the same manner, and hence the first column of this matrix is called the second transposition column. Waksman shows that if g_k such matrices are formed the g_kth transposition column will be the set $\{a_i\}$. Therefore the first row of the g_kth matrix, or the (g_k-1)st transposition column generates $\{a_i\}$ in a manner corresponding to that of the generating function. Hence the (g_k-1)st transposition column may be taken as $\{b_i\}$.

A numerical example may make this process somewhat clearer. Suppose we are to generate {021} continually, so k=3 and g_k=3. Then write

0 2 1 and follow formula (1.4) 3-1=2 times

(1) 0 2 1

2 0 1 — 0 2 2 is the first transposition column

2 1 1

(2) 0 2 2

0 2 0 is the second transposition column and so 0,2,0 = b_i

2 1 2

0 0 2

To check that {0,2,0} will indeed produce {0,2,1} we shall form the g_k or third matrix.

(3) 0 2 0

2 2 0

1 2 0

We see then, that the desired sequence {0,2,1} is produced. To illustrate the previously mentioned fact that if this process is continued the entire matrix will be repeated, and so the g_kth transposition column will be repeated we shall continue a few more time steps.

⋮

1 2 0

0 2 0

2 2 0

1 2 0

0 2 0

2 2 0

1 2 0

0 2 0

2 2 0

⋮

We shall now start the generating function with $\{b_i\}=\{0,2,0\}$

TABLE 2

t ↓ / j →	0	1	2	3	4	5	6	7	8	9	10
0	P	0	2	0	Q	Q	-	-	-	-	-
1	0	P	2	2	0	Q	-	-	-	-	-
2	0	2	P	1	2	0	Q	-	-	-	-
3	0	2	1	P	0	2	0	Q	-	-	-
4	0	2	1	0	P	2	2	0	Q	-	-
5	0	2	1	0	2	P	1	2	0	Q	-
6	0	2	1	0	2	1	P	0	2	0	Q
7	0	2	1	0	2	1	0	P	2	2	0

After time step 3 we note that the desired sequence has been generated once and that the "generating bud" is the same as it was when initialized. Thus we see that the process will work.

An important consideration in judging the merit of such a scheme is the number of states required. Waksman requires the following states:

1 Q the quiescent state

1 P the transition state

$2.g_k$ the integers from 0 to (g_k-1) and also a flip flop to indicate whether a cell has entered its terminal state or not.

This gives a total of $2(g_k+1)$ states, that is about twice as many states as output characters. Another method of continual generation of sequences will be introduced in Chapter 4 and at that time it will be

useful to compare the number of states required for the present method and the one introduced at that time.

II A More Direct Method of Determining $\{b_i\}$

Once $\{b_i\}$ has been determined and the generating function has been initialized the generation of a_i's is straight forward. The process occurs as fast as can be expected for such a structure; that is, one new output cell per time step. The generation of $\{b_i\}$ by Waksman's method is, however, very tedious, expecially if g_k is fairly large. Fortunately it turns out that the method is somewhat inefficient and that $\{b_i\}$ may be determined directly, rather than by iterative procedures, from $\{a_i\}$. Consider the following discussion:

Recall equation 1.2

$$a_i = \sum_{j=1}^{i} \binom{i-1}{j-1} b_j \quad .$$

Rewriting this in vector-matrix notation, we have:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \cdot & 1 & & \cdot \\ \cdot & & & \cdot \\ \cdot & \binom{i-1}{j-1} & & \cdot \\ 1 & & & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \qquad (2.1)$$

or

$$a = A\,b$$

We want to express b in terms of a. This may be done by rewriting equation 2.1 to yield

$$b = A^{-1}\,a$$

Our problem is now simply to invert A. We shall show that:

$$A^{-1} = A' = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ -1 & 1 & & & \vdots \\ 1 & & 1 & & \\ \vdots & & & & \\ & \{\binom{i-1}{j-1}(-1)^{i+j}\} & & & 0 \\ (-1)^{k+1} & k(-1)^{k+2} & \cdots & k(-1)^{2k-1} & 1 \end{bmatrix}$$

To express this simply in words, A^{-1} has the same elements as A, however the (k,j)th element has the sign of $(-1)^{i+j}$. A more useful way of writing this relation would be as Theorem 2.1.

Theorem 2.1

$$b_i = \sum_{j=1}^{i} \binom{i-1}{j-1}(-1)^{i+j} a_j .$$

To prove this theorem we need the following lemma:

Lemma 2.11

$$\sum_{m=j}^{i} \binom{i-1}{m-1}\binom{m-1}{j-1}(-1)^m = 0$$

$$= \sum_{m=j}^{i} \frac{(i-1)!}{(m-1)!(i-m)!} \cdot \frac{(m-1)!}{(j-1)!(m-j)!} (-1)^m$$

$$= \sum_{l=j}^{i} \frac{(i-1)!}{(j-1)!(i-j)!} \cdot \frac{(i-j)!}{(i-m)!(m-j)!} (-1)^m$$

$$= \binom{i-1}{j-1} \sum_{m=j}^{i} \binom{i-j}{m-j}(-1)^m \qquad (i-1)-(j-1)=i-j$$

$$(i-j)-(m-j)=i-m$$

let $k = m-j$

$h = i-j$

$$= (-1)^j \binom{i-1}{j-1} \sum_{k=0}^{h} \binom{h}{k}(-1)^k$$

By expanding $(1-1)^h$ we note that

$$\sum_{k=0}^{h} \binom{h}{k}(-1)^k = 0$$

and hence

$$\sum_{m=j}^{i} \binom{i-1}{m-1}\binom{m-1}{j-1}(-1)^1 = 0$$

QED

We are now ready to prove the theorem.

Proof of Theorem 2.1

$$\text{Let } a_{ij} = \begin{cases} \binom{i-1}{j-1} & \text{for } i=1,2,\ldots,k \quad j=1,2,\ldots,i \\ 0 & \text{for } i=1,2,\ldots,k \quad j=i+1,\ldots,k \end{cases}$$

This defines the matrix $A = \{a_{ij}\}$.

Similarly we define

$$a'_{ij} = \begin{cases} \binom{i-1}{j-1}(-1)^{i+j} & i=1,2,\ldots,k \quad j=1,2,\ldots,i \\ 0 & i=1,2,\ldots,k \quad j=i+1,\ldots,k \end{cases}$$

and so define $A' = \{a'_{ij}\}$.

Then from the definition of $\{b_i\}$ in equation 1.2, we have shown that $a = A\,b$.

What we are to prove is that

$$b_i = \sum_{j=1}^{i} \binom{i-1}{j-1} (-1)^{i+j}\, a_j$$

or that $b = A'a$.

That means $A^{-1} = A'$ or that $AA'=I$.

Let $AA' = \{\alpha_{ij}\}$

then $\alpha_{ij} = \sum_{m=1}^{k} a_{im}\, a'_{mj}$.

We shall show that $\alpha_{ii} = 1$ and that $\alpha_{ij} = 0$ if $i \neq j$, by considering the three cases (i) $i < j$

(ii) $i = j$

(iii) $i > j$

(i) $i < j$

A is triangular, therefore $a_{im} = 0$ for $m=i+1,\ldots,k$

A' is triangular, therefore $a'_{mj} = 0$ for $m=1,2,\ldots,j-1$

Therefore $a_{im}\, a'_{mj} = 0$ for $m=1,2,\ldots,(j-1)$

and for $m=(i+1),\ldots,k$

and as $i < j$ $a_{im} a_{mj} = 0$ for $m=1,2,\ldots,k$

Hence $\alpha_{ij} = 0$ for $i < j$.

(ii) $i = j$

Again we have

$a_{im} = 0$ for $m=i+1,\ldots,k$

$a'_{mj} = 0$ for $m=1,2,\ldots(j-1)$

but as $i = j$ we have

$a_{im}\, a'_{mj} = 0 \quad$ for $m \neq i$

therefore $\alpha_{ii} = a_{ii}\, a'_{ii}$

but

$a_{ii} = \binom{i-1}{i-1} = 1$

$a'_{ii} = \binom{i-1}{i-1}(-1)^{2i} = 1$

therefore $\alpha_{ij} = 1 \quad$ for $i = j$

(iii) $i > j$

again $a_{im} = 0 \quad$ for $m=(i+1),\ldots,k$

$a'_{mj} = 0 \quad$ for $m=1,\ldots,(j-1)$

therefore $a_{km}\, a'_{mj} = 0 \quad$ for $m=1,\ldots,j-1$ and $m=(i+1),\ldots,k$

Hence $\alpha_{ij} = \sum_{m=1}^{k} a_{im}\, a'_{mj} = \sum_{m=j}^{i} a_{im}\, a_{mj}$

$$= \sum_{m=j}^{i} \binom{i-1}{m-1}\binom{m-1}{j-1}(-1)^{m+j}$$

$$= (-1)^{j} \sum_{m=j}^{i} \binom{i-1}{m-1}\binom{m-1}{j-1}(-1)^{m}$$

$= 0 \quad$ by lemma 2.11

Therefore $\alpha_{ij} = 0 \quad$ for $i > j$.

Hence $\alpha_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$

So $AA' = I \quad$ or $A' = A^{-1}$

thus $b_i = \sum_{j=1}^{i} \binom{i-1}{j-1}(-1)^{i+j}\, a_j$

QED

Recalling the example at the end of chapter I we shall now recalculate $\{b_i\}$ given $\{a_i\} = \{0,2,1\}$ $k=3$ $g_k=3$

By Theorem 2.1

$$b_i = \sum_{j=1}^{i} \binom{i-1}{j-1}(-1)^{i+j} a_j$$

so

$$b_1 = a_1 = 0$$

$$b_2 = a_2 - a_1 = 2$$

$$b_3 = a_3 - 2a_2 + a_1 = 1 - 2(2) + 0 = 0 \qquad (\text{mod } 3)$$

Hence we have $\{b_i\} = \{0,2,0\}$ which agrees with the value calculated by Waksman's method. It is easy to see that when k is fairly large Waksman's method involves a great deal of calculation -- $(g_k-1)k(k-1)$ additions. The method which we have just developed is much more direct, requiring less than $\sum_{i=1}^{k} 2(i-1)$ arithmetic operations. That is less than $2 \cdot \frac{0+k-1}{2} \cdot k = k(k-1)$ arithmetic operations. The method is better by a factor g_k. For the entire process to have any meaning g_k must be at least 2.

III The Value of g_k

g_k is essentially the number of characters permitted in the alphabet over which the string is generated by the Waksman technique, since there are g_k elements in $\mathbb{Z}g_k$.

Waksman says nothing more about the size of the alphabet, from which the characters to be generated may be chosen, other than to define g_k as $g.c.d\{\binom{k}{i}\}$ $i=1,2,\ldots,k-1$. As it turns out, upon closer

investigation, g_k is 1 unless k is a prime or a power of a prime. To put it even more simply, g_k is in general 1 and so the Waksman technique of generating sequences continually is meaningless except in special cases of k. For to generate a string of any length with only one character continually is merely to generate this one character continually.

To prove the result we have just stated we must first prove several lemmas.

First, define $N_p(x)$ as the number of times the prime p is a factor of x. Hence $N_2(12)=2$, $N_5(17)=0$ et cetera, It is quite clear $N_p(xq)=N_p(x)+N_p(y)$ when $x\neq 0$, $y\neq 0$.

Lemma 3.11

$$N_p(i+kp^\beta)=N_p(i) \quad \text{for } i=1,2,\ldots,(p^\beta-1)$$

or if $i=p^\beta$ and $p \nmid (k+1)$.

Proof

Let $i = mp^\alpha$ $p \nmid m$ then $0\leq\alpha<\beta$

so $N_p(i+kp^\beta) = N_p(p^\alpha(m+kp^{\beta-\alpha})) = \alpha+N_p(m+kp^{\beta-\alpha})$

But as $p \nmid m$, $p \nmid (m+kp^{\beta-\alpha})$

then $N_p(i+kp^\beta) = \alpha = N_p(i)$ for $i=1,2,\ldots,(p^\beta-1)$

If on the other hand, $i=p^\beta$ and $p \nmid (k+1)$ we have

$$N_p(i+kp^\beta) = N_p((k+1)p^\beta) = \beta + N_p(k+1)$$

But, as

$$p \nmid (k+1)$$

$$N_p(i+kp^\beta) = \beta = N_p(i)$$

QED

Lemma 3.12

$$N_p(p^{\alpha}!) = \sum_{j=1}^{\alpha} p^{j-1}$$

Proof:

We shall prove this lemma by induction on α. First, it is obviously true for $\alpha=0$ and for $\alpha=1$. We shall assume the lemma is true for $\alpha=\beta$ and prove it for $\alpha=\beta+1$. Hence by induction the lemma will be true.

Assuming $N_p(p^{\beta}!) = \sum_{j=1}^{\beta} (p^{j-1})$ write out $(p^{\beta+1}!)$ in full and divide it into p sections as shown.

$$p^{(\beta+1)}! = 1\cdot 2\cdot \ldots p^{\beta} \mid (p^{\beta}+1) \ldots 2p^{\beta} \mid \ldots \mid ((p-1)p^{\beta}+1)\cdot \ldots p^{\beta+1}$$

Now by lemma 3.11 we know that

$$N_p(i+kp^{\beta}) = N_p(i) \quad \text{for } i=1,2,\ldots,p^{\beta} \text{ when } k=0,1,\ldots(p-2)$$

and also for $i=1,2,\ldots(p^{\beta}-1)$ when $k=p-1$.

Furthermore when $i=p^{\beta}$ and $k=p-1$

$$N_p(i+kp^{\beta}) = N_p((p-1+1)p^{\beta}) = N_p(p^{\beta+1}) = \beta+1 = N(p^{\beta}) +1$$

Thus p is a factor of each of the sections shown, except the last, the same number of times; and is a factor of the last one more time than of the others. However, the first section is $p^{\beta}!$, and so p is a factor $\sum_{j=1}^{\beta} (p^{j-1})$ times.

Thus
$$\begin{aligned} N_p(p^{(\beta+1)}!) &= p \sum_{j=1}^{\beta} (p^{j-1}) + 1 \\ &= \sum_{j=1}^{\beta} p^{j} + 1 \\ &= \sum_{j=0}^{\beta} p^{j} \\ &= \sum_{j=1}^{\beta+1} p^{j-1} \end{aligned}$$

QED

This lemma may be extended to give the value of $N_p(r!)$ where r is not necessarily a power of p.

Lemma 3.13

$$N_p((p^\alpha x)!) = xN_p(p^\alpha!) + N_p(x!)$$

Proof:

Clearly the lemma is true for x=1. We shall therefore assume the lemma is true for x=y and prove it for x=y+1. Hence by induction the lemma will be true.

$$(p^\alpha(y+1))! = (p^\alpha y)! \cdot (1+p^\alpha y)\cdot \ldots \cdot(y+1)^\alpha$$

$$= (p^\alpha y)! \cdot \prod_{i=1}^{p^\alpha} (i+p^\alpha y)$$

By lemma 3.11

$$N_p\ (i+p^\alpha y) = N_p(i) \quad \text{for } i=1,2,\ldots,(p^\alpha-1)$$

and $$N_p\ (p^\alpha+p^\alpha y) = \alpha + N_p(y+1) = N_p(p^\alpha) + N_p(y+1)$$

then $$N_p((p^\alpha(y+1)!) = N_p((p^\alpha y)!) + N_p(p^\alpha!) + N_p(y+1)$$

(using the induction assumption)

$$= yN_p(p^\alpha!) + N_p(y!) + N_p(p^\alpha!) + N_p(y+1)$$

$$= (y+1)\ N_p(p^\alpha!) + N_p((y+1)!)$$

[noting that $N_p((y+1)!) = N_p(y!) + N_p(y+1)$]

QED

Lemma 3.14

If $k = p^\alpha x$, where $p \nmid x$, $x > 1$ and p is prime, then $p \nmid g_k$.

Proof:

$$g_k = g.c.d\{\binom{k}{i}\} \qquad i=1,2,\ldots,(k-1).$$

To show $p \nmid g_k$ we need only find one i such that $p \nmid \binom{k}{i}$.

Consider the case in which $i = p^{\alpha}$.

Let us evaluate $N_p\left(\binom{p^{\alpha}x}{p^{\alpha}}\right)$

Since $\binom{p^{\alpha}x}{p^{\alpha}} = \dfrac{(p^{\alpha}x!)}{p^{\alpha}!\ ((x-1)p^{\alpha})!}$,

we have $N_p\left(\binom{p^{\alpha}x}{p^{\alpha}}\right) = N_p((p^{\alpha}x)!) - N_p(p^{\alpha}!) - N_p(((x-1)p^{\alpha})!)$

but then

$$N_p((p^{\alpha}x)!) = x \sum_{j=1}^{\alpha} p^{j-1} + N_p(x!)$$

$$N_p(p^{\alpha}!) = \sum_{j=1}^{\alpha} p^{j-1}$$

and $$N_p(((x-1)p^{\alpha})!) = (x-1) \sum_{j=1}^{\alpha} p^{j-1} + N_p((x-1)!)$$

from lemmas 3.12 and 3.13.

Then $$N_p\left(\binom{p^{\alpha}x}{p^{\alpha}}\right) = x \sum_{j=1}^{\alpha} p^{j-1} + N_p(x!)$$

$$- \sum_{j=1}^{\alpha} p^{j-1} - (x-1) \sum_{j=1}^{\alpha} p^{j-1} - N_p((x-1)!)$$

$$= N_p(x!) - N_p((x-1)!) + (x-1-(x-1)) \sum_{j=1}^{\alpha} p^{j-1}$$

$$= N_p(x!) - N_p((x-1)!)$$

$$= N_p(x) = 0 \qquad \text{as } p \nmid x$$

Hence $p \nmid g_k$

QED

With the proof of this lemma the desired result follows easily.

Theorem 3.1

If $k \neq p$, $g_k = 1$, or the elements of $\{\binom{k}{i}\}$ $\quad i=1,2,\ldots(k-1)$ are relatively prime.

Proof:

$k = \binom{k}{i} \subset \{\binom{k}{i}\}$, hence $g_k \mid k$.

Then for every prime p which divides k, apply lemma 3.14. Hence, no factor of k is a factor of g_k. Then, since $g_k \mid k$, $g_k = 1$.

QED

Let us now consider the case in which $k = p^{\alpha}$ and determine the size of the alphabet permitted in generating continually a string of length k by Waksman's method. We can see by the following Theorem that in this case $g_k = p$.

Theorem 3.2

If $k = p^{\alpha}$, where p is prime and $\alpha \geq 1$, then $g_k = p$.

Proof:

By the definition of g_k, it must be a factor of k. Hence, in this case, $g_k = p^{\beta}$, where $0 \leq \beta \leq \alpha$.

We shall show that $g_k = p$ by showing first that $\beta \geq 1$ and secondly that $\beta \leq 1$.

1) We are to show $\beta \geq 1$, that is, $p \mid g_k$.

Consider first the term

$$\frac{(p^{\alpha}-i)\cdot(p^{\alpha}-i+1)\ \ldots\ (p^{\alpha}-1)}{i!}\ ,$$ which is

of the form of the product of i consecutive integers

divided by i!. By Theorem 74 of Hardy and Wright (1945), i! divides the product of any i consecutive positive integers. Hence

$$N_p \left(\frac{(p^\alpha - i)(p^\alpha - i + 1) \ldots (p^\alpha - 1)}{i!} \right)$$

is defined and ≥ 0.

If we replace $(p^\alpha - i)$, in the term, by p^α, clearly the value of N_p will be increased as $N_p(p^\alpha - i) \leq N_p(p^\alpha)$ for $1 < i < p^\alpha$.

Then

$$N_p\left(\binom{p^\alpha}{i}\right) = N_p \left(\frac{(p^\alpha - i + 1) \ldots (p^\alpha - 1) \cdot p^\alpha}{i!} \right)$$

$$> N_p \left(\frac{(p^\alpha - i)(p^\alpha - i + 1) \ldots (p^\alpha - 1)}{i!} \right)$$

$$\geq 0.$$

Thus $N_p\left(\binom{p^\alpha}{i}\right) \geq 1$.

QED

2) We shall now show that $\beta \leq 1$ and so that $g_k = p$.

Consider now the case in which $i \neq p^{\alpha-1}$ and so the member of $\{\binom{p}{i}\}$, $\binom{p^\alpha}{p^{\alpha-1}}$

$$\binom{p^\alpha}{p^{\alpha-1}} = \frac{p^\alpha!}{(p^{\alpha-1})!(p^\alpha - p^{\alpha-1})!}$$

Therefore,

$$N_p\left[\binom{p^\alpha}{p^{\alpha-1}}\right] = N_p(p^\alpha!) - N_p(p^{\alpha-1}!) - N_p((p^\alpha - p^{\alpha-1})!)$$

where $N_p(p^{\alpha}!) = \sum_{j=1}^{\alpha} p^{j-1}$

$$N_p(p^{\alpha-1}!) = \sum_{j=1}^{\alpha-1} p^{j-1}$$

$$N_p((p^{\alpha} - p^{\alpha-1})!) = N_p((p^{\alpha-1}(p-1))!)$$

$$= (p-1) \sum_{j=1}^{\alpha-1} p^{j-1} + N_p(p-1) = (p-1) \sum_{j=1}^{\alpha} p^{j-1}$$

as $p \nmid (p-1)$

Therefore

$$N_p\left[\left(\frac{p^{\alpha}}{p^{\alpha-1}}\right)\right] = \sum_{j=1}^{\alpha} p^{j-1} - \sum_{j=1}^{\alpha-1} p^{j-1} - (p-1) \sum_{j=1}^{\alpha-1} p^{j-1}$$

$$= 1 + (p-1-(p-1)) \sum_{j=1}^{\alpha-1} p^{j-1}$$

$$= 1$$

Hence $g_k = p$ where $k = p^{\alpha}$

QED

We can now see the full implications of Waksman's method of continual replication of strings. If the length of the desired string, k, is a prime, P, or a power of P; then the alphabet from which the elements of k may be chosen is bijective to $\mathbb{Z}_p$. Otherwise the alphabet consists of a single character and hence no meaningful string can be generated.

As an example, let us generate the string (120222101). We note that this can be done using Waksman's technique as k=9 hence g_k=3.

The calculation of $\{b_i\}$ is carried out in Table 4.

TABLE 3

Pascal's Triangle

The binomial coefficients $\binom{i}{j}$

i ↓ \ j →	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1

TABLE 4

Calculation of $\{b_i\}$

Refer to TABLE 3 for $\binom{i-1}{j-1}$

$\{a_i\} = \{1,2,0,2,2,2,1,0,1\}$

$$b_i = \sum_{j=1}^{i} \binom{i-1}{j-1}(-1)^{i+j} a_j \qquad \text{arithmetic is mod } g_k$$

$b_1 = 1$

$b_2 = 2-1 = 1$

$b_3 = 0 -2(2) + 1 = 0$

$b_4 = 2 -3(0) + 3(2) -1 = 1$

$b_5 = 2 -4(2) + 6(0) - 4(2) +1 = 2$

$b_6 = 2 -5(2) + 10(2) - 10(0) + 5(2) - 1 = 0$

$b_7 = 1 -6(2) + 15(2) - 20(2) + 15(0) - 6(2) + 1 = 1$

$b_8 = 0 -7(1) + 21(2) - 35(2) + 35(2) - 21(0) + 7(2) - 1 = 0$

$b_9 = 1 -8(0) + 28(1) - 56(2) + 70(2) - 56(2) + 28(0) - 8(2) + 1 = 2$

With $\{b_i\}$ determined we may proceed with the continual generation of the sequence under the rules of TABLE 1.

TABLE 5

Generation of the sequence (120222101)

t ↓ \ cell i →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	P	1	1	0	1	2	0	1	0	2	Q	Q	Q	-	-	-	-	-	-	-	-
1	1	P	2	1	1	0	2	1	1	2	2	Q	Q	-	-	-	-	-	-	-	-
2	1	2	P	0	2	1	2	0	2	0	1	2	Q	Q	-	-	-	-	-	-	-
3	1	2	0	P	2	0	0	2	2	2	1	0	2	Q	Q	-	-	-	-	-	-
4	1	2	0	2	P	2	0	2	1	1	0	1	2	2	Q	Q	-	-	-	-	-
5	1	2	0	2	2	P	2	2	0	2	1	1	0	1	2	Q	Q	-	-	-	-
6	1	2	0	2	2	2	P	1	2	2	0	2	1	1	0	2	Q	Q	-	-	-
7	1	2	0	2	2	2	1	P	0	1	2	2	0	2	1	2	2	Q	Q	-	-
8	1	2	0	2	2	2	1	0	P	1	0	1	2	2	0	0	1	2	Q	Q	-
9	1	2	0	2	2	2	1	0	1	P	1	1	0	1	2	0	1	0	2	Q	Q
10	1	2	0	2	2	2	1	0	1	1	P	2	1	1	0	2	1	1	2	2	Q

After the 9th time step we notice that the pattern a_i has been generated once and that the initial configuration b_i is now in cells 10-18. At this point we see that the process will clearly generate the desired sequence continually.

IV An Alternative Method of Continual Replication of Linear Patterns - The Wheel Algorithm

It is clear that Waksman's method is very restrictive with regards to the alphabet permitted in continual generation of a sequence

of arbitrary length. It is also quite clear that no simple modification of his "modulo arithmetic scheme" can be generalized to any significant degree. For this reason we now turn to a more general method of replication. This method, in its elementary form, requires $O(n^2)$ states, however, rather than $O(n)$ as did Waksman's method, where n is the size of the alphabet from which the characters are chosen. This method may be modified somewhat so that the number of states required is about $n + 4 \log_2 n$ and so less than the $2n+2$ required in Waksman's method for reasonably large n.

The idea behind this general scheme is quite simple. Write the state of each cell as a pair of elements of the desired output alphabet -- say $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}c\\d\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}e\\f\end{smallmatrix}\right]$. Then think of this six-tuple of states as positions on a wheel rolling to the right, and so clockwise. Hence the algorithm may be referred to as the wheel algorithm. The next position of the wheel will be $\left[\begin{smallmatrix}b\\d\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}a\\f\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}c\\e\end{smallmatrix}\right]$. Now suppose b is left in the position initially occupied by $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$, that is, the wheel leaves a track behind it as it rolls, the image of the bottom part which was last in that position.

Using $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}c\\d\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}e\\f\end{smallmatrix}\right]$ as an initial configuration we have TABLE 6.

TABLE 6

$\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}c\\d\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}e\\f\end{smallmatrix}\right]$ Q Q

b $\left[\begin{smallmatrix}b\\d\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}a\\f\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}c\\e\end{smallmatrix}\right]$ Q

b d $\left[\begin{smallmatrix}d\\f\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}b\\e\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}a\\c\end{smallmatrix}\right]$ Q

b d f $\left[\begin{smallmatrix}f\\e\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}d\\c\end{smallmatrix}\right]$ $\left[\begin{smallmatrix}b\\a\end{smallmatrix}\right]$ Q

From this it can be seen the sequence b,d,f,e,c,a will be generated continually. Thus to generate a,b,c,d,e,f the appropriate initial configuration would be obtained by writing this sequence in a counter-clockwise manner around the wheel -- $[^{f}_{a}]$ $[^{e}_{b}]$ $[^{d}_{c}]$. It may be noted that this method is directly applicable to sequences of even length. If a sequence of odd length is to be replicated it may be written twice and considered a sequence of even length. Thus to generate continually [abc] the appropriate initial configuration would be $[^{c}_{a}]$ $[^{b}_{b}]$ $[^{a}_{c}]$.

The general rule for determining the state of a cell at time t+1 is that the upper half of the cell moves two positions to the right and the lower half stays in position, except at the ends. The lower half of the left-most pair stays in position, but also moves to occupy the top half of the cell to its right, which then becomes the leftmost cell. The top half of the right-most occupied cell does not move two positions to the right, but moves to occupy the bottom half of the cell immediately to its right which was previously in the quiescent state, but now becomes the right-most occupied cell.

This process violates one of the rules which Waksman had stated. That is with the wheel algorithm the state of cell i at time t+1 will depend, in general, on the state of cell i-2 (the top half of the cell moves two places to the right) at time t. To modify the scheme and avoid this problem requires the introduction of a time flip-flop and so essentially doubling the number of states. This flip-flop will alternate, being 0 at even time steps and 1 at odd times. The upper half of the pair representation of a cell will move 1 position to the right at each

time step. The ends will be handled by appropriate means depending on time. TABLE 7 formally describes the function.

TABLE 7

a,b,c	-	-	-	output characters
0	-	-	-	an arbitrary character
Q	-	-	-	quiescent state
$[^a_b]$	-	-	-	generating pairs
Φ	-	-	-	no cell present
X	-	-	-	any character

Present State / Left Neighbour	Right Neighbour $[^a_b]$	X(t=1)	X(t=0)
	$[^e_f]$	$[^e_b]$	$[^e_b]$
	g	b	$[^g_b]$
	Φ	b	-

a	X
Φ	a
b	a

Q	Q	
a	a	
$[^a_b]$	$[^0_a]$	t=1
$[^a_b]$	Q	t=0

To illustrate this, consider the following example:

It is desired to generate continually the pattern [abcb]. Then we have:

t=0 $\begin{pmatrix} b \\ a \\ 0 \end{pmatrix} \begin{pmatrix} c \\ b \\ 0 \end{pmatrix}$ Q — Initial configuration (the bottom symbol is a time flip-flop)

t=1 a $\begin{pmatrix} b \\ b \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ c \\ 1 \end{pmatrix}$ Q — Note that any state could be substituted for the dummy state 0

t=2 a $\begin{pmatrix} a \\ b \\ 0 \end{pmatrix} \begin{pmatrix} b \\ c \\ 0 \end{pmatrix}$ Q

t=3 a b $\begin{pmatrix} a \\ c \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ b \\ 1 \end{pmatrix}$ Q

t=4 a b $\begin{pmatrix} b \\ c \\ 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ 0 \end{pmatrix}$ Q

t=5 a b c $\begin{pmatrix} b \\ b \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ a \\ 1 \end{pmatrix}$ Q

t=6 a b c $\begin{pmatrix} c \\ b \\ 0 \end{pmatrix} \begin{pmatrix} b \\ a \\ 0 \end{pmatrix}$ Q

t=7 a b c b $\begin{pmatrix} c \\ a \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ b \\ 1 \end{pmatrix}$ Q

One slight drawback to this method is that 2 time steps are required to generate each new output character. The number of states required can be seen to be $0(n^2)$ as:

$2 \cdot n^2$	states are required for all possible pairs of outputs and time flip-flop.
n	output states
1	Q, the quiescent state
$2n^2+n+1$	states is the total number required.

It should be noted that the number of states required is totally independent of the length of the string to be replicated.

The next question to be asked is can the number of states required be reduced from $0(n^2)$ to $0(n)$, perhaps at the expense of the time required for generation of each output character?

The basic scheme of this algorithm is to represent each output state as a binary number and so only pairs of binary digits are manipulated. This mapping is an arbitrary bijection from the output alphabet to $\mathbb{Z}_n$. As it takes log n* bits to represent uniquely the integers of $\mathbb{Z}_n$ a time counter running from 0 to log n will be required as well. Hence 4(log n) + 1 states are required to represent this. The n output states and the quiescent state are also required. This gives a total of n + 4(log n) + 5 states which are needed for this method.

Let F denote the mapping from the output alphabet to $\mathbb{Z}_n$. For example, if the output alphabet is (a,b,c), then n=3 and we could define F(a)=0, F(b)=1,F(c)=2. TABLE 8 indicates formally the workings of the binary wheel algorithm.

*log n will be used to denote the least integer $\geq \log_2 n$.

TABLE 8

a,b,c	binary digits if in pairs, output states if alone
Q	quiescent state
0	0 state
X	any arbitrary cell state

At each step the time counter is incremented by 1 modulo ((log N) +1). Output states are reached at time (t+1).

Right Neighbour

t=0 $\begin{bmatrix} a \\ b \end{bmatrix}$	$\begin{bmatrix} c \\ d \end{bmatrix}$	Q
$\begin{bmatrix} e \\ f \end{bmatrix}$	$\begin{bmatrix} e \\ b \end{bmatrix}$	$\begin{bmatrix} e \\ b \end{bmatrix}$
Φ,g	b	b

Left Neighbour

Q	Q
Q	Q
$\begin{bmatrix} a \\ b \end{bmatrix}$	$\begin{bmatrix} 0 \\ a \end{bmatrix}$
c	c

$\begin{bmatrix} a \\ a \end{bmatrix}$	X
Φ,b	a

t=1,2,...,((log n)-1)

$\begin{bmatrix} a \\ b \end{bmatrix}$	$\begin{bmatrix} c \\ d \end{bmatrix}$	Q
$\begin{bmatrix} e \\ f \end{bmatrix}$	$\begin{bmatrix} e \\ d \end{bmatrix}$	$\begin{bmatrix} 0 \\ e \end{bmatrix}$
g	$\begin{bmatrix} \frac{F(g)}{2^{t-2}}^{*} \\ d \end{bmatrix}$	—

If right neighbour is Q then a=0 is the only possible value during these intermediate time steps.

*greatest integer $\leq \frac{F(g)}{2^{t-2}}$

Q	Q
X	Q

a	$[^{c}_{d}]$
Φ,b	$[F^{-1}(F(a) + 2^{t-1} \cdot d)]$

t=log n

$[^{a}_{b}]$	$[^{c}_{d}]$	Q
$[^{e}_{f}]$	$[^{e}_{b}]$	$[^{e}_{b}]$
g	$[^{g}_{b}]$	-

Q	Q
X	Q

a	$[^{c}_{d}]$
Φ,b	a

As an example, consider the generation of the sequence [021], where n=3 and so log n=2.

Then $0 = 00_2$

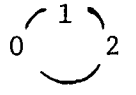
$1 = 01_2$

$2 = 10_2$

Initialize at time 0 writing 021 in binary around the wheel in a counter clockwise manner but with the order of bits of each character inverted. Since the product of log n and the length of the sequence is even only one copy of the binary representation of the sequence is required.

Thus:

t=0 $\begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}$ Q

This represents 

.

t=1 $\quad 0 \quad \begin{pmatrix}0\\0\\1\end{pmatrix} \begin{pmatrix}1\\0\\1\end{pmatrix} \begin{pmatrix}0\\1\\1\end{pmatrix}$ Q

t=2 $\quad 0 \quad \begin{pmatrix}0\\0\\2\end{pmatrix} \begin{pmatrix}0\\1\\2\end{pmatrix} \begin{pmatrix}0\\1\\2\end{pmatrix}$ Q

t=3 $\quad 0 \quad \begin{pmatrix}0\\0\\0\end{pmatrix} \begin{pmatrix}0\\1\\0\end{pmatrix} \begin{pmatrix}0\\1\\0\end{pmatrix}$ Q

t=4 $\quad 0 \; 0 \quad \begin{pmatrix}0\\1\\1\end{pmatrix} \begin{pmatrix}0\\1\\1\end{pmatrix} \begin{pmatrix}0\\0\\1\end{pmatrix}$ Q

t=5 $\quad 0 \; 2 \quad \begin{pmatrix}0\\1\\2\end{pmatrix} \begin{pmatrix}0\\0\\2\end{pmatrix} \begin{pmatrix}0\\0\\2\end{pmatrix}$ Q

t=6 $\quad 0 \; 2 \quad \begin{pmatrix}1\\1\\0\end{pmatrix} \begin{pmatrix}0\\0\\0\end{pmatrix} \begin{pmatrix}0\\0\\0\end{pmatrix}$ Q

t=7 $\quad 0 \; 2 \; 1 \quad \begin{pmatrix}1\\0\\1\end{pmatrix} \begin{pmatrix}1\\0\\1\end{pmatrix} \begin{pmatrix}0\\0\\1\end{pmatrix}$ Q

t=8 $\quad 0 \; 2 \; 1 \quad \begin{pmatrix}1\\0\\2\end{pmatrix} \begin{pmatrix}1\\0\\2\end{pmatrix} \begin{pmatrix}0\\0\\2\end{pmatrix}$ Q

t=9 $\quad 0 \; 2 \; 1 \quad \begin{pmatrix}0\\0\\0\end{pmatrix} \begin{pmatrix}1\\0\\0\end{pmatrix} \begin{pmatrix}1\\0\\0\end{pmatrix}$ Q

At this point the string has been completely generated once and the generating bud section (cells 4,5,6) is the same as the initial configuration of the original generating bud section (cells 1,2,3 at t=0).

A reminder on time considerations would appear to be in order. Waksman's method is undoubtedly the fastest of the three methods discussed as one cell is generated at each time step. The simple wheel algorithm requires two time steps for each symbol and the binary wheel is the slowest, requiring ((log n) +1) time steps for each character to be generated. The advantage of the wheel and binary wheel algorithms is that the length of the desired sequence has no bearing on the size of alphabet which may be used. In fact the size of alphabet and length of string are both completely arbitrary. For a given string length the size of alphabet is determined automatically in Waksman's method. Furthermore, it is only in special cases that the size of this alphabet is not 1. The advantage of the binary wheel algorithm over the simple wheel algorithm is the drastic reduction in the number of states required.

V Extension of the Wheel Algorithm to the 2-Dimensional Case.

The next step in the preceding line of thought is clearly to generate rectangular, rather than just linear patterns. This can be approached in much the same manner as the linear case. The problem can now be formulated in the following manner.

Define a function on a 2-dimensional array of identical finite state automata, such that the state of cell (i,j) at time (t+1) is a function of the states of that cell and its four immediate neighbours [(i,j), (i-1,j), (i+1,j), (i,j-1), (i,j+1)] and so that an arbitrary predetermined rectangular pattern of cell states will be generated

continually throughout the space.

This problem may be solved by applying the wheel algorithm twice. First the algorithm is used, moving in the horizontal direction, to produce generators similar to those used in the one dimensional case. These then generate the pattern vertically. The generators in the one dimensional case are pairs of output characters together with a time flip-flop. In the two dimensional case we shall start with pairs of pairs and a time flip-flop. The first application of the wheel algorithm generates pairs of characters, and also sets a time flip-flop to zero.

Consider the general case of rectangular pattern replication, that is, generate

$$\begin{array}{lcl} a_{m,1} & a_{m,2} \cdots\cdots\cdots & a_{m,n} \\ \vdots & & \vdots \\ a_{2,1} & & \\ a_{1,1} & a_{1,2} \cdots\cdots\cdots & a_{1,n} \end{array}$$

continually.

It should be noted that cells are numbered as points in the first quadrant of the Cartesian plane and not as matrix elements. Hence $a_{1,1}$ is in the lower left hand corner.

The first problem is to decide what the initial configuration should be. To generate the columns in the upward direction, we require that at some time column $j + kn$ $k=0,1,2,\ldots,$ $j=1,2,\ldots,n$ be of the

form

$$\left(\begin{pmatrix} a_{1,j} \\ a_{m,j} \end{pmatrix} \begin{pmatrix} a_{2,j} \\ a_{m-1,j} \end{pmatrix} \vdots \begin{pmatrix} a_{m-i+1,j} \\ a_{i,j} \end{pmatrix} \vdots \begin{pmatrix} a_{m-1,j} \\ a_{2,j} \end{pmatrix} \begin{pmatrix} a_{m,j} \\ a_{i,j} \end{pmatrix} \right)$$

with all cells above cell (m,j) in the quiescent state. Once this configuration is reached a straightforward application of the wheel algorithm will generate the (j + kn)th column in the proper manner.

Therefore the generation process on the ith row i=1,2,...,m should yield as output

$$\left(\begin{pmatrix} a_{m-i+1,1} \\ a_{i,1} \end{pmatrix} \begin{pmatrix} a_{m-i+1,2} \\ a_{i,2} \end{pmatrix} \cdots \begin{pmatrix} a_{m-i+1,j} \\ a_{i,j} \end{pmatrix} \cdots \begin{pmatrix} a_{m-i+1,n} \\ a_{i,n} \end{pmatrix} \right)$$

repeatedly. Thus the initial configuration becomes evident. Cell (i,j),

$i=1,2,\ldots,m, \quad j=1,2,\ldots,n,$ will be in state

$$\begin{pmatrix} \begin{pmatrix} a_{m-i+1,\ n-j+1} \\ a_{i,n-j+1} \end{pmatrix} \\ \\ \begin{pmatrix} a_{m-i+1,j} \\ a_{i,j} \end{pmatrix} \\ \\ 0 \end{pmatrix} \tag{5.1}$$

initially.

It may be noted that in certain cases only part of this initial configuration must be present, although the inclusion of the entire configuration as stated above will certainly produce the correct result. It may be noted that if m is even the first $\frac{m}{2}$ rows are identical to rows $\frac{m}{2}+1$ to m; and hence, only the first $\frac{m}{2}$ rows need be initialized. Similarly if n is even only the first $\frac{n}{2}$ columns need be initialized.

To illustrate this process more fully let us consider the continual generation of the pattern

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & m \end{pmatrix}$$

The time flip-flop will be seen to be 0 at even times and 1 at odd times for horizontal generation and reversed for vertical generation.

The initial configuration is given by (5.1) as

$$
\begin{array}{lccc}
 & Q & Q & \\
t=0 & \begin{pmatrix} \begin{bmatrix} m \\ d \end{bmatrix} \\ \begin{bmatrix} i \\ a \end{bmatrix} \\ 0 \end{pmatrix} & \begin{pmatrix} \begin{bmatrix} k \\ c \end{bmatrix} \\ \begin{bmatrix} j \\ b \end{bmatrix} \\ 0 \end{pmatrix} & Q \\
 & \begin{pmatrix} \begin{bmatrix} h \\ h \end{bmatrix} \\ \begin{bmatrix} e \\ e \end{bmatrix} \\ 0 \end{pmatrix} & \begin{pmatrix} \begin{bmatrix} g \\ g \end{bmatrix} \\ \begin{bmatrix} f \\ f \end{bmatrix} \\ 0 \end{pmatrix} & Q \\
 & \begin{pmatrix} \begin{bmatrix} d \\ m \end{bmatrix} \\ \begin{bmatrix} a \\ i \end{bmatrix} \\ 0 \end{pmatrix} & \begin{pmatrix} \begin{bmatrix} c \\ k \end{bmatrix} \\ \begin{bmatrix} b \\ j \end{bmatrix} \\ 0 \end{pmatrix} & Q
\end{array}
$$

The development of the pattern may be calculated by following a table similar to TABLE 7, but in which we consider pairs of the form $\begin{bmatrix} a \\ b \end{bmatrix}$ as one symbol and produce output symbols of the form $\begin{pmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \\ 0 \end{pmatrix}$. After these symbols are produced we continue with them as in TABLE 7, reading lower neighbour for left neighbour, and upper neighbour for right neighbour. Thus the desired output symbols are propagated in an upward direction.

t=1

$$\begin{pmatrix} [^{i}_{a}] \\ 0 \end{pmatrix} \quad \begin{pmatrix} [^{m}_{d}] \\ [^{j}_{b}] \\ 1 \end{pmatrix} \quad \begin{pmatrix} [^{0}_{0}] \\ [^{k}_{c}] \\ 1 \end{pmatrix} \quad Q$$

$$\begin{pmatrix} [^{e}_{e}] \\ 0 \end{pmatrix} \quad \begin{pmatrix} [^{h}_{h}] \\ [^{f}_{f}] \\ 1 \end{pmatrix} \quad \begin{pmatrix} [^{0}_{0}] \\ [^{g}_{g}] \\ 1 \end{pmatrix} \quad Q$$

$$\begin{pmatrix} [^{a}_{i}] \\ 0 \end{pmatrix} \quad \begin{pmatrix} [^{d}_{m}] \\ [^{b}_{j}] \\ 1 \end{pmatrix} \quad \begin{pmatrix} [^{0}_{0}] \\ [^{c}_{k}] \\ 1 \end{pmatrix} \quad Q$$

t=2

$$\begin{pmatrix} [^{0}_{i}] \\ 1 \end{pmatrix} \quad Q \quad Q \quad Q$$

$$\begin{pmatrix} [^{e}_{a}] \\ 1 \end{pmatrix} \quad \begin{pmatrix} [^{i}_{a}] \\ [^{j}_{b}] \\ 0 \end{pmatrix} \quad \begin{pmatrix} [^{m}_{d}] \\ [^{k}_{c}] \\ 0 \end{pmatrix} \quad Q$$

$$\begin{pmatrix} [^{a}_{e}] \\ 1 \end{pmatrix} \quad \begin{pmatrix} [^{e}_{e}] \\ [^{f}_{f}] \\ 0 \end{pmatrix} \quad \begin{pmatrix} [^{h}_{h}] \\ [^{g}_{g}] \\ 0 \end{pmatrix} \quad Q$$

$$i \quad \begin{pmatrix} [^{g}_{i}] \\ [^{b}_{j}] \\ 0 \end{pmatrix} \quad \begin{pmatrix} [^{d}_{m}] \\ [^{c}_{k}] \\ 0 \end{pmatrix} \quad Q$$

The generation of the pattern can be seen clearly from this point. The general form of the pattern while being generated can be seen at time t=12.

t=12

row \ column	1	2	3	4	5	6	7	8	9
10	Q	Q	Q	Q	Q	Q	Q	Q	Q
9	$\begin{pmatrix}[0_a]\\1\end{pmatrix}$	Q	Q	Q	Q	Q	Q	Q	Q
8	$\begin{pmatrix}[e_i]\\1\end{pmatrix}$	$\begin{pmatrix}[0_f]\\1\end{pmatrix}$	Q	Q	Q	Q	Q	Q	Q
7	$\begin{pmatrix}[i_e]\\1\end{pmatrix}$	$\begin{pmatrix}[b_j]\\1\end{pmatrix}$	$\begin{pmatrix}[0_k]\\1\end{pmatrix}$	Q	Q	Q	Q	Q	Q
6	a	$\begin{pmatrix}[j_b]\\1\end{pmatrix}$	$\begin{pmatrix}[g_c]\\1\end{pmatrix}$	$\begin{pmatrix}[0_d]\\1\end{pmatrix}$	Q	Q	Q	Q	Q
5	e	f	$\begin{pmatrix}[c_g]\\1\end{pmatrix}$	$\begin{pmatrix}[m_h]\\1\end{pmatrix}$	$\begin{pmatrix}[0_e]\\1\end{pmatrix}$	Q	Q	Q	Q
4	i	j	k	$\begin{pmatrix}[h_m]\\1\end{pmatrix}$	$\begin{pmatrix}[a_i]\\1\end{pmatrix}$	$\begin{pmatrix}[0_j]\\1\end{pmatrix}$	Q	Q	Q
3	a	b	c	d	$\begin{pmatrix}[i_a]\\1\end{pmatrix}$	$\begin{pmatrix}[f_b]\\1\end{pmatrix}$	$\begin{pmatrix}[j_b]\\[k_c]\\0\end{pmatrix}$	$\begin{pmatrix}[i_a]\\[m_d]\\0\end{pmatrix}$	Q
2	e	f	g	h	e	$\begin{pmatrix}[b_f]\\1\end{pmatrix}$	$\begin{pmatrix}[f_f]\\[g_g]\\0\end{pmatrix}$	$\begin{pmatrix}[e_e]\\[h_h]\\0\end{pmatrix}$	Q
1	i	j	k	m	i	j	$\begin{pmatrix}[b_j]\\[c_k]\\0\end{pmatrix}$	$\begin{pmatrix}[a_i]\\[d_m]\\0\end{pmatrix}$	Q

This algorithm works as quickly as can be expected. The length of each column is increased by one every second time step and the number of columns containing final output symbols is increased every second time step. However the number of states required for this operation is fairly large since we are forced to deal with quadruples of output states in the initial generation process. The actual number of states may be calculated as follows:

1	quiescent state
$2n^4$	all possible quadruples of output states in each of the two possible time positions
$2n^2$	all possible pairs in each time position
n	output states
$2n^4+2n^2+n+1$	

Clearly this is quite an undesirable number of states to require especially when n is quite large. At this point we may look back to the one dimensional case and recall that the corresponding problem was solved by mapping bijectively the n output characters, in an arbitrary manner, onto the ring of integers modulo n, that is $\mathbb{Z}_n$. Then only bits need be manipulated until the actual character generating time step. There must however, be a time counter running from 0 to log n associated with the pairs of bits and a time flip-flop associated with the quadruples. Therefore the number of states for this method may be determined as follows

1	quiescent state
$2 \cdot 2^4$	possible quadruples of bits and a time flip-flop
$2^2 \cdot ((\log n)+1$	possible pairs with time counter
n	output states
n+4 log n+36	states are required

As in the one dimensional case, the binary representation has two drawbacks. First, it is more difficult to write down the required initial configuration, which occupies log n times as many cells in each direction as does the non-binary form, and secondly, the output characters are produced only once in every (log n) + 1 time steps in each column. In general, however, the column generation processes will be out of phase with each other due to the fact that column generation will begin on a new column at every other time step. Symbols are produced in a "triangular" form as in the non-binary case. Hence the rate of production of a new output cells is proportional to the time t.

A simple example of the use of the binary wheel algorithm in two dimensions would probably make the workings of the general case much clearer.

Suppose we are to replicate continually the square $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

Then n=4, so log n=2. We can define a bijection F from the alphabet to $\mathbb{Z}_4$ by

$$a \leftrightarrow 0 = 00_2$$

$$b \leftrightarrow 1 = 01_2$$

$$c \leftrightarrow 2 = 10_2$$

$$d \leftrightarrow 3 = 11_2$$

Then using the non-binary method we could initialize the process with cell (1,1) in state $\begin{pmatrix} \begin{bmatrix} a \\ c \end{bmatrix} \\ \begin{bmatrix} b \\ d \end{bmatrix} \\ 0 \end{pmatrix}$.

However, using the binary technique the process is not quite as easy to initialize. Let us first look at the sets of pairs which

must be generated by the quadruples in order to generate the desired pattern. For the sake of clarity, let us temporarily abandon the function F as defined and let

$$F(a) = 2 \cdot a_1 + a_2$$

$$F(b) = 2 \cdot b_1 + b_2$$

$$F(c) = 2 \cdot c_1 + c_2$$

$$F(d) = 2 \cdot d_1 + d_2$$

where a_1, a_2, ... are either 0 or 1.

Then to generate the odd numbered columns, which have a {caca ...} form we must produce

$$\text{(row 2)} \begin{pmatrix} a_2 \\ c_1 \\ 0 \end{pmatrix}$$

$$\text{(row 1)} \begin{pmatrix} a_1 \\ c_2 \\ 0 \end{pmatrix}$$

as the elements in the first two rows of these columns.

Similarly

$$\text{(row 2)} \begin{pmatrix} b_2 \\ d_1 \\ 0 \end{pmatrix}$$

$$\text{(row 1)} \begin{pmatrix} b_1 \\ d_2 \\ 0 \end{pmatrix}$$

must be generated in the even numbered columns.

Thus the output for the horizontal generator in row 1 is

$$\begin{pmatrix} a_1 \\ c_2 \\ 0 \end{pmatrix} \quad \begin{pmatrix} b_1 \\ d_2 \\ 0 \end{pmatrix} \qquad \text{and in row 2}$$

$$\begin{pmatrix} a_2 \\ c_1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} b_2 \\ d_1 \\ 0 \end{pmatrix}$$

must be generated continually.

Therefore the initial configuration of row 1 is

$$\left(\begin{pmatrix} b_1 \\ d_2 \end{pmatrix} \begin{pmatrix} a_1 \\ c_2 \end{pmatrix} 0 \right) \qquad \text{and for row 2} \qquad \left(\begin{pmatrix} b_2 \\ d_1 \end{pmatrix} \begin{pmatrix} a_2 \\ c_1 \end{pmatrix} 0 \right)$$

TABLE 10 traces the development of the pattern through several time steps after replacing a_1, a_2,... with their defined values. Horizontal propagation in straight forward, vertical propagation follows TABLE 8 in Chapter IV.

TABLE 10

t=0			
	row(2)	$\begin{pmatrix} [1_1] \\ [0_1] \\ 0 \end{pmatrix}$	Q
	row(1)	$\begin{pmatrix} [0_1] \\ [0_0] \\ 0 \end{pmatrix}$	Q
t=1		$\begin{pmatrix} [0_1] \\ 0 \end{pmatrix}$ $\begin{pmatrix} [0_0] \\ [1_1] \\ 1 \end{pmatrix}$	Q
		$\begin{pmatrix} [0_0] \\ 0 \end{pmatrix}$ $\begin{pmatrix} [0_0] \\ [0_1] \\ 1 \end{pmatrix}$	Q
t=2		$\begin{pmatrix} [0_0] \\ 1 \end{pmatrix}$ Q	Q
		$\begin{pmatrix} [0_1] \\ 1 \end{pmatrix}$ $\begin{pmatrix} [0_1] \\ [1_1] \\ 0 \end{pmatrix}$	Q
		a $\begin{pmatrix} [0_0] \\ [0_1] \\ 0 \end{pmatrix}$	Q

t=3

$$\begin{pmatrix}[0_0]\\0\\2\end{pmatrix} \quad Q \quad Q \quad Q$$

$$\begin{pmatrix}[0_0]\\0\\2\end{pmatrix} \quad \begin{pmatrix}[1_1]\\1\\0\end{pmatrix} \quad \begin{pmatrix}[0_0]\\[0_1]\\1\end{pmatrix} \quad Q$$

$$c \quad \begin{pmatrix}[0_1]\\1\\0\end{pmatrix} \quad \begin{pmatrix}[0_0]\\[0_0]\\1\end{pmatrix} \quad Q$$

t=4

$$\begin{pmatrix}[0_0]\\0\\0\end{pmatrix} \quad \begin{pmatrix}[0_1]\\1\\1\end{pmatrix} \quad Q \quad Q$$

$$\begin{pmatrix}[1_0]\\0\\0\end{pmatrix} \quad \begin{pmatrix}[0_1]\\1\\1\end{pmatrix} \quad \begin{pmatrix}[1_1]\\[0_1]\\0\end{pmatrix} \quad Q$$

$$c \quad b \quad \begin{pmatrix}[0_1]\\[0_0]\\0\end{pmatrix} \quad Q$$

t=5

$$\begin{pmatrix}[0_0]\\1\\1\end{pmatrix} \quad Q \quad Q \quad Q \quad Q$$

$$\begin{pmatrix}[1_0]\\1\\1\end{pmatrix} \quad \begin{pmatrix}[0_0]\\2\\2\end{pmatrix} \quad Q \quad Q \quad Q$$

$$a \quad \begin{pmatrix}[1_1]\\2\\2\end{pmatrix} \quad \begin{pmatrix}[0_1]\\0\\0\end{pmatrix} \quad \begin{pmatrix}[0_0]\\[1_1]\\1\end{pmatrix} \quad Q$$

$$c \quad d \quad \begin{pmatrix}[0_0]\\0\\0\end{pmatrix} \quad \begin{pmatrix}[0_0]\\[0_1]\\1\end{pmatrix} \quad Q$$

From this the general replication pattern may be seen.

VI Extension to d-dimensional space

The next step in the development of this theory is the extension to arbitrary (d) dimensional space. That is, to define a function on a d-dimensional array of identical finite automata so that, given the appropriate finite initial configuration, the entire positive region of d-space will be filled with repeated images of an arbitrary predetermined d-dimensional hypercuboid. Again we have the condition that the state of any cell at time (t+1) be a function of the states of that cell and its 2^d immediate neighbours at time t.

The technique used is the obvious extension of that used in the 2-dimensional case. The wheel algorithm is used on the dth level generators to form (d-1)st level generators. These in turn generate (d-2)nd level generators in the same way that the vertical or first level generators were produced by the horizontal, or second level generators in the 2-dimensional case. In any case, after d such transformations the terminal or output states emerge. As in the 2-dimensional case, when n is large the number of states required becomes much larger. Actually, as may be expected, when d is large the number of states required becomes astronomical. Therefore, to keep the number of states within reason signals may be sent in binary, as in the 2-dimensional binary wheel method. Only in the actual generation of output symbols do non-binary forms have to be dealt with. Hence, it should not be surprising that the effect of dimension size (d) and alphabet size (n) upon

number of states required are totally independent. That is

number states = F(d) + G(n).

Let us now determine the number of states required to generate continually an arbitrary d-dimensional hypercuboid of elements of an alphabet of cardinality n. First consider the non-binary representation. To apply the wheel algorithm and move from 1 dimensional generators to the output state we require the quiescent state, the n output states and all possible pairs of outputs together with a time flip-flop, or $2n^2$ states. To generate the pairs, quadruples are needed, and so on up to 2^d tuples. Thus the number of states S(d,n) required will be given by

$$S(d,n) = 1 + n + \sum_{j=1}^{d} n^{2^j} \quad ; \qquad (6.1)$$

a large number for surprisingly small d.

In the binary wheel representation, the number is much lower. The quiescent state and n output states are required as are the 4((log n)+1) states which are needed in moving from binary pairs to output states. At higher levels, however, the process is a simple transfer of 2^j tuples. In general the jth level requires all possible 2^j tuples at both time positions, or $2 \cdot 2^{2^j}$ states. Therefore the number of states required in the d-dimensional application of the binary wheel algorithm ($S_b(d,n)$) is given by

$$S_b(d,n) = 1 + n + 4((\log n)+1) + 2 \sum_{j=2}^{d} 2^{2^j}$$

$$= n + 4\log n + \sum_{j=1}^{d} 2^{2^j+1} - 3. \qquad (6.3)$$

As was mentioned before, this can be viewed as the sum of a function of n and a function of d.

VII Conclusion

The restrictions inherent in Waksman's method of continual replication of a linear string have been shown. If the length of the desired string is not of the form P^{α} where P is a prime, only one character may be produced. This means that no meaningful string may be generated. Furthermore if the string length is of the form P , only P output characters are permitted. These restrictions cannot be overcome using a modulo arithmetic algorithm. For this reason the wheel algorithm was developed. Using this algorithm the number of characters in the output alphabet is completely independent of the length of the string, and in fact, both are arbitrary. The binary wheel algorithm was developed to reduce the number of states required to produce a string containing n different characters to $0(n)$. It was shown also that both the wheel algorithm and the binary wheel algorithm can be generalized to produce continually a d-dimensional hypercuboid. Finally, it should again be noted that the number of states required to generate patterns in d-space using an alphabet of cardinality n is of the form

$$F(d) + G(n)$$

where

$$F(d) = 0(2^{2^d})$$

and $$G(n) = n + 4\log_2 n.$$

BIBLIOGRAPHY

1. Hardy, G. H. and Wright, E. M. An Introduction to the Theory of Numbers. Oxford University Press, London, 1945.

2. Waksman, A. A Model of Replication. Journal of the Association for Computing Machinery 16,1 (January 1969), pp. 178-188.