

ON THE LENGTH OF PROGRAMS FOR COMPUTING FINITE BINARY SEQUENCES

Journal of the ACM 13 (1966),
pp. 547–569

Gregory J. Chaitin¹

*The City College of the City University of New York
New York, N.Y.*

Abstract

The use of Turing machines for calculating finite binary sequences is studied from the point of view of information theory and the theory of recursive functions. Various results are obtained concerning the number of instructions in programs. A modified form of Turing machine is studied from the same point of view. An application to the problem of defining a patternless sequence is proposed in terms of the concepts here

developed.

Introduction

In this paper the Turing machine is regarded as a general purpose computer and some practical questions are asked about programming it. Given an arbitrary finite binary sequence, what is the length of the shortest program for calculating it? What are the properties of those binary sequences of a given length which require the longest programs? Do most of the binary sequences of a given length require programs of about the same length?

The questions posed above are answered in Part 1. In the course of answering them, the logical design of the Turing machine is examined as to redundancies, and it is found that it is possible to increase the efficiency of the Turing machine as a computing instrument without a major alteration in the philosophy of its logical design. Also, the following question raised by C. E. Shannon [1] is partially answered: What effect does the number of different symbols that a Turing machine can write on its tape have on the length of the program required for a given calculation?

In Part 2 a major alteration in the logical design of the Turing machine is introduced, and then all the questions about the lengths of programs which had previously been asked about the first computer are asked again. The change in the logical design may be described in the following terms: Programs for Turing machines may have transfers from any part of the program to any other part, but in the programs for the Turing machines which are considered in Part 2 there is a fixed upper bound on the length of transfers.

Part 3 deals with the somewhat philosophical problem of defining a random or patternless binary sequence. The following definition is proposed: Patternless finite binary sequences of a given length are sequences which in order to be computed require programs of approximately the same length as the longest programs required to compute

¹This paper was written in part with the help of NSF Undergraduate Research Participation Grant GY-161.

any binary sequences of that given length. Previous work along these lines and its relationship to the present proposal are discussed briefly.

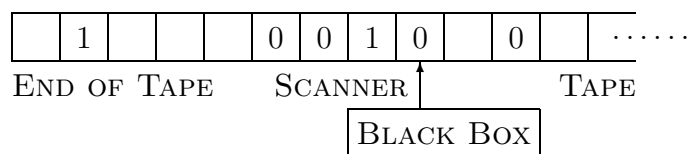
Part 1

1.1. We define an N -state M -tape-symbol Turing machine by an N -row by M -column table. Each of the NM places in this table must have an entry consisting of an ordered pair (i, j) of natural numbers, where i goes from 0 to N and j goes from 1 to $M + 2$. These entries constitute, when specified, the program of the N -state M -tape-symbol Turing machine. They are to be interpreted as follows: An entry (i, j) in the k th row and the p th column of the table means that when the machine is in its k th state and the square of its one-way infinite tape which is being scanned is marked with the p th symbol, then the machine is to go to its i th state if $i \neq 0$ (the machine is to halt if $i = 0$) after performing the operation of

1. moving the tape one square to the right if $j = M + 2$,
2. moving the tape one square to the left if $j = M + 1$, and
3. marking (overprinting) the square of the tape being scanned with the j th symbol if $1 \leq j \leq M$.

Special names are given to the first, second and third symbols. They are, respectively, the blank (for unmarked square), 0 and 1.

A Turing machine may be represented schematically as follows:



It is stipulated that

(1.1A) Initially the machine is in its first state and scanning the first square of the tape.

N large at least $1/M$ of the bits of information of a program are redundant. Later we shall be in a position to ask to what extent the remaining portion of $(1 - 1/M)$ of the bits is redundant.

The basic reason for this redundancy is that any renumbering of the rows of the table (this amounts to a renaming of the states of the machine) in no way changes the behavior that a given program will cause the machine to have. Thus the states can be named in a manner determined by the sequencing of the program, and this makes possible the omission of state numbers from the program. This idea is by no means new. It may be seen in most computers with random access memories. In these computers the address of the next instruction to be executed is usually 1 more than the address of the current instruction, and this makes it generally unnecessary to use memory space in order to give the address of the next instruction to be executed. Since we are not concerned with the practical engineering feasibility of a logical design, we can take this idea a step farther.

1.3. In the presentation of the redesigned Turing machine let us begin with an example of the manner in which one can take a program for a Turing machine and reorder its rows (rename its states) until it is in the format of the redesigned machine. In the process, several row numbers in the program are removed and replaced by + or ++ —this is how redundant information in the program is removed. The “operation codes” (which are 1 for “print blank,” 2 for “print zero,” 3 for “print one,” 4 for “shift tape left” and 5 for “shift tape right”) are omitted from the program; every time the rows are reordered, the op-codes are just carried along. The program used as an example is as follows:

row 1	1	9	7
row 2	8	8	8
row 3	9	6	1
row 4	3	2	0
row 5	7	7	8
row 6	6	5	4
row 7	8	6	9
row 8	9	8	1
row 9	9	1	8

To prevent confusion later, letters instead of numbers are used in

the program:

row A	A	I	G
row B	H	H	H
row C	I	F	A
row D	C	B	J
row E	G	G	H
row F	F	E	D
row G	H	F	I
row H	I	H	A
row I	I	A	H

Row A is the first row of the table and shall remain so. Replace A by 1 throughout the table:

row 1	1	I	G
row B	H	H	H
row C	I	F	1
row D	C	B	J
row E	G	G	H
row F	F	E	D
row G	H	F	I
row H	I	H	1
row I	I	1	H

To find to which row of the table to assign the number 2, read across the first row of the table until a letter is reached. Having found an I,

1. replace it by a +,
2. move row I so that it becomes the second row of the table, and
3. replace I by 2 throughout the table:

row 1	1	+	G
row 2	2	1	H
row B	H	H	H
row C	2	F	1
row D	C	B	J
row E	G	G	H
row F	F	E	D
row G	H	F	2
row H	2	H	1

To find to which row of the table to assign the number 3, read across the second row of the table until a letter is found. Having found an H,

1. replace it by a +,
2. move row H so that it becomes the third row of the table, and
3. replace H by 3 throughout the table:

row 1	1	+	G
row 2	2	1	+
row 3	2	3	1
row B	3	3	3
row C	2	F	1
row D	C	B	J
row E	G	G	3
row F	F	E	D
row G	3	F	2

To find to which row of the table to assign the number 4, read across the third row of the table until a letter is found. Having failed to find one, read across rows 1, 2 and 3, respectively, until a letter is found. (A letter must be found, for otherwise rows 1, 2 and 3 are the whole program.) Having found a G in row 1,

1. replace it by a ++,
2. move row G so that it becomes the fourth row of the table, and
3. replace G by 4 throughout the table:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	F	2
row B	3	3	3
row C	2	F	1
row D	C	B	J
row E	4	4	3
row F	F	E	D

The next two assignments proceed as in the case of rows 2 and 3:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	E	D
row B	3	3	3
row C	2	5	1
row D	C	B	J
row E	4	4	3

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	+	D
row 6	4	4	3
row B	3	3	3
row C	2	5	1
row D	C	B	J

To find to which row of the table to assign the number 7, read across the sixth row of the table until a letter is found. Having failed to find one, read across rows 1, 2, 3, 4, 5 and 6, respectively, until a letter is found. (A letter must be found, for otherwise rows 1, 2, 3, 4, 5 and 6 are the whole program.) Having found a D in row 5,

1. replace it by a ++,
2. move row D so that it becomes the seventh row of the table, and
3. replace D by 7 throughout the table:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	+	++
row 6	4	4	3
row 7	C	B	J
row B	3	3	3
row C	2	5	1

After three more assignments the following is finally obtained:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	+	++
row 6	4	4	3
row 7	+	++	++
row 8	2	5	1
row 9	3	3	3
row 10			

This example is atypical in several respects: The state order could have needed a more elaborate scrambling (instead of which the row of the table to which a number was assigned always happened to be the last row of the table at the moment), and the fictitious state used for the purposes of halting (state 0 in the formulation of Section 1.1) could have ended up as any one of the rows of the table except the first row (instead of which it ended up as the last row of the table).

The reader will note, however, that 9 row numbers have been eliminated (and replaced by + or ++) in a program of 9 (actual) rows, and that, in general *this process will eliminate a row number from the*

program for each row of the program. Note too that if a program is “linear” (i.e., the machine executes the instruction in storage address 1, then the instruction in storage address 2, then the instruction in storage address 3, etc.), only + will be used; departures from linearity necessitate use of ++.

There follows a description of the redesigned machine. In the formalism of that description the program given above is as follows:

	(1, ,0)	(0, ,1)	(0, ,2)
	(2, ,0)	(1, ,0)	(0, ,1)
	(2, ,0)	(3, ,0)	(1, ,0)
	(3, ,0)	(0, ,1)	(2, ,0)
10	(5, ,0)	(0, ,1)	(0, ,2)
	(4, ,0)	(4, ,0)	(3, ,0)
	(0, ,1)	(0, ,2)	(0, ,2)
	(2, ,0)	(5, ,0)	(1, ,0)
	(3, ,0)	(3, ,0)	(3, ,0)

Here the third member of a triple is the number of +’s, the second member is the op-code, and the first member is the number of the next state of the machine if there are no +’s (if there are +’s, the first member of the triple is 0). The number outside the table is the number of the fictitious row of the program used for the purposes of halting.

We define an N -state M -tape-symbol Turing machine by an $(N + 1) \times M$ table and a natural number n ($2 \leq n \leq N + 1$). Each of the $(N + 1)M$ places in this table (with the exception of those in the n th row) must have an entry consisting of an ordered triple (i, j, k) of natural numbers, where k is 0, 1 or 2; j goes from 1 to $M + 2$; and i goes from 1 to $N + 1$ if $k = 0$, $i = 0$ if $k \neq 0$. (Places in the n th row are left blank.) In addition:

(1.3.1) The entries in which $k = 1$ or $k = 2$ are N in number.

Entries are interpreted as follows:

(1.3.2) An entry $(i, j, 0)$ the p th row and the m th column of the table means that when the machine is in the p th state and the square of its one-way infinite tape which is being scanned is marked with the m th symbol, then the machine is to go to its i th state if $i \neq n$

(if $i = n$, the machine is instead to halt) after performing the operation of

1. moving the tape one square to the right if $j = M + 2$,
2. moving the tape one square to the left if $j = M + 1$, and
3. marking (overprinting) the square of the tape being scanned with the j th symbol if $1 \leq j \leq M$.

(1.3.3) An entry $(0, j, 1)$ in the p th row and m th column of the table is to be interpreted in accordance with (1.3.2) as if it were the entry $(p + 1, j, 0)$.

(1.3.4) For an entry $(0, j, 2)$ in the p th row and m th column of the table the machine proceeds as follows:

(1.3.4a) It determines the number p' of entries of the form $(0, , 2)$ in rows of the table preceding the p th row or to the left of the m th column in the p th row.

(1.3.4b) It determines the first $p' + 1$ rows of the table which have no entries of the form $(0, , 1)$ or $(0, , 2)$. Suppose the last of these $p' + 1$ rows is the p'' th row of the table.

(1.3.4c) It interprets the entry in accordance with (1.3.2) as if it were the entry $(p'' + 1, j, 0)$.

1.4. In Section 1.2 it was stated that the programs of the N -state M -tape-symbol Turing machines of Section 1.3 require in order to be specified $(1 - 1/M)$ the number of bits of information required to specify the programs of the N -state M -tape-symbol Turing machines of Section 1.1. (As before, M is regarded to be fixed and N to be large.) This assertion is justified here. In view of (1.3.1), at most

$$N(3(M + 2))^{NM}(N + 1)^{N(M-1)}$$

ways of making entries in the table of an N -state M -tape-symbol Turing machine of Section 1.3 count as programs. Thus only \log_2 of this number or asymptotically $N(M - 1) \log_2 N$ bits are required to specify the program of an N -state M -tape-symbol machine of Section 1.3.

Henceforth, in speaking of an N -state M -tape-symbol Turing machine, one of the machines of Section 1.3 will be meant.

1.5. We now define two sets of functions which play a fundamental role in all that follows.

The members $L_M(\cdot)$ of the first set are defined for $M = 3, 4, 5, \dots$ on the set of all finite binary sequences S as follows: An N -state M -tape-symbol Turing machine can be programmed to calculate S if and only if $N \geq L_M(S)$.

The second set $L_M(C_n)$ ($M = 3, 4, 5, \dots$) is defined by

$$L_M(C_n) = \max_S L_M(S),$$

where S is any binary sequence of length n .

Finally, we denote by C_n^M ($M = 3, 4, 5, \dots$) the set of all binary sequences S of length n satisfying $L_M(S) = L_M(C_n)$.

1.6. In this section it is shown that for $M = 3, 4, 5, \dots$,

$$L_M(C_n) \sim \frac{n}{(M-1)\log_2 n}.$$

We first show that $L_M(C_n)$ is greater than a function of n which is asymptotically equal to $(n/((M-1)\log_2 n))$. From Section 1.4 it is clear that there are at most

$$2^{((1+\epsilon_N)N(M-1)\log_2 N)}$$

different programs for an N -state M -tape-symbol Turing machine, where ϵ_x denotes a (not necessarily positive) function of x and possibly other variables which tends to zero as x goes to infinity with any other variables held fixed. Since a different program is required to calculate each of the 2^n different binary sequences of length n , we see that an N -state M -tape-symbol Turing machine can be programmed to calculate any binary sequence of length n only if

$$(1 + \epsilon_N)N(M-1)\log_2 N \geq n$$

or

$$N \geq (1 + \epsilon_n) \frac{n}{(M-1)\log_2 n}.$$

It follows from the definition of $L_M(C_n)$ that

$$L_M(C_n) \geq (1 + \epsilon_n) \frac{n}{(M - 1) \log_2 n}.$$

Next we show that $L_M(C_n)$ is less than a function of n which is asymptotically equal to $(n/((M - 1) \log_2 n))$. This is done by showing how to construct for any binary sequence S of length not greater than $(1 + \epsilon_N)N(M - 1) \log_2 N$ a program which causes an N -state M -tape-symbol Turing machine to calculate S . The main idea is illustrated in the case where $M = 3$.

Row Number	Column Number			
	1	2	3	
1	2, 4	2, 4	2, 4	Section I: approximately $(1 - 1/\log_2 N)N$ rows
2	..., 2	..., 3	3, 4	
3	..., 2	..., 3	4, 4	
4	..., 2	..., 3	5, 4	
5	..., 2	..., 3	6, 4	
6	..., 2	..., 3	7, 4	
7	..., 2	..., 3	8, 4	
8	..., 2	..., 3	9, 4	
⋮	⋮	⋮	⋮	
d	$d + 1, 4$	$d + 1, 4$	$d + 1, 4$	Section II: approximately $N/\log_2 N$ rows
$d + 1$	$d + 2, 4$	$d + 2, 4$	$d + 2, 4$	
$d + 2$	$d + 3, 4$	$d + 3, 4$	$d + 3, 4$	
$d + 3$	$d + 4, 4$	$d + 4, 4$	$d + 4, 4$	
$d + 4$	$d + 5, 4$	$d + 5, 4$	$d + 5, 4$	
$d + 5$	$d + 6, 4$	$d + 6, 4$	$d + 6, 4$	
$d + 6$	$d + 7, 4$	$d + 7, 4$	$d + 7, 4$	
$d + 7$	$d + 8, 4$	$d + 8, 4$	$d + 8, 4$	
⋮	⋮	⋮	⋮	
f	This section is the same (except for the changes in row numbers caused by relocation) regardless of the value of N .			Section III: a fixed number of rows
$f + 1$				
$f + 2$				
$f + 3$				
$f + 4$				
⋮	⋮	⋮	⋮	

Now control has been passed to Section III. First of all, Section III accumulates in base-two on the tape a count of the number of blank squares between the scanner and P when f assumes control. (This number is $m - 1$.) This base-two count, which is written on the tape, is simply a binary sequence with a 1 at its left end. Section III then removes this 1 from the left end of the binary sequence. The resulting sequence is called S_n .

Note that if the row numbers entered in

$$\begin{cases} \text{row } i + 2, \text{ column 1 if } n = 2i + 1, \\ \text{row } i + 2, \text{ column 2 if } n = 2i + 2, \end{cases}$$

of Section I are suitably specified, this binary sequence S_n can be made any one of the 2^v binary sequences of length $v =$ (the greatest integer not greater than $\log_2(f - d) - 1$). Finally, Section III writes S_n in a region of the tape far to the right where all the previous S_j ($j = 1, 2, \dots, n - 1$) have been written during previous phases, cleans up the tape so that only the sequences P and S_j ($j = 1, 2, 3, \dots, n$) remain on it, positions the scanner back on the square at the end of the tape and, as the last act of phase n , passes control back to row 1 again.

The foregoing description of the workings of the program omits some important details for the sake of clarity. These follow.

It must be indicated how Section III knows when the last phase (phase $2(d - 2)$) has occurred. During the n th phase, P is copied just to the right of S_1, S_2, \dots, S_n (of course a blank square is left between S_n and the copy of P). And during the $(n + 1)$ -th phase, Section III checks whether or not P is currently different from what it was during the n th phase when the copy of it was made. If it isn't different, then Section III knows that phasing has in fact stopped and that a termination routine must be executed.

The termination routine first forms the finite binary sequence S^* consisting of

$$S_1, S_2, \dots, S_{2(d-2)},$$

each immediately following the other. As each of the S_j can be any one of the 2^v binary sequences of length v if the row numbers in the entries in Section I are appropriately specified, it follows that S^* can be any

one of the 2^w binary sequences of length $w = 2(d - 2)v$. Note that

$$2(d - 2)(\log_2(f - d) - 1) \geq w > 2(d - 2)(\log_2(f - d) - 2),$$

so that

$$w \sim 2 \left(\left(1 - \frac{1}{\log_2 N}\right)N \right) \left(\log_2 \frac{N}{\log_2 N} \right) \sim 2N \log_2 N.$$

As we want the program to be able to compute any sequence S of length not greater than $(2 + \epsilon_N)N \log_2 N$, we have S^* consist of S followed to the right by a single 1 and then a string of 0's, and the termination routine removes the rightmost 0's and first 1 from S^* . Q.E.D.

The result just obtained shows that it is impossible to make further improvement in the logical design of the Turing machine of the kind described in Section 1.2 and actually effected in Section 1.3; if we let the number of tape symbols be fixed and speak asymptotically as the number of states goes to infinity, in our present Turing machines 100 percent of the bits required to specify a program also serve to specify the behavior of the machine.

Note too that the argument presented in the first paragraph of this section in fact establishes that, say, for any fixed s greater than zero, at most $n^{-s}2^n$ binary sequences S of length n satisfy

$$L_M(S) \leq (1 + \epsilon_n) \frac{n}{(M - 1) \log_2 n}.$$

Thus we have: For any fixed s greater than zero, at most $n^{-s}2^n$ binary sequences of length n fail to satisfy the double inequality

$$(1 + \epsilon_n) \frac{n}{(M - 1) \log_2 n} \leq L_M(S) \leq (1 + \epsilon'_n) \frac{n}{(M - 1) \log_2 n}.$$

1.7. It may be desirable to have some idea of the “local” as well as the “global” behavior of $L_M(C_n)$. The following program of 8 rows causes an 8-state 3-tape-symbol Turing machine to compute the binary sequence 01100101 of length 8 (this program is in the format of the machines of Section 1.1):

1,2	2,4	2,4
2,3	3,4	3,4
3,3	4,4	4,4
4,2	5,4	5,4
5,2	6,4	6,4
6,3	7,4	7,4
7,2	8,4	8,4
8,3	0,4	0,4

And in general:

(1.7.1) $L_M(C_n) \leq n$.

From this it is easy to see that for m greater than n :

(1.7.2) $L_M(C_m) \leq L_M(C_n) + (m - n)$.

Also, for m greater than n :

(1.7.3) $L_M(C_m) + 1 \geq L_M(C_n)$.

For if one can calculate any binary sequence of length m greater than n with an M -tape-symbol Turing machine having $L_M(C_m)$ states, one can certainly program any M -tape-symbol Turing machine having $L_M(C_m) + 1$ states to calculate the binary sequence consisting of (any particular sequence of length n) followed by a 1 followed by [a sequence of $(m - n - 1)$ 0's], and then—instead of immediately halting—to first erase all the 0's and the first 1 on the right end of the sequence. This last part of the program takes up only a single row of the table; in the format of the machines of Section 1.1 this row r is:

row r	$r,5$	$r,1$	0,1
---------	-------	-------	-----

Together (1.7.2) and (1.7.3) yield:

(1.7.4) $|L_M(C_{n+1}) - L_M(C_n)| \leq 1$.

From (1.7.1) it is obvious that $L_M(C_1) = 1$, and with (1.7.4) and the fact that $L_M(C_n)$ goes to infinity with n it finally is concluded that:

(1.7.5) For any positive integer p there is at least one solution n of

$$L_M(C_n) = p.$$

1.8. In this section a certain amount of insight is obtained into the properties of finite binary sequences S of length n for which $L_M(S)$ is close to $L_M(C_n)$. M is considered to be fixed throughout this section. There is some connection between the present subject and that of Shannon in [2, Pt. I, especially Th. 9].

The main result is as follows:

(1.8.1) For any $e > 0$ and $d > 1$ one has for all sufficiently large n :
If S is any binary sequence of length n satisfying the statement that

(1.8.2) the ratio of the number of 0's in S to n differs from $\frac{1}{2}$ by more than e ,

then $L_M(S) < L_M(C_{\lceil ndH(\frac{1}{2}+e, \frac{1}{2}-e) \rceil})$.

Here $H(p, q)$ ($p \geq 0, q \geq 0, p+q = 1$) is a special case of the entropy function of Boltzmann statistical mechanics and information theory and equals 0 if $p = 0$ or 1, and $-p \log_2 p - q \log_2 q$ otherwise. Also, a real number enclosed in brackets denotes the least integer greater than the enclosed real. The H function comes up because the logarithm to the base-two of the number

$$\sum_{\left| \frac{k}{n} - \frac{1}{2} \right| > e} \binom{n}{k}$$

of binary sequences of length n satisfying (1.8.2) is asymptotic to $nH(\frac{1}{2} + e, \frac{1}{2} - e)$. This may be shown easily by considering the ratio of successive binomial coefficients and using the fact that $\log(n!) \sim n \log n$.

To prove (1.8.1), first construct a class of effectively computable functions $M_n(\cdot)$ with the natural numbers from 1 to 2^n as range and all binary sequences of length n as domain. $M_n(S)$ is defined to be the ordinal number of the position of S in an ordering of the binary sequences of length n defined as follows:

1. If two binary sequences S and S' have, respectively, m and m' 0's, then S comes before (after) S' according as $|\frac{m}{n} - \frac{1}{2}|$ is greater (less) than $|\frac{m'}{n} - \frac{1}{2}|$.

2. If 1 does not settle which comes first, take S to come before (after) S' according as S represents (ignoring 0's to the left) a larger (smaller) number in base-two notation than S' represents.

The only essential feature of this ordering is that it gives small ordinal numbers to sequences for which $|\frac{m}{n} - \frac{1}{2}|$ has large values. In fact, as there are only

$$2^{(1+\epsilon_n)nH(\frac{1}{2}+e, \frac{1}{2}-e)}$$

binary sequences S of length n satisfying (1.8.2), it follows that at worst $M_n(S)$ is a number which in base-two notation is represented by a binary sequence of length $\sim nH(\frac{1}{2}+e, \frac{1}{2}-e)$. Thus in order to obtain a short program for computing an S of length n satisfying (1.8.2), let us just give a program of fixed length r the values of n and $M_n(S)$ and have it compute $S (= M_n^{-1}(M_n(S)))$ from this data. The manner in which for n sufficiently large we give the values of n and $M_n(S)$ to the program is to pack them into a single binary sequence of length at most

$$\left[n\left(1 + \frac{d-1}{2}\right)H\left(\frac{1}{2} + e, \frac{1}{2} - e\right) \right] + 2(1 + \lceil \log_2 n \rceil)$$

as follows: Consider (the binary sequence representing $M_n(S)$ in base-two notation) followed by 01 followed by [the binary sequence representing n with each of its bits doubled (e.g., if $n = 43$, this is 110011001111)]. Clearly both n and $M_n(S)$ can be recovered from this sequence. And this sequence can be computed by a program of

$$L_M(C_{\lceil n(1+\frac{d-1}{2})H(\frac{1}{2}+e, \frac{1}{2}-e) \rceil + 2(1+\lceil \log_2 n \rceil)})$$

rows. Thus for n sufficiently large this many rows plus r is all that is needed to compute any binary sequence S of length n satisfying (1.8.2). And by the asymptotic formula for $L_M(C_n)$ of Section 1.6, it is seen that the total number of rows of program required is, for n sufficiently large, less than

$$L_M(C_{\lceil ndH(\frac{1}{2}+e, \frac{1}{2}-e) \rceil}).$$

Q.E.D.

From (1.8.1) and the fact that $H(p, q) \leq 1$ with equality if and only if $p = q = \frac{1}{2}$, it follows from $L_M(C_n) \sim (n/((M-1)\log_2 n))$ that, for example,

(1.8.3) For any $\epsilon > 0$, all binary sequences S in C_n^M , n sufficiently large, violate (1.8.2);

and more generally,

(1.8.4) Let

$$S_{n_1}, S_{n_2}, S_{n_3}, \dots$$

be any infinite sequence of distinct finite binary sequences of lengths, respectively, n_1, n_2, n_3, \dots which satisfies

$$L_M(S_{n_k}) \sim L_M(C_{n_k}).$$

Then as k goes to infinity, the ratio of the number of 0's in S_{n_k} to n_k tends to the limit $\frac{1}{2}$.

We now wish to apply (1.8.4) to programs for Turing machines. In order to do this we need to be able to represent the table of entries defining any program as a single binary sequence. A method is sketched here for coding any program $T_{N,M}$ occupying the table of an N -state M -tape-symbol Turing machine into a single binary sequence $C(T_{N,M})$ of length $(1 + \epsilon_N)N(M - 1) \log_2 N$.

First, write all the members of the ordered triples entered in the table in base-two notation, adding a sufficient number of 0's to the left of the numerals for all numerals to be

1. as long as the base-two numeral for $N + 1$ if they result from the first member of a triple,
2. as long as the base-two numeral for $M + 2$ if they result from the second member, and
3. as long as the base-two numeral for 2 if they result from the third member.

The only exception to this rule is that if the third member of a triple is 1 or 2, then the first member of the triple is not written in base-two notation; no binary sequences are generated from the first members of such triples. Last, all the binary sequences that have just been obtained are joined together, starting with the binary sequence

that was generated from the first member of the triple entered at the intersection of row 1 with column 1 of the table, then with the binary sequence generated from the second member of the triple... , ... from the third member... , ... from the first member of the triple entered at the intersection of row 1 with column 2, ... from the second member... , ... from the third member... , and so on across the first row of the table, then across the second row of the table, then the third, ... and finally across the N th row.

The result of all this is a single binary sequence of length $(1 + \epsilon_N)N(M - 1) \log_2 N$ (in view of (1.3.1)) from which one can effectively determine the whole table of entries which was coded into it, if only one is given the values of N and M . But it is possible to code in these last pieces of information using only the rightmost

$$2(1 + \lceil \log_2 N \rceil) + 2(1 + \lceil \log_2 M \rceil)$$

bits of a binary sequence consequently of total length

$$\begin{aligned} (1 + \epsilon_N)N(M - 1) \log_2 N + 2(1 + \lceil \log_2 N \rceil) + 2(1 + \lceil \log_2 M \rceil) \\ = (1 + \epsilon'_N)N(M - 1) \log_2 N, \end{aligned}$$

by employing the same trick that was used to pack two pieces of information into a single binary sequence earlier in this section.

Thus we have a simple procedure for coding the whole table of entries $T_{N,M}$ defining a program of an N -state M -tape-symbol Turing machine and the parameters N and M of the machine into a binary sequence $C(T_{N,M})$ of $(1 + \epsilon_N)N(M - 1) \log_2 N$ bits.

We now obtain the result:

(1.8.5) Let

$$T_{L_M(S_1),M}, T_{L_M(S_2),M}, \dots$$

be an infinite sequence of tables of entries which define programs for computing, respectively, the distinct finite binary sequences S_1, S_2, \dots . Then

$$L_M(C(T_{L_M(S_k),M})) \sim L_M(C_{n_k}),$$

where n_k is the length of

$$C(T_{L_M(S_k),M}).$$

With (1.8.4) this gives the proposition:

(1.8.6) On the hypothesis of (1.8.5), as k goes to infinity, the ratio of the number of 0's in

$$C(T_{L_M(S_k),M})$$

to its length tends to the limit $\frac{1}{2}$.

The proof of (1.8.5) depends on three facts:

(1.8.7a) There is an effective procedure for coding the table of entries $T_{N,M}$ defining the program of an N -state M -tape-symbol Turing machine together with the two parameters N and M into a single binary sequence $C(T_{N,M})$ of length $(1 + \epsilon_N)N(M - 1) \log_2 N$.

(1.8.7b) Any binary sequence of length not greater than

$$(1 + \epsilon_N)N(M - 1) \log_2 N$$

can be calculated by a suitably programmed N -state M -tape-symbol Turing machine.

(1.8.7c) From a universal Turing machine program it is possible to construct a program for a Turing machine (with a fixed number r of rows) to take $C(T_{N,M})$ and decode it and to then imitate the calculations of the machine whose table of entries $T_{N,M}$ it then knows, until it finally calculates the finite binary sequence S which the program being imitated calculates, if S exists.

(1.8.7a) has just been demonstrated. (1.8.7b) was shown in Section 1.6.

(The concept of a universal program is due to Turing [3].)

The proof of (1.8.5) follows. From (1.8.7a) and (1.8.7b),

$$L_M(C(T_{L_M(S_k),M})) \leq (1 + \epsilon_k)L_M(S_k),$$

and from (1.8.7c) and the hypothesis of (1.8.5),

$$L_M(C(T_{L_M(S_k),M})) + r \geq L_M(S_k).$$

It follows that

$$L_M(C(T_{L_M(S_k),M})) = (1 + \epsilon_k)L_M(S_k),$$

which is—since the length of

$$C(T_{L_M(S_k),M})$$

is

$$(1 + \epsilon_k)L_M(S_k)(M - 1) \log_2 L_M(S_k)$$

and

$$L_M(C_{(1+\epsilon_k)L_M(S_k)(M-1)\log_2 L_M(S_k)}) = (1 + \epsilon'_k)L_M(S_k)$$

—simply the conclusion of (1.8.5).

1.9. The topic of this section is an application of everything that precedes with the exception of Section 1.7 and the first half of Section 1.8. C. E. Shannon suggests [1, p. 165] that the state-symbol product NM is a good measure of the calculating abilities of an N -state M -tape-symbol Turing machine. If one is interested in *comparing* the calculating abilities of *large Turing machines whose M values vary over a finite range*, the results that follow suggest that $N(M - 1)$ is a good measure of calculating abilities. We have as an application of a slight generalization of the ideas used to prove (1.8.5):

(1.9.1a) Any calculation which an N -state M -tape-symbol Turing machine can be programmed to perform can be imitated by any N' -state M' -tape-symbol Turing machine satisfying

$$(1 + \epsilon_N)N(M - 1) \log_2 N < (1 + \epsilon''_N)N'(M' - 1) \log_2 N'$$

if it is suitably programmed.

And directly from the asymptotic formula for $L_M(C_n)$ we have:

(1.9.1b) If

$$(1 + \epsilon_N)N(M - 1) \log_2 N < (1 + \epsilon''_N)N'(M' - 1) \log_2 N',$$

then there exist finite binary sequences which an N' -state M' -tape-symbol Turing machine can be programmed to calculate and which it is impossible to program an N -state M -tape-symbol Turing machine to calculate.

As

$$\frac{(1 + \epsilon_N)N(M - 1) \log_2 N}{((1 + \epsilon'_N)N(M - 1)) \log_2 ((1 + \epsilon'_N)N(M - 1))}$$

and for x and x' greater than one, $x \log_2 x$ is greater (less) than $x' \log_2 x'$ according as x is greater (less) than x' , it follows that the inequalities of (1.9.1a) and (1.9.1b) give the same *ordering* of calculating abilities as do inequalities involving functions of the form $(1 + \epsilon_N)N(M - 1)$.

Part 2

2.1. In this section we return to the Turing machines of Section 1.1 and add to the conventions (1.1A), (1.1B) and (1.1C),

(2.1D) An entry (i, j) in the p th row of the table of a Turing machine must satisfy $|i - p| \leq b$. In addition, while a fictitious state is used (as before) for the purpose of halting, the row of the table for this fictitious state is now considered to come directly after the actual last row of the program.

Here b is a constant whose value is to be regarded as fixed throughout Part 2. In Section 2.2 it will be shown that b can be chosen sufficiently large that the Turing machines thus defined (which we take the liberty of naming “bounded-transfer Turing machines”) have all the calculating capabilities that are basically required of Turing machines for theoretical purposes (e.g., such purposes as defining what one means by “effective process for determining...”), and hence have calculating abilities sufficient for the proofs of Part 2 to be carried out.

(2.1D) may be regarded as a mere convention, but it is more properly considered as a change in the basic philosophy of the logical design of the Turing machine (i.e., the philosophy expressed by A. M. Turing [3, Sec. 9]).

Here in Part 2 there will be little point in considering the general M -tape-symbol machine. It will be understood that we are always speaking of 3-tape-symbol machines.

There is a simple and convenient notational change which can be made at this point; it makes all programs for bounded-transfer Turing machines instantly relocatable (which is convenient if one puts together

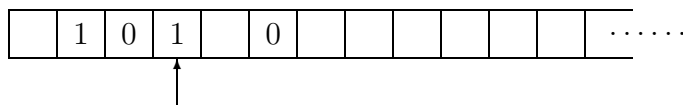
a program from subroutines) and it saves a great deal of superfluous writing. Entries in the tables of machines will from now on consist of ordered pairs (i', j') , where i' goes from $-b$ to b and j' goes from 1 to 5. A “new” entry (i', j') is to be interpreted in terms of the functioning of the machine in a manner depending on the number p of the row of the table it is in; this entry has the same meaning as the “old” entry $(p + i', j')$ used to have.

Thus, halting is now accomplished by entries of the form (k, j) ($1 \leq k \leq b$) in the k th row (from the end) of the table. Such an entry causes the machine to halt after performing the operation indicated by j .

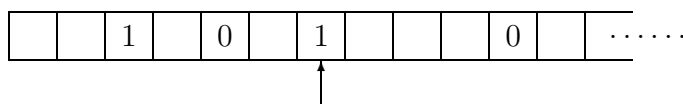
2.2. In this section we attempt to give an idea of the versatility of the bounded-transfer Turing machine. It is here shown in two ways that b can be chosen sufficiently large so that any calculation which one of the Turing machines of Section 1.1 can be programmed to perform can be imitated by a suitably programmed bounded-transfer Turing machine.

As the first proof, b is taken to be the number of rows in a 3-tape-symbol universal Turing machine program for the machines of Section 1.1. This universal program (with its format changed to that of the bounded-transfer Turing machines) occupies the last rows of a program for a bounded-transfer Turing machine, a program which is mainly devoted to writing out on the tape the information which will enable the universal program to imitate any calculation which any one of the Turing machines of Section 1.1 can be programmed to perform. One row of the program is used to write out each symbol of this information (as in the program in Section 1.7), and control passes straight through the program row after row until it reaches the universal program.

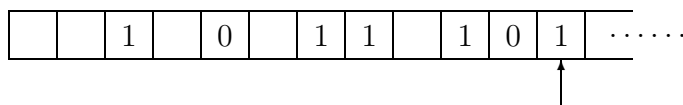
Now for the second proof. To program a bounded-transfer Turing machine in such a manner that it imitates the calculations performed by a Turing machine of Section 1.1, consider alternate squares on the tape of the bounded-transfer Turing machine to be the squares of the tape of the machine being imitated. Thus



is imitated by



After the operation of a state (i.e., write 0, write 1, write blank, shift tape left, shift tape right) has been imitated, as many 1's as the number of the next state to be imitated are written on the squares of the tape of the bounded-transfer Turing machine which are not used to imitate the squares of the other machine's tape, starting on the square immediately to the right of the one on which is the scanner of the bounded-transfer Turing machine. Thus if in the foregoing situation the next state to be imitated is state number three, then the tape of the bounded-transfer Turing machine becomes



The rows of the table which cause the bounded-transfer Turing machine to do the foregoing (type I rows) are interwoven or braided with two other types of rows. The first of these (type II rows) is used for the sole purpose of putting the bounded-transfer Turing machine back in its initial state (row 1 of the table; this row is a type III row). They appear (as do the other two types of rows) periodically throughout the table, and each of them does nothing but transfer control to the preceding one. The second of these (type III rows) serve to pass control back in

the other direction; each time control is about to pass a block of type I rows that imitate a particular state of the other machine while traveling through type III rows, the type III rows erase the rightmost of the 1's used to write out the number of the next state to be imitated. When finally none of these place-marking 1's is left, control is passed to the group of type I rows that was about to be passed, which then proceeds to imitate the appropriate state of the Turing machine of Section 1.1.

Thus the obstacle of the upper bound on the length of transfers in bounded-transfer Turing machines is overcome by passing up and down the table by small jumps, while keeping track of the progress to the desired destination is achieved by subtracting a unit from a count written on the tape just prior to departure.

Although bounded-transfer Turing machines have been shown to be versatile, it is not true that as the number of states goes to infinity, asymptotically 100 percent of the bits required to specify a program also serve to specify the behavior of the bounded-transfer Turing machine.

2.3. In this section the following fundamental result is proved.

(2.3.1) $L(C_n) \sim a^*n$, where a^* is, of course, a positive constant.

First it is shown that there exists an a greater than zero such that:

(2.3.2) $L(C_n) \geq an$.

It is clear that there are exactly

$$((5)(2b + 1))^{3N}$$

different ways of making entries in the table of an N -state bounded-transfer Turing machine; that is, there are

$$2^{((3 \log_2(10b+5))N)}$$

different programs for an N -state bounded-transfer Turing machine. Since a different program is required to have the machine calculate each of the 2^n different binary sequences of length n , it can be seen that an N -state bounded-transfer Turing machine can be programmed to calculate any binary sequence of length n only if

$$(3 \log_2(10b + 5))N \geq n \text{ or } N \geq \frac{n}{3 \log_2(10b + 5)}.$$

Thus one can take $a = (1/(3 \log_2(10b + 5)))$.

Next it is shown that:

$$(2.3.3) \quad L(C_n) + L(C_m) \geq L(C_{n+m}).$$

To do this we present a way of making entries in a table with at most $L(C_n) + L(C_m)$ rows which causes the bounded-transfer Turing machine thus programmed to calculate any particular binary sequence S of length $n + m$. S can be expressed as a binary sequence S' of length n followed by a binary sequence S'' of length m . The table is then formed from two sections which are numbered in the order in which they are encountered in reading from row 1 to the last row of the table. Section I consists of at most $L(C_n)$ rows. It is a program which calculates S' . Section II consists of at most $L(C_m)$ rows. It is a program which calculates S'' . It follows from this construction and the definitions that (2.3.3) holds.

(2.3.2) and (2.3.3) together imply (2.3.1).² This will be shown by a demonstration of the following general proposition:

(2.3.4) Let A_1, A_2, A_3, \dots be an infinite sequence of natural numbers satisfying

$$(2.3.5) \quad A_n + A_m \geq A_{n+m}.$$

Then as n goes to infinity, (A_n/n) tends to a limit from above.

²[As stated in the preface of this book, it is straightforward to apply to LISP the techniques used here to study bounded-transfer Turing machines. Let us define $H_{\text{LISP}}(x)$ where x is a bit string to be the size in characters of the smallest LISP S-expression whose value is the list x of 0's and 1's. Consider the LISP S-expression (APPEND P Q), where P is a minimal LISP S-expression for the bit string x and Q is a minimal S-expression for the bit string y . I.e., the value of P is the list of bits x and P is $H_{\text{LISP}}(x)$ characters long, and the value of Q is the list of bits y and Q is $H_{\text{LISP}}(y)$ characters long. (APPEND P Q) evaluates to the concatenation of the bit strings x and y and is $H_{\text{LISP}}(x) + H_{\text{LISP}}(y) + 10$ characters long. Therefore, let us define $H'_{\text{LISP}}(x)$ to be $H_{\text{LISP}}(x) + 10$. Now H'_{LISP} is subadditive like $L(S)$. The discussion of bounded-transfer Turing machines in this paper and the next therefore applies practically word for word to $H'_{\text{LISP}} = H_{\text{LISP}} + 10$. In particular, let $B(n)$ be the maximum of $H'_{\text{LISP}}(x)$ taken over all n -bit strings x . Then $B(n)/n$ is bounded away from zero, $B(n + m) \leq B(n) + B(m)$, and $B(n)$ is asymptotic to a nonzero constant times n .]

For all n , $A_n \geq 0$, so that $(A_n/n) \geq 0$; that is, $\{(A_n/n)\}$ is a set of reals bounded from below. It is concluded that this set has a greatest lowest bound a^* . We now show that

$$\lim_{n \rightarrow \infty} \frac{A_n}{n} = a^*.$$

Since a^* is the greatest lower bound of the set $\{(A_n/n)\}$, for any e greater than zero there is a d for which

$$(2.3.6) \quad (A_d/d) < a^* + e.$$

Every natural number n can be expressed in the form $n = qd + r$, where $0 \leq r < d$. From (2.3.5) it can be seen that for any $n_1, n_2, n_3, \dots, n_{q+1}$,

$$\sum_{k=1}^{q+1} A_{n_k} \geq A_{(\sum_{k=1}^{q+1} n_k)}.$$

Taking $n_k = d$ ($k = 1, 2, \dots, q$) and $n_{q+1} = r$ in this, we obtain

$$qA_d + A_r \geq A_{qd+r} = A_n,$$

which with (2.3.6) gives

$$qd(a^* + e) = (n - r)(a^* + e) \geq A_n - A_r$$

or

$$(1 - \frac{r}{n})(a^* + e) \geq \frac{A_n}{n} - \frac{A_r}{n},$$

which implies

$$a^* + e \geq \frac{A_n}{n} + \epsilon_n$$

or

$$\limsup_{n \rightarrow \infty} \frac{A_n}{n} \leq a^* + e.$$

Since $e > 0$ is arbitrary, it can be concluded that

$$\limsup_{n \rightarrow \infty} \frac{A_n}{n} \leq a^*,$$

which with the fact that $(A_n/n) \geq a^*$ for all n gives

$$\lim_{n \rightarrow \infty} \frac{A_n}{n} = a^*.$$

2.4. In Section 2.3 an asymptotic formula analogous to a part of Section 1.6 was demonstrated; in this section a result is obtained which completes the analogy. This result is most conveniently stated with the aid of the notation $B(m)$ (where m is a natural number) for the binary sequence which is the numeral representing m in base-two notation (e.g., $B(6) = 110$).

(2.4.1) There exists a constant c such that those binary sequences S of length n satisfying

(2.4.2)

$$\begin{aligned} L(S) \leq & L(C_n) - L(B(L(C_n))) - [\log_2 L(B(L(C_n)))] \\ & - L(C_m) - [\log_2 L(C_m)] - c \end{aligned}$$

are less than 2^{n-m} in number.

The proof of (2.4.1) is by contradiction. We suppose that those S of length n satisfying (2.4.2) are 2^{n-m} or more in number and we conclude that for any particular binary sequence S^\sim of length n there is a program of at most $L(C_n) - 1$ rows that causes a bounded-transfer Turing machine to calculate S^\sim . This table consists of 11 sections which come one after the other. The first section consists of a single row which moves the tape one square to the left (1,4 1,4 1,4 will certainly do this). The second section consists of exactly $L(B(L(C_n)))$ rows; it is a program for computing $B(L(C_n))$ consisting of the smallest possible number of rows. The third section is merely a repetition of the first section. The fourth section consists of exactly $[\log_2 L(B(L(C_n)))]$ rows. Its function is to write out on the tape the binary sequence which represents the number $L(B(L(C_n)))$ in base-two notation. Since this is a sequence of exactly $[\log_2 L(B(L(C_n)))]$ bits, a simple program exists for calculating it consisting of exactly $[\log_2 L(B(L(C_n)))]$ rows each of which causes the machine to write out a single bit of the sequence and then shift the tape a single square to the left (e.g., 0,2 1,4 1,4 will do

for a 0 in the sequence). The fifth section is merely a repetition of the first section. The sixth section consists of at most $L(C_m)$ rows; it is a program consisting of the smallest possible number of rows for computing the sequence S^R of the m rightmost bits of S^\sim . The seventh section is merely a repetition of the first section. The eighth section consists of exactly $\lceil \log_2 L(C_m) \rceil$ rows. Its function is to write out on the tape the binary sequence which represents the number $L(C_m)$ in base-two notation. Since this is a sequence of exactly $\lceil \log_2 L(C_m) \rceil$ bits, a simple program exists for calculating it consisting of exactly $\lceil \log_2 L(C_m) \rceil$ rows each of which causes the machine to write out a single bit of the sequence and then shift the tape a single square to the left. The ninth section is merely a repetition of the first section. The tenth section consists of at most as many rows as the expression on the right-hand side of the inequality (2.4.2). It is a program for calculating one (out of not less than 2^{n-m}) of the sequences of length n satisfying (2.4.2) (which one it is depends on S^\sim in a manner which will become clear from the discussion of the eleventh section; for now we merely denote it by S^L).

We now come to the last and crucial eleventh section, which consists *by definition* of $(c-6)$ rows, and which therefore brings the total number of rows up to at most $1 + L(B(L(C_n))) + 1 + \lceil \log_2 L(B(L(C_n))) \rceil + 1 + L(C_m) + 1 + \lceil \log_2 L(C_m) \rceil + 1 +$ (the expression on the right-hand side of the inequality (2.4.2)) $+ (c-6) = L(C_n) - 1$. When this section of the program takes over, the numbers and sequences $L(C_n), L(B(L(C_n))), S^R, L(C_m), S^L$ are written—in the above order—on the tape. Note, first of all, that section 11 can:

1. compute the value v of the right-hand side of the inequality (2.4.2) from this data,
2. find the value of n from this data (simply by counting the number of bits in the sequence S^L), and
3. find the value of m from this data (simply by counting the number of bits in S^R).

Using its knowledge of v , m and n , section 11 then computes from the sequence S^L a new sequence $S^{L'}$ which is of length $(n-m)$. The manner

in which it does this is discussed in the next paragraph. Finally, section 11 adjoins the sequence S^R to the right of $S^{L'}$, positions this sequence which is in fact S^\sim properly for it to be able to be regarded calculated, cleans up the rest of the tape, and halts scanning the square just to the right of S^\sim . S^\sim has been calculated.

To finish the proof of (2.4.1) we must now only indicate how section 11 arrives at $S^{L'}$ (of length $(n - m)$) from v , m , n , and S^L . (And it must be here that it is made clear how the choice of S^L depends on S^\sim .) By assumption, S^L satisfies

(2.4.3) $L(S^L) \leq v$ and S^L is of length n .

Also by assumption there are at least 2^{n-m} sequences which satisfy (2.4.3). Now section 11 contains a procedure which when given any one of some particular serially ordered set Q_v^n of 2^{n-m} sequences satisfying (2.4.3), will find the ordinal number of its position in Q_v^n . And the number of the position of S^L in Q_v^n is the number of the position of $S^{L'}$ in the natural ordering of all binary sequences of length $(n - m)$ (i.e., 000...00, 000...01, 000...10, 000...11, ..., 111...00, 111...01, 111...10, 111...11). In the next and final paragraph of this proof, the foregoing italicized sentence is explained.

It is sufficient to give here a procedure for serially calculating the members of Q_v^n in order. (That is, we define a serially ordered Q_v^n for which there is a procedure.) By assumption we know that the predicate which is satisfied by all members of Q_v^n , namely,

$$(L(\dots) \leq v) \ \& \ (\dots \text{ is of length } n),$$

is satisfied by at least 2^{n-m} sequences. It should also be clear to the reader on the basis of some background in Turing machine and recursive function theory (see especially Davis [4], where recursive function theory is developed from the concept of the Turing machine) that the set Q of

all natural numbers of the form $2^n 3^v 5^e$, where e is the natural number represented in base-two notation by a binary sequence S satisfying $(L(S) \leq v) \ \& \ (S \text{ is of length } n)$

is recursively enumerable. Let T denote some particular Turing machine which is programmed in such a manner that it recursively enumerates (or, to use E. Post's term, generates) Q . The definition of Q_v^n can now be given:

Q_v^n is the set of binary sequences of length n which represent in base-two notation the exponents of 5 in the prime factorization of the first 2^{n-m} members of Q generated by T whose prime factorizations have 2 with an exponent of n and 3 with an exponent of v , and their order in Q_v^n is the order in which T generates them.

Q.E.D.

It can be proved by contradiction that the set Q is not recursive. For were Q recursive, there would be a program which given any finite binary sequence S would calculate $L(S)$. Hence there would be a program which given any natural number n would calculate the members of C_n . Giving n to this program can be done by a program of length $\lceil \log_2 n \rceil$. Thus there would be a program of length $\lceil \log_2 n \rceil + c$ which would calculate an element of C_n . But we know that the shortest program for calculating an element of C_n is of length $\sim a^*n$, so that we would have for n sufficiently large an impossibility.

It should be emphasized that if $L(C_n)$ is an effectively computable function of n then the method of this section yields the far stronger result: There exists a constant c such that those binary sequences S of length n satisfying $L(S) \leq L(C_n) - L(C_m) - c$ are less than 2^{n-m} in number.³

2.5. The purpose of this section is to investigate the behavior of the right-hand side of (2.4.2). We start by showing a result which is stronger for n sufficiently large than the inequality $L(C_n) \leq n$, namely, that the constant a^* in the asymptotic evaluation $L(C_n) \sim a^*n$ of Section 2.3 is less than 1. This is done by deriving:

³[For LISP we also obtain this much neater form of result that most n -bit strings have close to the maximum complexity H_{LISP} . The reason is that by using EVAL a quoted LISP S-expression tells us its size as well as its value. In other words, $H_{\text{LISP}}(x) = H_{\text{LISP}}(x, H_{\text{LISP}}(x)) + O(1)$.]

(2.5.1) For any s there exist n and m such that

$$L(C_s) \leq L(C_n) + L(C_m) + c,$$

and $(n + m)$ is the smallest integral solution x of the inequality

$$s \leq x + \lceil \log_2 x \rceil - 1.$$

From (2.5.1) it will follow immediately that if $e(n)$ denotes the function satisfying $L(C_n) = a^*n + e(n)$ (note that by Section 2.3 $(e(n)/n)$ tends to 0 from above as n goes to infinity), then for any s , $L(C_s) \leq L(C_n) + L(C_m) + c$ for some n and m satisfying $(n + m) = s - (1 + \epsilon_s) \log_2 s$, which implies

$$a^*s \leq a^*(s - (1 + \epsilon_s) \log_2 s) + e(n) + e(m)$$

or

$$(a^* + \epsilon_s) \log_2 s \leq e(n) + e(m).$$

Hence as n and m are both less than s and at least one of $e(n)$, $e(m)$ is greater than $\frac{1}{2}(a^* + \epsilon_s) \log_2 s$, there are an infinity of n for which $e(n) \geq \frac{1}{2}(a^* + \epsilon_n) \log_2 n$. That is,

$$(2.5.2) \quad \limsup \frac{L(C_n) - a^*n}{a^* \log_2 n} \geq \frac{1}{2}.$$

From (2.5.2) with $L(C_n) \leq n$ follows immediately

$$(2.5.3) \quad a^* < 1.$$

The proof of (2.5.1) is presented by examples. The notation $T * U$ is used, where T and U are finite binary sequences for the sequence resulting from adjoining U to the right of T . Suppose it is desired to calculate some finite binary sequence S of length s , say $S = 010110010100110$ and $s = 15$. The smallest integral solution x of $s \leq x + \lceil \log_2 x \rceil - 1$ for this value of s is 12. Then S is expressed as $S' * S^T$ where S' is of length $x = 12$ and S^T is of length $s - x = 15 - 12 = 3$, so that $S' = 010110010100$ and $S^T = 110$. Next S' is expressed as $S^L * S^R$ where the length m of S^L satisfies $A * B(m) = S^T$ for some (possibly null) sequence A consisting entirely of 0's, and the length n of S^R is

$x - m$. In this case $A * B(m) = 110$, so that $m = 6$, $S^L = 010110$ and $S^R = 010100$. The final result is that one has obtained the sequences S^L and S^R from the sequence S . And—this is the crucial point—if one is given the S^L and S^R resulting by the foregoing process from some unknown sequence S , one can reverse the procedure and determine S . Thus suppose $S^L = 1110110$ and $S^R = 01110110000$ are given. Then the length m of S^L is 7, the length n of S^R is 11, and the sum x of m and n is $7 + 11 = 18$. Therefore the length s of S must be $s = x + \lceil \log_2 x \rceil - 1 = 18 + 5 - 1 = 22$. Thus $S = S^L * S^R * S^T$, where S^T is of length $s - x = 22 - 18 = 4$, and so from $A * B(m) = S^T$ or $0 * B(7) = S^T$ one finds $S^T = 0111$. It is concluded that

$$S = S^L * S^R * S^T = 1110110011101100000111.$$

(For x of the form 2^h what precedes is not strictly correct. In such cases s may equal the foregoing indicated quantity or the foregoing indicated quantity minus one. It will be indicated later how such cases are to be dealt with.)

Let us now denote by F the function carrying (S^L, S^R) into S , and by F_R^{-1} the function carrying S into S^R , defining F_L^{-1} similarly. Then for any particular binary sequence S of length s the following program consists of at most

$$1 + L(F_L^{-1}(S)) + 1 + L(F_R^{-1}(S)) + 2 + (c - 4) \leq L(C_n) + L(C_m) + c$$

rows with $m + n = x$ being the smallest integral solution of $s \leq x + \lceil \log_2 x \rceil - 1$.

<p><i>Section I:</i> 1,4 1,4 1,4</p>
<p><i>Section II</i> consists of $L(F_L^{-1}(S))$ rows. It is a program with the smallest possible number of rows for calculating $F_L^{-1}(S)$.</p>
<p><i>Section III:</i> 1,4 1,4 1,4</p>
<p><i>Section IV</i> consists of $L(F_R^{-1}(S))$ rows. It is a program with the smallest possible number of rows for calculating $F_R^{-1}(S)$. (Should x be of the form 2^h, another section is added at this point to tell Section V which of the two possible values s happens to have. This section consists of two rows; it is either 1,4 1,4 1,4 1,2 1,2 1,2 or 1,4 1,4 1,4 1,3 1,3 1,3.)</p>
<p><i>Section V</i> consists of $c - 4$ rows, by definition. It is a program that is able to compute F. It computes $F(F_L^{-1}(S), F_R^{-1}(S)) = S$, positions S properly on the tape, cleans up the rest of the tape, positions the scanner on the square just to the right of S, and halts.</p>

As this program causes S to be calculated, the proof is easily seen to be complete.

The second result is:

(2.5.4) Let $f(n)$ be any effectively computable function that goes to infinity with n and satisfies $f(n+1) - f(n) = 0$ or 1 . Then there are an infinity of distinct n_k for which $L(B(L(C_{n_k}))) < f(n_k)$.

This is proved from (2.5.5), the proof being identical with that of (1.7.5).

(2.5.5) For any positive integer p there is at least one solution n of $L(C_n) = p$.

Let the n_k satisfy $L(C_{n_k}) = f^{-1}(k)$, where $f^{-1}(k)$ is defined to be the smallest value of j for which $f(j) = k$. Then since $L(C_n) \leq n$, $f^{-1}(k) \leq n_k$. Noting that f^{-1} is an effectively computable function, it is easily seen that

$$L(B(L(C_{n_k}))) = L(B(f^{-1}(k))) \leq L(B(k)) + c \leq [\log_2 k] + c.$$

Hence, for all sufficiently large k ,

$$L(B(L(C_{n_k}))) \leq [\log_2 k] + c < k = f(f^{-1}(k)) \leq f(n_k).$$

Q.E.D.

(2.5.4) and (2.4.1) yield:

(2.5.6) Let $f(n)$ be any effectively computable function that goes to infinity with n and satisfies $f(n+1) - f(n) = 0$ or 1 . Then there are an infinity of distinct n_k for which less than $2^{n_k - f(n_k)}$ binary sequences S of length n_k satisfy $L(S) \leq L(C_{n_k}) - (a^* + \epsilon_k)f(n_k)$.

Part 3

3.1. Consider a scientist who has been observing a closed system that once every second either emits a ray of light or does not. He summarizes his observations in a sequence of 0's and 1's in which a zero represents "ray not emitted" and a one represents "ray emitted." The sequence may start

0110101110...

and continue for a few thousand more bits. The scientist then examines the sequence in the hope of observing some kind of pattern or law. What does he mean by this? It seems plausible that a sequence of 0's and 1's is patternless if there is no better way to calculate it than just

by writing it all out at once from a table giving the whole sequence:

My Scientific Theory

0
1
1
0
1
0
1
1
1
0
⋮

This would not be considered acceptable. On the other hand, if the scientist should hit upon a method by which the whole sequence could be calculated by a computer whose program is short compared with the sequence, he would certainly not consider the sequence to be entirely patternless or random. And the shorter the program, the greater the pattern he might ascribe to the sequence.

There are many genuine parallels between the foregoing and the way scientists actually think. For example, a simple theory that accounts for a set of facts is generally considered better or more likely to be true than one that needs a large number of assumptions. By “simplicity” is *not* meant “ease of use in making predictions.” For although General or Extended Relativity is considered to be the simple theory par excellence, very extended calculations are necessary to make predictions from it. Instead, one refers to the number of arbitrary choices which have been made in specifying the theoretical structure. One naturally is suspicious of a theory the number of whose arbitrary elements is of an order of magnitude comparable to the amount of information about reality that it accounts for.

On the basis of these considerations it may perhaps not appear entirely arbitrary to define a patternless or random finite binary sequence as a sequence which in order to be calculated requires, roughly speaking, at least as long a program as any other binary sequence of the same

length. A patternless or random infinite binary sequence is then defined to be one whose initial segments are all random. In making these definitions mathematically approachable it is necessary to specify the kind of computer referred to in them. This would seem to involve a rather arbitrary choice, and thus to make our definitions less plausible, but in fact both of the kinds of Turing machines which have been studied by such different methods in Parts 1 and 2 lead to precise mathematical definitions of patternless sequences (namely, the patternless or random finite binary sequences are those sequences S of length n for which $L(S)$ is approximately equal to $L(C_n)$, or, fixing M , those for which $L_M(S)$ is approximately equal to $L_M(C_n)$) whose provable statistical properties start with forms of the law of large numbers. Some of these properties will be established in a paper of the author to appear.⁴

A final word. In scientific research it is generally considered better for a proposed new theory to account for a phenomenon which had not previously been contained in a theoretical structure, before the discovery of that phenomenon rather than after. It may therefore be of some interest to mention that the intuitive considerations of this section antedated the investigations of Parts 1 and 2.

3.2. The definition which has just been proposed⁵ is one of many attempts which have been made to define what one means by a patternless or random sequence of numbers. One of these was begun by R. von Mises [5] with contributions by A. Wald [6], and was brought to its culmination by A. Church [7]. K. R. Popper [8] criticized this definition. The definition given here deals with the concept of a patternless binary sequence, a concept which corresponds roughly in intuitive intent with the random sequences associated with probability half of Church. However, the author does not follow the basic philosophy of the von

⁴The author has subsequently learned of work of P. Martin-Löf ("The Definition of Random Sequences," research report of the Institutionen för Försäkringsmatematik och Matematisk Statistik, Stockholm, Jan. 1966, 21 pp.) establishing statistical properties of sequences defined to be patternless on the basis of a type of machine suggested by A. N. Kolmogorov. Cf. footnote 5.

⁵The author has subsequently learned of the paper of A. N. Kolmogorov, Three approaches to the definition of the concept "amount of information," *Problemy Peredachi Informatsii* [Problems of Information Transmission], 1, 1 (1965), 3–11 [in Russian], in which essentially the definition offered here is put forth.

Mises–Wald–Church definition; instead, the author is in accord with the opinion of Popper [8, Sec. 57, footnote 1]:

I come here to the point where I failed to carry out fully my intuitive program—that of analyzing randomness as far as it is possible within the region of *finite* sequences, and of proceeding to *infinite* reference sequences (in which we need *limits* of relative frequencies) only afterwards, with the aim of obtaining a theory in which the existence of frequency limits follows from the random character of the sequence.

Nonetheless the methods given here are similar to those of Church; the concept of effective computability is here made the central one.

A discussion can be given of just how patternless or random the sequences given in this paper appear to be for practical purposes. How do they perform when subjected to statistical tests of randomness? Can they be used in the Monte Carlo method? Here the somewhat tantalizing remark of J. von Neumann [9] should perhaps be mentioned:

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number—there are only methods to produce random numbers, and a strict arithmetical procedure of course is not such a method. (It is true that a problem that we suspect of being solvable by random methods may be solvable by some rigorously defined sequence, but this is a deeper mathematical question than we can now go into.)

Acknowledgment

The author is indebted to Professor Donald Loveland of New York University, whose constructive criticism enabled this paper to be much clearer than it would have been otherwise.

References

- [1] SHANNON, C. E. A universal Turing machine with two internal states. In *Automata Studies*, Shannon and McCarthy, Eds., Princeton U. Press, Princeton, N. J., 1956.
- [2] —. A mathematical theory of communication. *Bell Syst. Tech. J.* 27 (1948), 379–423.
- [3] TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* {2} 42 (1936–37), 230–265; Correction, *ibid.*, 43 (1937), 544–546.
- [4] DAVIS, M. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.
- [5] VON MISES, R. *Probability, Statistics and Truth*. MacMillan, New York, 1939.
- [6] WALD, A. Die Widerspruchsfreiheit des Kollektivbegriffes der Wahrscheinlichkeitsrechnung. *Ergebnisse eines mathematischen Kolloquiums 8* (1937), 38–72.
- [7] CHURCH, A. On the concept of a random sequence. *Bull. Amer. Math. Soc.* 46 (1940), 130–135.
- [8] POPPER, K. R. *The Logic of Scientific Discovery*. U. of Toronto Press, Toronto, 1959.
- [9] VON NEUMANN, J. Various techniques used in connection with random digits. In *John von Neumann, Collected Works, Vol. V*. A. H. Taub, Ed., MacMillan, New York, 1963.
- [10] CHAITIN, G. J. On the length of programs for computing finite binary sequences by bounded-transfer Turing machines. Abstract 66T-26, *Notic. Amer. Math. Soc.* 13 (1966), 133.
- [11] —. On the length of programs for computing finite binary sequences by bounded-transfer Turing machines II. Abstract 631-6, *Notic. Amer. Math. Soc.* 13 (1966), 228–229. (Erratum, p. 229, line 5: replace “ P ” by “ L ”.)

RECEIVED OCTOBER, 1965; REVISED MARCH, 1966