indices (now 1, 2, 3) or we may have 0, which is the code for "stop" (see the 1-line of Card 3).

The Turing machine operates on a potentially both-ways infinite tape, divided into squares, each of which contains a 0 or 1. At any moment, one of these squares is scanned, and one of the cards is "in control" in the sense that the instructions on that card are to be executed.

The figure below shows a situation where Card 3 is in control and a 0 is scanned (the ... at either end means that all squares not shown contain 0's).

Now let us start on an all-0 tape with its Card 1 the Turing machine described above. We find that we receive the stop instruction after four shifts; the final tape situation is as follows:

Next, consider another 3-card Turing machine given below.

CARD 1			
0	1	1	2
1	1	0	3

CARD 2			
0	1	0	1
1	1	1	2

CARD 3			
0	1	0	2
1	1	1	0

Starting this machine on an all-0 tape with its Card 1, we find that the stop instruction is received after 13 shifts. The final tape situation is

As a last illustration, consider the 3-card Turing machine on the following page.

CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	1	0	3
1	1	1	2

CARD 3			
0	1	0	1
1	1	0	3

Starting this machine on an all-0 tape with its Card 1, we find after a while that the machine fails to reach the situation required for stopping (see the 1-line of Card 1). Now the question is: will this machine ever stop? To get better insight, it is convenient to use the following diagram for the "operating record" of the Turing machine.

0_1
$1\ 0_2$
$1_3\ 1$
$0_3\ 1\ 1$
$0_1\ 1\ 1\ 1$
$1\ 1_2\ 1\ 1$
$1\ 1\ 1_2\ 1$
$1\ 1\ 1\ 1_2$
$1\ 1\ 1\ 1\ 0_2$
$1\ 1\ 1\ 1_3\ 1$
$1\ 1\ 1_3\ 1\ 1$
$1\ 1_3\ 1\ 1\ 1$
$1_3\ 1\ 1\ 1\ 1$
$0_3\ 1\ 1\ 1\ 1\ 1$
$0_1\ 1\ 1\ 1\ 1\ 1\ 1$
$1\ 1_2\ 1\ 1\ 1\ 1\ 1$
$1\ 1\ 1_2\ 1\ 1\ 1\ 1$
$1\ 1\ 1\ 1_2\ 1\ 1\ 1$
$1\ 1\ 1\ 1\ 1_2\ 1\ 1$
$1\ 1\ 1\ 1\ 1\ 1_2\ 1$
$1\ 1\ 1\ 1\ 1\ 1\ 1_2$

This diagram is obtained by showing the successive tape situations individually; it is very suggestive in formulating conjectures about the behavior of a machine. Each row of the diagram shows the tape only to the point (right and left) beyond which the tape contains 0's only. The subscripts in the various squares show the index of the card in control. The previous diagram shows the operating record through the first twenty shifts.

Looking at the operating record, we note that the tape situations which are framed there show a certain similarity; and so we surmise that the machine is in a "loop" and hence will never stop. We shall return to this point later on. For the moment, we merely observe that it may be difficult (or even impossible) to determine by inspection whether or not a given machine will ever stop.

As shown in the preceding discussion, the Turing machine,

CARD 1	
0	1 1 2
1	1 1 3

CARD 2	
0	1 0 1
1	1 1 2

CARD 3	
0	1 0 2
1	1 1 0

(started on an all-0 tape with its Card 1) prints two ones on the tape by the time it stops. On the other hand, the Turing machine

CARD 1	
0	1 1 2
1	1 0 3

CARD 2	
0	1 0 1
1	1 1 2

CARD 3	
0	1 0 2
1	1 1 0

prints out six 1's by the time it stops. The following problem arises: consider, for a fixed positive integer n, the class K_n of all the n-card binary Turing machines (with the card format described above). Let M be a Turing machine in this class K_n. Start M, with its Card 1, on an all-0 tape. If M stops after a while, then M is termed a "valid entry" in the BB-n

contest (the n-card classification of the Busy Beaver logical game), and its score $\sigma(m)$ is the number of 1's remaining on the tape at the time it stops. Since K_n is a finite class (the number of n-card binary Turing machines is easily seen to be $[4(n+1)]^{2n}$, the number of valid entries in the BB-n contest is also finite. Hence, the scores of these valid entries constitute a non-empty finite set of non-negative integers, and thus this set has a (unique) largest element which we denote by $\sum(n)$, to stress that this largest element depends upon the card-number n. It is practically trivial that this function $\sum(n)$ is not general recursive (see T. Rado [2], [3]). On the other hand, it may be possible to determine the value of $\sum(n)$ for particular values of n. It has been conjectured that $\sum(3) = 6$. The problem mentioned above is to decide whether or not this conjecture is valid.

The solution of this quite special problem was attempted by several competent mathematicians and programmers, by means of increasingly elaborate computer programs. The first definite solution is contained in the present work. After some experimenting, one will readily observe that the crux of the matter is, for any card number n, the determination of the function SH(n) defined as follows. Each valid entry M in the BB-n contest performs a certain number s(M) of shifts by the time it stops; the function SH(n) is the maximum of s(M) for all valid entries in the BB-n contest. As shown in [2], the function SH(n) is not general recursive either. However, if for some particular value of n the value of SH(n) can be determined, then for the same value of n the value of $\sum(n)$ can also be effectively determined. Indeed, we merely run each n-card machine (starting with Card 1 on an all-0 tape) through not more than SH(n) shifts; we note the scores of those that stop, and the largest one of these scores is then $\sum(n)$.

On the basis of extensive computer experiments, it has been conjectured that SH(3) = 21; and a 3-card Turing machine that shifted 21 times by the time it stopped has been found. In the present work, we verify that this conjecture is also valid.

As observed in Chapter I, the main objective in the present study of such very special issue is to throw light upon the controversial issue of the "effective computability" of individual well-defined integers. It is our hope that this study will suggest further work on this basic issue.

CHAPTER III

THE METHOD

The total number of 3-card Turing machines can easily be seen to be $[4(3+1)]^6$ or about 17 million. We reduce this number by proper normalization (see next section for details) to 82,944 which is then divided into four lots. For each lot, our computer program first generates the machines and stores their conveniently coded descriptions in a table which we call the machine table. Then the program finds and discards those machines that stop in not more than twenty-one shifts and at the same time takes note of their scores and shift numbers (when they stop). The list of the machines that were not discarded is then scrambled up in the machine table and the first fifty are printed out (the purpose is to enable us to observe the behavior patterns of the undecided machines). Their operating records are then made up and each is examined for some pattern of behavior indicating that the particular machine considered will never stop. From these, we observed a certain recurrence pattern (called below the partial recurrence) which we programmed. As a matter of luck, it turned out that this simple recurrence pattern disposed of all but forty of the machines. When the operating records of the forty "holdouts" were examined, it turned out that they all showed patterns (to be discussed below) which enabled us to decide that all the forty "holdouts" were never-stoppers. We may stress here a certain point of interest. Even though only forty "holdouts" were left, it was not clear a priori that they can be decided as to whether

they are never-stoppers or not, for a given machine may exhibit such a bizarre operating record or exhibit patterns that occur only after a prohibitive number of shifts that no human being could be expected to decide that it will never stop. It is also entirely conceivable that we may have on our hands a machine which is undecidable for some logical reason. Luckily this did not happen in this particular case. In this manner it was established that those machines that stopped at all stopped in no more than twenty-one shifts. Since the program showed us a stopper in twenty-one shifts, we conclude that SH(3) = 21 and the BB-3 problem was solved.

We now proceed to some details of our work.

The four lots

The number of binary 3-card Turing machines is (see above) $(4 \cdot 4)^6 = 2^{24} = 16,777,216$. However, in searching for the actual values of $\sum(3)$ and SH(3), it is sufficient to consider a subset of these machines, obtained by the following considerations. First, let us observe that all the 3-card machines are of the form

(1)

CARD 1	
0	$p_{10}{}^s{}_{10}{}^c{}_{10}$
1	$p_{11}{}^s{}_{11}{}^c{}_{11}$

CARD 2	
0	$p_{20}{}^s{}_{20}{}^c{}_{20}$
1	$p_{21}{}^s{}_{21}{}^c{}_{21}$

CARD 3	
0	$p_{30}{}^s{}_{30}{}^c{}_{30}$
1	$p_{31}{}^s{}_{31}{}^c{}_{31}$

where $p_{ij} = 0$ or 1, $s_{ij} = 0$ or 1, $c_{ij} = 0$ or 1 or 2 or 3. Now consider one of these machines; denote it by M_0. Suppose M_0 is a valid BB-3 entry, with a score $\sigma(M_0)$ and shift number $s(M_0)$. Let $M_0{}^*$ be the "mirror image" of M_0; that is, the machine obtained by replacing (in the cards for M_0) each right shift by a left shift and each left shift by a right shift. Evidently, $M_0{}^*$ is again a valid BB-3 entry, and $\sigma(M_0{}^*) = \sigma(M_0)$, $s(M_0{}^*) = s(M_0)$. Accordingly, we can restrict ourselves to consider those 3-card machines for which

(2) $s_{10} = 1$.

Next, we note that if M_0 is a valid entry such that

(3) $p_{10} = p_{20} = p_{30} = 0$,

then clearly $\sigma(M_0) = 0$ and $s(M_0) \leqq 3$. Since we know that $\sum(3) \geqq 6$ and $SH(3) \geqq 21$, such a machine can be disregarded in searching for the actual value of $\sum(3)$ and $SH(3)$. Accordingly, it is sufficient to consider 3-card machines for which at least one of p_{10}, p_{20}, p_{30} is equal to one. It is also clear that if such a machine M_0 is a valid BB-3 entry, then before M_0 stops, a card C_j with $p_{j0} = 1$ must have been used if the situation $\sigma(M_0) = 0$, $s(M_0) \leqq 3$ is to be avoided. Now let C_i be the card of M_0 which is in control when M_0 first overprints a 1; then $p_{i0} = 1$. Let M_0' be the machine obtained from M_0 by renumbering the cards of M_0 (and adjusting the call instructions c_{ij}) so that the original card C_i is re-named C_1. Clearly $\sigma(M_0') = \sigma(M_0)$, and $s(M_0) \leqq s(M_0') + 2$. After this modification, we can assume that

(4) $p_{10} = 1$.

Next, if we have now $c_{10} = 0$, then clearly $\sigma(M_0) = 1$, $s(M_0) = 1$; hence any machine with $c_{10} = 0$ can be disregarded. Since then $c_{10} \neq 0$, by renumbering the cards 2 and 3 of M_0 (and adjusting the call numbers c_{ij}), we can assume that

(5) $c_{10} = 2$.

Finally, if now $c_{20} = 0$, then clearly $\sigma(M_0) \leqq 2$, $s(M_0) = 2$. Hence, the machines with $c_{20} = 0$ can be disregarded. In view of (2), (4), (5) we can therefore assume that

(6) $p_{10} = 1$, $s_{10} = 1$, $c_{10} = 2$, $c_{20} \neq 0$,

without changing the actual value of $\sum(3)$. As regards $SH(3)$, it is clear from the preceding comments that on denoting by $SH^*(3)$ the maximum of $s(M)$ for valid BB-3 entries normalized in the manner shown in (6), then $SH(3) \leqq SH^*(3) + 2$.

Next, let M_0 be a valid BB-3 entry. Even though there may be several "stop-lines" in the cards for M_0, clearly only one of the several stop instructions will actually be used. Accordingly, we can assume that exactly one of c_{11}, c_{21}, c_{30}, c_{31} is equal to zero. Furthermore, the shift instruction in the unique stop-line of M_0 does not affect either $\sigma(M_0)$ or $s(M_0)$; hence we can assume that the stop-line orders a right shift. Finally, if we specify that the stop-line should issue the "overprint by 1" instruction, then clearly we do not diminish $\sigma(M_0)$. Hence, we can assume that the stop-line has the form 1 1 0. Now the unique stop-line may occur in just four locations; namely, as the 1-line of Card 1, or as the 1-line of Card 2, or the 0-line or 1-line of Card 3. It follows that the machines that we have to investigate can be classified into four lots as follows:

Lot 1

CARD 1	
0	1 1 2
1	1 1 0

CARD 2	
0	$P_{20}{}^{s}{}_{20} \neq 0$
1	$P_{21}{}^{s}{}_{21} \neq 0$

CARD 3	
0	$P_{30}{}^{s}{}_{30} \neq 0$
1	$P_{31}{}^{s}{}_{31} \neq 0$

Lot 2

CARD 1	
0	1 1 2
1	$P_{11}{}^{s}{}_{11} \neq 0$

CARD 2	
0	$P_{20}{}^{s}{}_{20} \neq 0$
1	1 1 0

CARD 3	
0	$P_{30}{}^{s}{}_{30} \neq 0$
1	$P_{31}{}^{s}{}_{31} \neq 0$

Lot 3

CARD 1	
0	1 1 2
1	$P_{11}{}^{s}{}_{11} \neq 0$

CARD 2	
0	$P_{20}{}^{s}{}_{20} \neq 0$
1	$P_{21}{}^{s}{}_{21} \neq 0$

CARD 3	
0	1 1 0
1	$P_{31}{}^{s}{}_{31} \neq 0$

Lot 4

CARD 1	
0	1 1 2
1	$P_{11}{}^{s}{}_{11} \neq 0$

CARD 2	
0	$P_{20}{}^{s}{}_{20} \neq 0$
1	$P_{21}{}^{s}{}_{21} \neq 0$

CARD 3	
0	$P_{30}{}^{s}{}_{30} \neq 0$
1	1 1 0

A simple computation shows that the number of machines in each one of these lots is equal to 20,736. Thus (as far as $\sum(3)$ is concerned) it is sufficient to investigate the $4 \cdot 20,736 = 82,944$ machines contained in the four lots. As regards SH(3), a little more work is involved; we shall return to this point later.

In the next section, we proceed to outline the procedures we followed in treating these four lots.

Description of the computer program

Each individual Turing machine is identified for the purpose of the program as follows. Each line of the Turing card is coded into a four bit binary word (with the "call" instruction occupying two bits). They are then packed in sequence from the 0-line of Card 1, 1-line of Card 1, to the 1-line of Card 3 into a single machine word. This enables us to identify each machine in terms of a single word. For example, the machine

CARD 1	
0	1 1 2
1	1 1 3

CARD 2	
0	1 0 3
1	1 1 0

CARD 3	
0	1 1 1
1	0 0 2

is coded as

1	1	1	0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	1	0	0	1	0

For convenience we also use the octal representation of this binary number in referring to the Turing machine. Thus we identify the above machine also by its "serial number" 73736322. Since the number of machines in each lot is still too large to code by hand, we generate these machines in our computer program by a generalized counting process and store them in a machine table. For each machine in a fixed lot, we have two fixed lines, namely the 0-line of Card 1, 112 (coded 1110) and the stop-line 110 (coded 1100)

COMPUTER STUDIES OF TURING MACHINE PROBLEMS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Shen Lin, B. Sc., M. A.

******

The Ohio State University
1963

Approved by

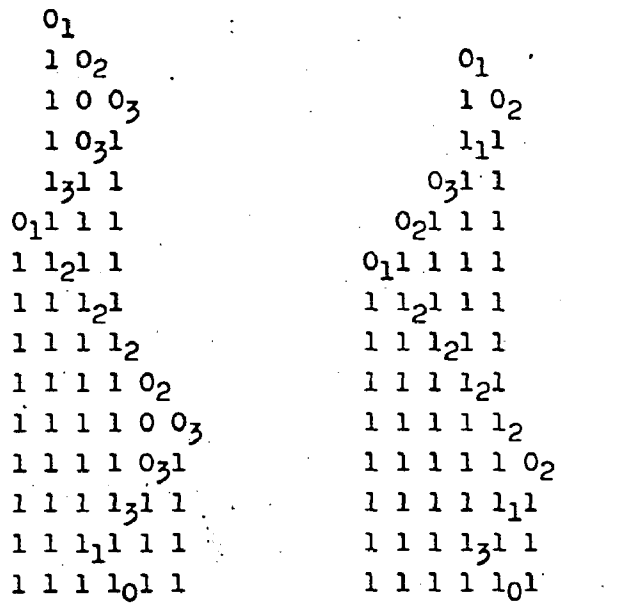
Tibor Rado

Adviser
Department of Mathematics

which occupy the same bit positions in every Turing machine coded in the lot. These are set up first in the storage locations assigned for the machine table. Each of the four other lines can have twelve possible cases. The program sets up these twelve cases of one line in the corresponding bit locations and "or's" them into the machine table consecutively, repeating this procedure 1728 times. Then the second line is set up, this time with each case repeated twelve times and the whole configuration of 144 entries repeatedly or'ed into the machine table 144 times. The third line is set up with each case first repeated 144 times and the whole configuration of 1728 entries repeatedly or'ed into the machine table twelve times. Finally, the last line is set up with each of the twelve possibilities repeated 1728 times and or'ed into the machine table. In this way all possible machines in a lot are obtained and their coded descriptions in the machine table are now ready for examination.

Previous work on the BB-3 problem led to the conjecture that $SH(3) = 21$. We therefore simulate the operation of each Turing machine in the four lots through twenty-one shifts in our computer. If a machine stops in less than or equal to twenty-one shifts, its shift-number and score are noted in a table and the machine is then discarded. It is our hope that we can show later that all those machines that do not stop in less than or equal to twenty-one shifts will never stop. Furthermore, descriptions of machines that score six (or more) or shifted twenty or twenty-one times are printed out. The statistics collected reveals the following: in all four lots, we have 26,073 stoppers in less than or equal to twenty-one shifts (out of a total of 82,944), five machines which scored six, one machine which shifted twenty-one times and two machines that shifted twenty times (see Figures 1 and 2 for their descriptions).

In order to reduce further the machine table size, we discard all machines in Lot 1 with no 1's in the "call" positions of Cards 2

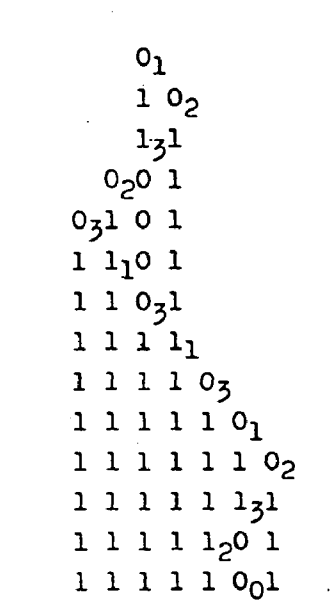


14 shifts

	C_1	C_2	C_3
0	1 1 2	0 1 3	1 0 3
1	1 1 0	1 1 2	1 0 1

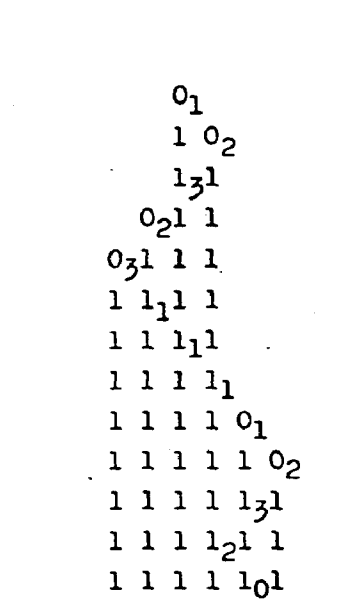
13 shifts

	C_1	C_2	C_3
0	1 1 2	1 0 1	1 0 2
1	1 0 3	1 1 2	1 1 0



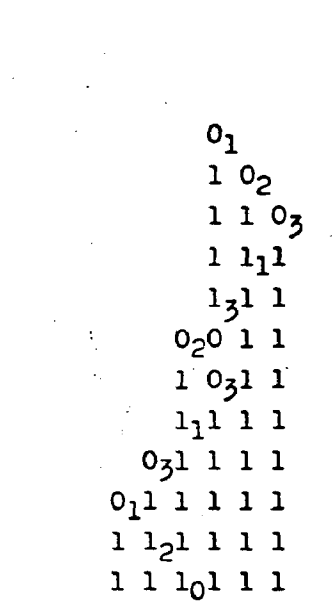
13 shifts

	C_1	C_2	C_3
0	1 1 2	1 0 3	1 1 1
1	1 1 3	1 1 0	0 0 2



12 shifts

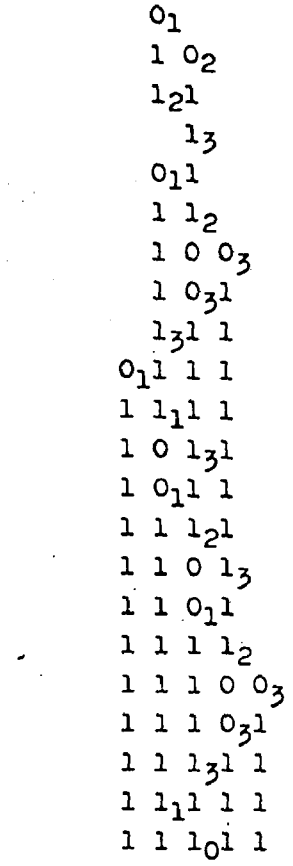
	C_1	C_2	C_3
0	1 1 2	1 0 3	1 1 1
1	1 1 1	1 1 0	1 0 2



11 shifts

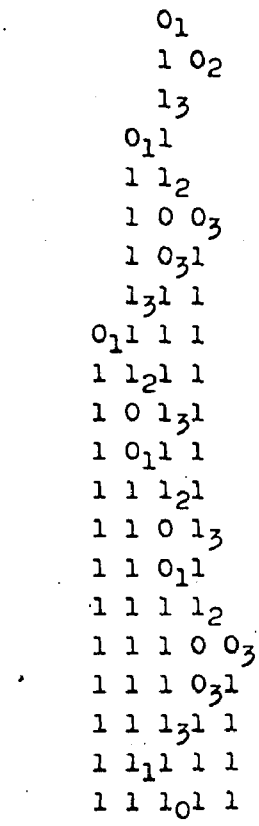
	C_1	C_2	C_3
0	1 1 2	1 1 3	1 0 1
1	1 0 3	1 1 0	0 0 2

Figure 1. -- Score champs and their operating records.



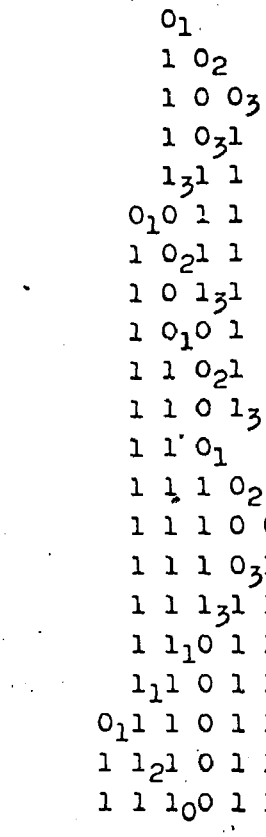
	C_1	C_2	C_3
0	1 1 2	1 0 2	1 0 3
1	1 1 0	0 1 3	1 0 1

The 21 shifter



	C_1	C_2	C_3
0	1 1 2	0 0 3	1 0 3
1	1 1 0	0 1 3	1 0 1

20 shifter



	C_1	C_2	C_3
0	1 1 2	0 1 3	1 0 3
1	1 0 1	1 1 0	0 0 1

20 shifter

Figure 2. -- High shifters and their operating records.

and 3, and all machines in lots 3 and 4 with no 3's in the "call" positions of Cards 1 and 2. These are obvious never-stoppers since the stop-lines can not be reached. In all four lots, 27,774 of these machines are discarded.

The next step in the investigation is to discard those never-stoppers which exhibit a recurrence pattern. The idea may be described briefly as follows. Suppose we operate a given Turing machine M and observe that Card i scans a tape square S_m containing the digit d after m shifts. Later, suppose the same Card i scans a square S_n containing the same digit d after n shifts. If, relative to the scanned squares S_m and S_n, the tape conditions in both instances are identical, it is clear that the same pattern of operation must repeat from then on and hence the Turing machine M is a never-stopper. We call this a "total recurrence" (see Figure 3). Further analysis reveals that we need not have to consider the total tape conditions in most cases. Suppose the square S_n is to the right of the square S_m and that, during the operation from m shifts to n shifts, the leftmost square scanned is S, which is, say k squares to the left of the square S_m. We call the square which is $k + 1$ squares to the left of S_m the "left barrier" relative to S_m. Similarly, the left barrier relative to S_n will be the square which is $k + 1$ squares to the left of the square S_n. It is clear then that if the tape conditions to the right of the left barrier relative to S_m after m shifts is identical to the tape condition to the right of the left barrier relative to S_n after n shifts, the same sequence of operations must repeat and the Turing machine M will never stop. We call this a partial recurrence pattern.

As an illustration, consider the Turing machine and its operating record in Figure 4. Card 2 scans a 1 after twelve shifts and again a 1 after nineteen shifts, during which the portion of the

Turing machine

CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	0	1	3
1	1	0	2

CARD 3			
0	1	0	1
1	0	1	2

Operating record

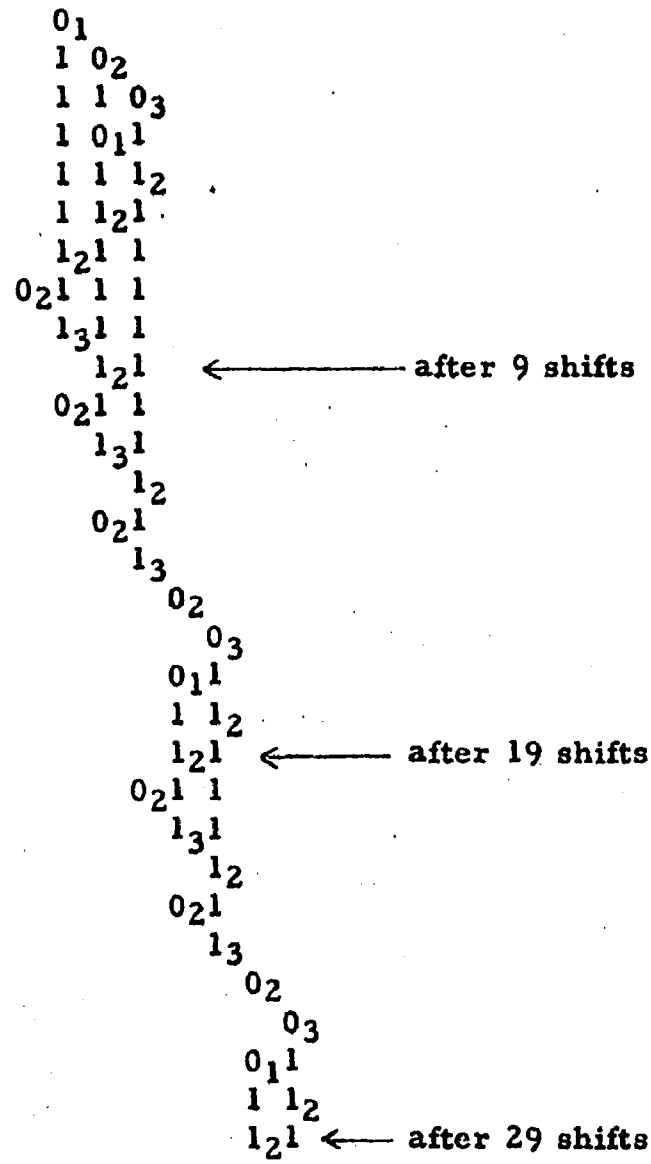


Figure 3. -- Operating record of the Turing machine whose serial number is 73075226 (octal) showing the total recurrence pattern.

Turing machine

CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	1	0	2
1	0	0	3

CARD 3			
0	1	0	1
1	1	1	1

Operating record

0_1
$1\ 0_2$
$1_2 1$
$0_3 0\ 1$
$0_1 1\ 0\ 1$
$1\ 1_2 0\ 1$
$1_3 0\ 0\ 1$
$1\ 0_1 0\ 1$
$1\ 1\ 0_2 1$
$1\ 1_2 1\ 1$
$1_3 0\ 1\ 1$
$1\ 0_1 1\ 1$
$1\ 1\ 1_2 1$ ⟵ after 12 shifts
$1\ 1_3 0\ 1$
$1\ 1\ 0_1 1$
$1\ 1\ 1\ 1_2$
$1\ 1\ 1_3$
$1\ 1\ 1\ 0_1$
$1\ 1\ 1\ 1\ 0_2$
$1\ 1\ 1\ 1_2 1$ ⟵ after 19 shifts
$1\ 1\ 1_3 0\ 1$
$1\ 1\ 1\ 0_1 1$
$1\ 1\ 1\ 1\ 1_2$
$1\ 1\ 1\ 1_3$
$1\ 1\ 1\ 1\ 0_1$
$1\ 1\ 1\ 1\ 1\ 0_2$

Figure 4. -- Operating record of the Turing machine whose serial number is 73121635 (octal) showing the partial recurrence with left barrier.

tape to the right of the left barrier relative to S_{19}, we see that the same sequence of operations must repeat from nineteen to twenty-six shifts, and so on, progressing to the right. It is obvious therefore that this machine will never stop.

If S_n is to the left of S_m, we may consider a right barrier similar to the left barrier described above. An illustration of this case is given in Figure 5.

If S_n happens to be the same square as S_m, we may use both the right barrier and the left barrier. If the portion of the tape between the right and the left barriers after m shifts is identical to that after n shifts, then a recurrence must appear and the machine will never stop.

Next, we construct a computer routine to discard never-stoppers showing the recurrence patterns described above. For the Turing tape we use a machine word of thirty-six bits with each bit representing a square and the starting square at bit 18. We further identify the squares on the tape by their "deviation" from the starting square: the starting square has deviation 0, the square to the right of the starting square has deviation 1, the square to the left of the starting square has deviation -1, and so on. Thus a square with a deviation D is represented by the bit 18 + D. After each shift, the tape condition T, herein represented by a single machine word of thirty-six bits, is stored in an appropriate tape table TB_{ij} corresponding to the card index i called and the digit j in the scanned square. The shift-number at that time and the deviation of the scanned square are also stored in the accompanying tables. Meanwhile the deviations of the scanned square after each shift are further stored in another table (called the deviation table), so that the maximum deviation D_{MAX} and the minimum deviation D_{MIN} may be determined for any portion of the operation of the Turing machine, say between S_1 shifts and S_2 shifts. This is to find out how far to

Turing machine

CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	1	0	3
1	1	1	1

CARD 3			
0	1	0	1
1	0	0	3

Operating record

0_1
$1\ 0_2$
$1_3 1$
$0_3 0\ 1$
$0_1 1\ 0\ 1$
$1\ 1_2 0\ 1$
$1\ 1\ 0_1 1$
$1\ 1\ 1\ 1_2$
$1\ 1\ 1\ 1\ 0_1$
$1\ 1\ 1\ 1\ 1\ 0_2$
$1\ 1\ 1\ 1\ 1_3 1$
$1\ 1\ 1\ 1_3 0\ 1$
$1\ 1\ 1_3 0\ 0\ 1$
$1\ 1_3 0\ 0\ 0\ 1$
$1_3 0\ 0\ 0\ 0\ 1$
$0_3 0\ 0\ 0\ 0\ 0\ 1$
$0_1 1\ 0\ 0\ 0\ 0\ 0\ 1$
$1\ 1_2 0\ 0\ 0\ 0\ 0\ 1$
$1\ 1\ 0_1 0\ 0\ 0\ 0\ 1$
$1\ 1\ 1\ 0_2 0\ 0\ 0\ 1$
$1\ 1\ 1_3 1\ 0\ 0\ 0\ 1$
$1\ 1_3 0\ 1\ 0\ 0\ 0\ 1$
$1_3 0\ 0\ 1\ 0\ 0\ 0\ 1$
$0_3 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$
$0_1 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$
$1\ 1_2 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$
$1\ 1\ 0_1 0\ 0\ 1\ 0\ 0\ 0\ 1$
$1\ 1\ 1\ 0_2 0\ 1\ 0\ 0\ 0\ 1$
$1\ 1\ 1_3 1\ 0\ 1\ 0\ 0\ 0\ 1$
$1\ 1_3 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1$
$1_3 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1$

Figure 5. -- Operating record of the Turing machine whose serial number is 73136623 (octal) showing the partial recurrence with right barrier.

the right and to the left the scanning head has moved during this portion of the operation (for use in finding the right and the left barriers). Whenever an entry T is made into a tape table and the tape table was previously non-empty, tests are made for recurrence as follows. If T_0 is a previous entry in the table with associated shift-number s_0 and deviation D_0, and s is the shift-number and D the deviation associated with the present entry T, D_0 and D are compared. If $D_0 < D$, minimum deviation D_{MIN} is determined from the deviation table for the operation between s_0 and s shifts. T_0 is shifted left $18 + D_{MIN}$ bits and T shifted left $18 + D_{MIN} + D - D_0$ bits and compared. If the resulting logical words are equal, the Turing machine operated on is discarded. Otherwise, T is tested against another previous entry in the same tape table TB_{ij} until all previous entries in the tape table TB_{ij} are checked. If no recurrence pattern is found, the Turing machine is given one more shift and the same procedure goes on. Symmetrical procedures hold when $D_0 > D$. If $D_0 = D$, both D_{MAX} and D_{MIN} are determined and T_0 and T are compared from bits $D + D_{MIN}$ to $D + D_{MAX}$ by the use of a mask.

A bound of fifty is set for the shift-number with a check for spill provided whenever the magnitude of the deviation exceeds seventeen. This is to insure that the portion of the tape scanned can be contained entirely in a single machine word; and the both-ways infinite portions of the tape to the right and to the left of the squares represented by the thirty-six bit machine word which have never been scanned can therefore be assumed to contain all 0's in all instances. If the machine does not show the recurrence pattern after fifty shifts, it is retained in the machine table and printed out later as a "holdout".

The results of this modest effort were quite unexpected.

In all four lots, only forty holdouts were left. That these forty holdouts are all never-stoppers will be shown in the next section. In Figure 6 below, we give the descriptions of these forty holdouts in terms of their octal "serial numbers".

Lot 1	Lot 2	Lot 3	Lot 4
73037233	73676261	70537311	70513754
73137233	73736122	70636711	70612634
73137123	71536037	70726711	70712634
73136523	73336333	72737311	72377034
73133271	71676261	71717312	72377234
73133251	73336133	72211715	72613234
73132742	73236333	72237311	
73132542	73236133	72311715	
73032532		72317716	
73032632		72331715	
73033132		72337311	
73033271		72337315	
73073271			
73075221			

Figure 6. -- The forty holdouts.

ACKNOWLEDGMENT

The writer wishes to express his sincere thanks to his adviser, Univeristy Research Professor Tibor Rado, for his patient help and advice which made this work possible. Also, he wishes to thank Dr. Roy F. Reeves, Director of the Numerical Computation Laboratory, The Ohio State University, for the use of the facilities; and to Miss Jade Lin, for her help in typing the final manuscript.

CHAPTER IV

THE FORTY HOLDOUTS

As stated in Chapter III, there remained forty Turing machines which the computer program failed to eliminate. According to our plan, these forty holdouts were checked by hand, and they were all recognized to be never-stoppers by inspection of their operating records. The Figures 7, 8 and 9 show some typical cases. To illustrate the methods used to show that they are never-stoppers, we discuss in detail two additional cases below.

As our first case, we consider the holdout whose operating record is shown in Figure 10. The cards of this machine are as follows.

CARD 1	
0	1 1 2
1	1 1 0

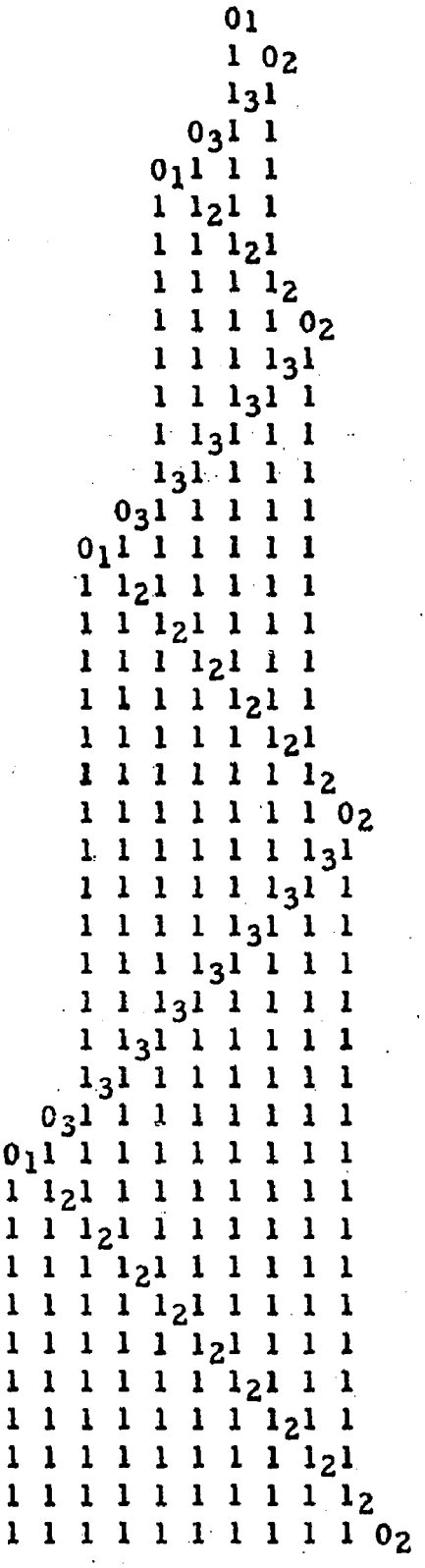
CARD 2	
0	0 0 3
1	1 1 2

CARD 3	
0	1 0 1
1	1 0 3

By inspecting its operating record (Figure 10), we observe that the following tape situation appears repeatedly.

• • •	1	1	1	1	13	0	• • •

This leads to the question of what happens next when we have this type of tape situation. A glance at Card 3 reveals that the string of 1's is first extended to the left by one. Let us use the code name XTNDL for this operation. After this, a left shift is made (code name GOTOL), and control is transferred to Card 1. Card 1 orders printing a 1 over a 0 (MARK); there follows a sequence of shifts to the right, after which control is transferred to Card 3 at the right

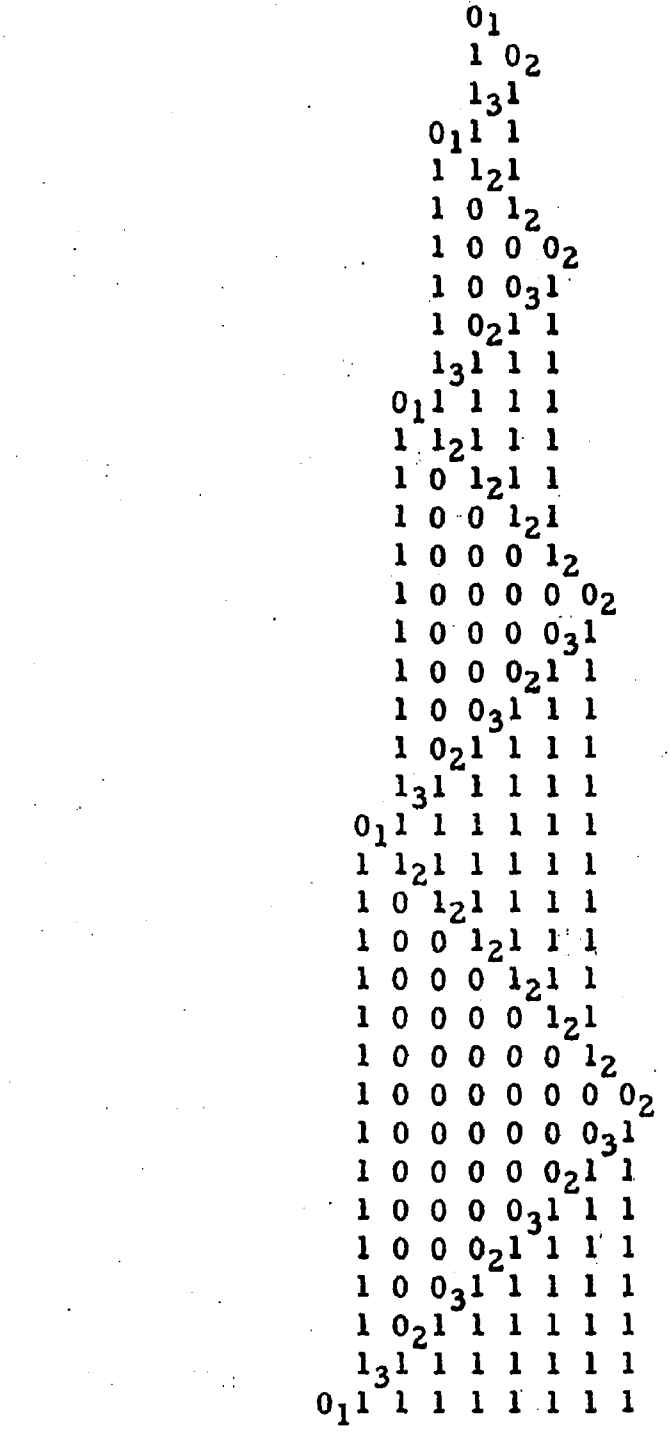


CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	1	0	3
1	1	1	2

CARD 3			
0	1	0	1
1	1	0	3

Figure 7. -- Operating record of the Turing machine whose serial number is 73137233_8.

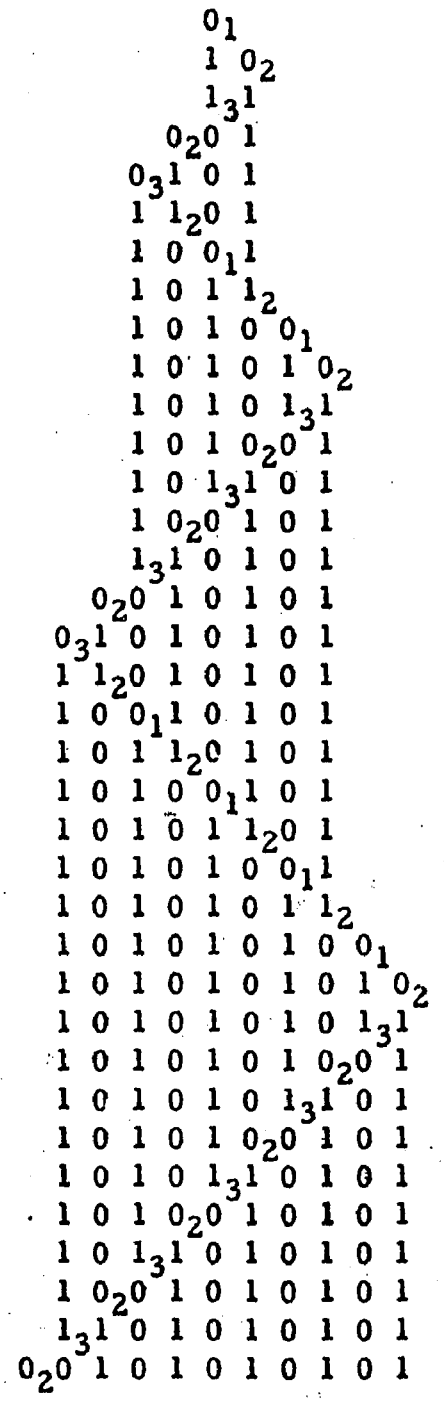


CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	1	0	3
1	0	1	2

CARD 3			
0	1	0	2
1	1	0	1

Figure 8. -- Operating record of the Turing machine whose serial number is 73133251_8 .

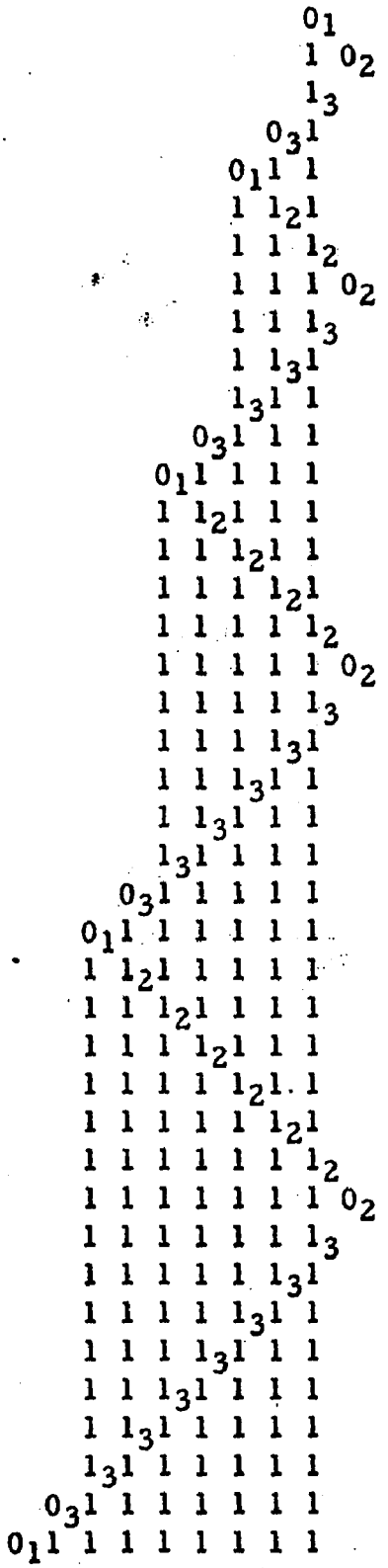


CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	1	0	3
1	0	1	1

CARD 3			
0	1	1	2
1	0	0	2

Figure 9. -- Operating record of the Turing machine whose serial number is 73132742_8.



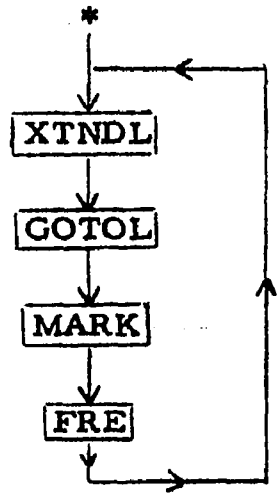
CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	0	0	3
1	1	1	2

CARD 3			
0	1	0	1
1	1	0	3

Figure 10. -- Operating record of the Turing machine whose serial number is 73037233_8.

of the string (FRE for find right end of the string of 1's). This verbal description becomes more intuitive by the use of the following "flowchart", which indicated clearly that the machine has



entered into a loop without exit (the string merely gets longer and longer to the left).

Now let us start this machine, with its Card 1, on an all-0 tape. After the second shift, the tape situation 1_3 arises where the length of the string is one. From this point on the sequence of events is shown by the above flowchart and it is clear that this machine is a never-stopper.

As our second illustration, we consider the holdout with the serial number 73033132_8 and the following card description.

CARD 1			
0	1	1	2
1	1	1	0

CARD 2			
0	0	0	3
1	0	1	2

CARD 3			
0	0	1	1
1	1	0	2

To come to a stop, this machine must get a tape situation where Card 1 scans a 1, $\boxed{}\boxed{}\boxed{1_1}\boxed{}\boxed{}$. Now Card 1 is called only if Card 3 scans a 0; and in this case, to get the stop situation, we should have a 1 to the right, $\boxed{0_3}\boxed{1}\boxed{}\boxed{}\boxed{}$. Now this situation cannot occur. Indeed, Card 3 is called only if Card 2 scans

a 0 and as the 0-line of Card 2 shows, it overprints the square by a 0 and shifts to the left. Hence Card 3 will always scan a square with a 0 to the right. Thus we see that the stop situation is never reached by this machine.

Let us note that the approaches used in these two illustrations involve important ideas ("flowchart" and "back-tracking") of general use in various fields.

CHAPTER V

SH(3) AND MISCELLANEOUS COMMENTS

As mentioned earlier, the results of our efforts were quite unexpected. Originally, 3-card Turing machines showing the induction patterns of the "holdouts" were discovered and programs were devised to eliminate them. This approach proved to be difficult and of little use, since only a few could be eliminated. However, if one should attempt to settle the BB-4 or the BB-5 problem, efficient programs to eliminate the Turing machines showing these patterns must be devised since they will be necessarily too numerous to check by hand. Also, new patterns must show up for increasing card numbers, since we know that $\sum(n)$ is non-computable [2].

Concerning the conjecture that SH(3) = 21, we note that $SH(3) \leqq SH^*(3) + 2$, where $SH^*(3)$ is the maximum of $s(M)$ for valid BB-3 entries M normalized in the manner discussed in Chapter III. We find from our work that $SH^*(3) = 21$, so that $SH(3) \leqq 23$. However, if there is a valid BB-3 entry M with $s(M) \geqq 22$, then upon renumbering the cards of M (readjusting the calling indices and considering a mirror image if necessary), we must have a normalized valid BB-3 entry M* in our four lots with either (i) $s(M^*) = 21$ and at least one of the entries p_{10}, p_{20}, p_{30} equal to zero; or (ii) $s(M^*) = 20$ and at least two of the entries p_{10}, p_{20}, p_{30} equal to zero. An inspection of the print-outs for the 20 and 21 shifters shows that this does not happen (Figure 2), and so SH(3) = 21.

A question was raised by some BB-n enthusiasts as to whether a maximum scorer in the BB-n game (a valid entry in the BB-n classification with a score of $\sum(n)$) must always have an unbroken string of ones in its output tape when it stops; the conjecture being that it must. An inspection of the print-outs for the five 6-scorers shows that this need not be the case (Figure 1), and this question is therefore also settled.

CHAPTER VI

CONCLUSION

Should one attempt to apply the method described above to the problem BB-1963, for example, then difficulties of prohibitive character are bound to arise. In the first place, the number of cases becomes astronomical, and the storage and execution for the computer programs involved will defeat any efforts to use existing computers. Even if we assume that somehow we managed to squeeze through the computer the portion of our approach involving partial recurrence patterns, the number of "holdouts" may be expected to be enormous. Over and beyond such "physical" difficulties, there is the basic fact of the non-computability of $\sum(n)$, which implies that no single finite computer program exists that will furnish the value of $\sum(n)$ for every n. Accordingly, there seems to be at present no justification for the assumption that $\sum(n)$ is effectively calculable for each individual n. Evidently, these comments suggests a number of questions relating to the BB-n problem which seem to be beyond the reach of presently known methods. Thus it appears that clarification of the idea of a "given" non-negative integer may be a fruitful and certainly difficult enterprise.

TABLE OF CONTENTS

BIBLIOGRAPHY

1. Kleene, S. C., Introduction to Metamathematics, D. Van Nostrand Co., Princeton, N. J., 1952.

2. Rado, T., On Non-Computable Functions, The Bell System Technical Journal, Vol. XLI, No. 3, May, 1962.

3. Rado, T., On Non-Computable Functions, The Bell Telephone System Monograph 4199, 1962.

AUTOBIOGRAPHY

I, Shen Lin, the eldest son of Chio-Shih Lin and Shui-Hsian Wang, was born in Amoy, China on February 4, 1931. I attended elementary school in China before our family emigrated to the Philippines prior to the onset of the Second World War. Thus I eventually finished my high school education in the Philippines and was graduated from the University of the Philippines in 1951 with the degree of Bachelor of Science in Mathematics (summa cum laude).

I came to the United States in 1952 to pursue my graduate work at The Ohio State University and obtained my Master of Arts in Mathematics a year later. From 1952 to 1955, I was a Graduate Assistant, and from 1956 to 1959, an Instructor in the Department of Mathematics, The Ohio State University. In 1959, I accepted a position as Assistant Professor of Mathematics at Ohio University in Athens, Ohio, and remained in that capacity until 1962. At present I hold a Research Associateship at the Numerical Computation Laboratory and serve as a part time Lecturer at the Department of Mathematics, The Ohio State University. In October, 1963, I shall join the Technical Staff of the Bell Telephone Laboratories at Murray Hill, New Jersey.

In 1956, I married Mona L. Lo in Columbus, Ohio, and we have three sons; John, six; David, four and a half; and Robert, two and a half. I became a naturalized United States citizen in 1961 and plan to continue my carser in this country.

LIST OF ILLUSTRATIONS

CHAPTER I

INTRODUCTION

The studies presented in this research originated with the following observations relating to general recursive functions (see Kleene [1] for terminology). Let GRF be the class of general recursive functions. By a basic theorem in Mathematical Logic, every general recursive function can be computed by a Turing machine. Hence, if we denote by TC the class of functions computable by Turing machine, then

(1) $GRF \subset TC$.

Next, it is easy to see that any given Turing machine can be simulated on any general-purpose computer. Hence, if we denote by CC the class of "computer-computable" functions, then

(2) $TC \subset CC$.

Actually, the classes GRF, TC, CC are identical; but for our present purposes the partial results (1), (2) are sufficient. As regards (2), let us note that a Turing machine operates on a potentially both-ways infinite tape, while an actual computer has only limited "fast storage". However, by going on tape if necessary, an actual computer has also potentially infinite sotrage, and this fact enables us to state and prove (2).

In view of (1) and (2), every general recursive function is programmable for any given general-purpose computer; accordingly, problems in Mathematical Logic relating to computability, decidability and solvability are accesible to study by means of computers. Thus it appears that such problems can be formulated

and studied in an entirely concrete manner; namely, as problems relating to computability by actual computers.

However, certain basic issues emerge if the line of study suggested by the preceding comments is pursued. Let us first note that by function we mean in the present context any function f of any finite number of variables such that the function and the variables assume only values that are non-negative integers. Consider now a constant function of a single variable x; let us denote such a function by $f_q(x)$, where q is the constant value of the function. By definition, $f_q(x)$ belongs (as a so-called initial function) to the class GRF of general recursive functions, and hence it should be "computer-computable". Now, evidently, to program the computation of $f_q(x)$, the constant value q must be "given" in a very concrete sense, since it is an input item for the program. On the other hand, the majority of mathematical logicians seem to feel that it is sufficient to know that q is uniquely determined by some set of conditions. To make this point clear, let us consider an extreme illustration, proposed by some mathematical logicians. Let q be the truth-value of the Riemann hypothesis on the roots of the zeta-function (that is, q = 1 if the Riemann hypothesis is true, and q = 0 if the Riemann hypothesis is false). Then q is uniquely determined; and the constant function $f_q(x)$ is also uniquely determined; and it is a member of the class GRF (since it is a constant function). Hence, there exists a Turing machine M that computes the value of $f_q(x)$ for every x; in particular, it computes $q = f_q(0)$, and thus it reveals to us whether the Riemann hypothesis is true or false.

Mathematicians (as distinguished from mathematical logicians) will probably view this method to settle the Riemann hypothesis with mixed feelings. It is also clear that the truth-value q of the Riemann hypothesis is not "given" (at present) in a manner

adequate for use as an input item in a computer program. Thus there arises the issue of assigning to the term "given non-negative integer" a precise meaning that coincides with "usability as an input item in a computer program". In the course of a correspondence on this type of issues, Kleene stated, essentially, that as far as he knows such a precise concept is not available at present. The purpose of this research is to contribute to the clarification of the issue so raised by a detailed study of certain very concrete problems relating to Turing machines. These problems arose in connection with the Busy Beaver logical game (see Rado [2]) which we shall briefly describe in the next chapter. This quite primitive logical game led to problems which defied the efforts of a number of experts who became interested in it. This group of people is referred to informally as the International Busy Beaver Club; it includes expert mathematicians, logicians, and skilled programmers. Published results are contained in references [2] and [3] . In this present work, we shall discuss and solve the BB-3 problem (the Busy Beaver logical game in the 3-card classification). We hope that this apparently very special issue will help to elucidate the difficulties involved in assigning a precise meaning to the intuitive concept of a "given" non-negative integer.

CHAPTER II

TERMINOLOGY

We assume as background familiarity with the discussion of Turing machines in Kleene [1] . We shall operate with binary Turing machines with the alphabet 0, 1. In the way of illustration, consider the following Turing machine.

CARD 1			
0	1	1	2
1	1	1	3

CARD 2			
0	1	0	1
1	1	1	2

CARD 3			
0	1	0	2
1	1	1	0

Actually, a Turing machine is not a machine, but rather a program (set of instructions) spelled out in a fixed format, as illustrated above. The instructions are specified on a finite number of "cards"; thus the above figure shows a 3-card Turing machine. The term "card" seems to be preferable to the term "state" or "internal configuration", since the idea of a Turing machine is not dependent upon physical computers. Let us also note that for reasons of convenience we deviate from Kleene by not permitting a "center shift". On each card, the left-most column contains the alphabet 0, 1. The next column to the right contains the "overprint by" instruction. The next column to the right contains the shift instruction, where 0 is the code for left shift, 1 is the code for right shift. The right-most column shows the "call" instruction; it shows the index of the card to which control is transferred.

In the "call" positions, we may have any one of the card