AN ABSTRACT OF THE THESIS OF

Allen Harold Brady (Name) for the Ph. D. (Degree) in Mathematics (Major)

Date thesis is presented December 10, 1964

Title SOLUTIONS OF RESTRICTED CASES OF THE HALTING PROBLEM APPLIED TO THE DETERMINATION OF PARTICULAR VALUES OF A NON-COMPUTABLE FUNCTION

Abstract approved Redacted for privacy
(Major professor)

Some simple theorems and procedures are derived for use in establishing that certain Turing machines will never halt because of conditions arising in their history of operation or because of the peculiar construction of the machines. The special case of the halting problem of two-symbol, k-state Turing machines with blank input tapes arises in the determination of particular solutions to a recursively unsolvable logical game, the so-called Busy Beaver game. Associated with this game are two well-defined integer functions of k, the "Busy Beaver number" BB(k), and the "shift number", SN(k). Both functions are known to be non-computable. To determine particular values of these functions, a heuristic program was written for a digital computer to "solve" the blank input tape halting problem and determine the values of these functions for k = 2, 3, and 4. In the program an efficient method for the generation of all non-redundant two-symbol, k-state machines is coupled with a heuristic method

for solving the halting problem. It was determined that $SN(2) = 6$ and that $BB(2) = 4$, $BB(3) = 6$, and $SN(3) = 21$. The latter three values corroborate known results, and it was further determined that the same values hold when the center move is allowed in the machines. It has been shown that $SN(4) \geq 84$ and that $BB(4) \geq 11$ (or ≥ 12 using a different stopping convention), and the halting problem of two-symbol, four-state Turing machines with blank input tapes has been reduced to deciding the individual cases of a set of machines containing more than 10,817 but fewer than 18,061 distinct members. The mechanization of certain procedures in addition to those used is suggested to reduce the number of cases which remain for $k = 4$. It is also suggested that suitable heuristics are needed for the application of mathematical induction to this problem, and that the problem should be studied for the values $k = 5$ and $k = 6$.

SOLUTIONS OF RESTRICTED CASES OF THE HALTING PROBLEM APPLIED TO THE DETERMINATION OF PARTICULAR VALUES OF A NON-COMPUTABLE FUNCTION

by

ALLEN HAROLD BRADY

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

DOCTOR OF PHILOSOPHY

June 1965

APPROVED:

Redacted for privacy

Professor of Mathematics

In Charge of Major

Redacted for privacy

Chairman of Department of Mathematics

Redacted for privacy

Dean of Graduate School

Date thesis is presented December 10, 1964

Typed by Carol Baker

ACKNOWLEDGEMENT

The author wishes to thank Professor H. E. Goheen for suggesting this problem and for his most helpful advice and ecouragement. The author also extends thanks to Professor W. R. Stahl for his interest and helpful discussion; to Dr. L. B. Lusted, Chairman, Department of Biomathematics and Automatic Data Processing of the Oregon Regional Primate Center at Beaverton, for making available the computer facilities; and to Mr. Robert W. Coffin, Chief Programmer at the Primate Center, for his invaluable assistance and advice in the use of the SDS 920 computer. This research was sponsored by the Graduate Research Council of Oregon State University, and support was also received from the National Institute of Health under Grant GM 11178.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT.)

LIST OF FIGURES

SOLUTIONS OF RESTRICTED CASES OF THE HALTING PROBLEM APPLIED TO THE DETERMINATION OF PARTICULAR VALUES OF A NON-COMPUTABLE FUNCTION

I. INTRODUCTION

In 1936 A. M. Turing [20] devised an idealized scheme for computation whereby a person (or machine) can carry out the rules of any algorithm that is expressed as a finite set of atomic acts which refer to the marking upon or shifting along a tape of potentially infinite length and to the choosing of a subsequent atomic act. He was able to show that his simple scheme with a strong intuitive basis was logically equivalent to other attempts to define precisely the concept of an algorithm such as the λ-definability of Church [21]. Any theoretical mechanization of a "computer" in this sense of Turing is classically referred to as a Turing machine.

A particular contribution of Turing's development of the notion of computability was his demonstration of the existence of a universal machine which can carry out the computation of an arbitrary Turing machine whose description and input tape comprise the input tape of the universal machine. The influence of this discovery upon the development of the electronic computers which were to follow in a decade cannot be overlooked [24, p. 37]. We shall first review Turing's concept and discuss some of the notation used.

Turing considered an "infinite" tape (or strip of paper)

measured off in squares which could be blank or could contain some symbol from a finite alphabet. A machine or person is able to scan a square and determine which symbol is marked upon it or that it is blank. This is done while the machine is in a particular configuration or the person in a particular state of mind such that the finite sequence of acts to follow are determined uniquely according to this state and the particular marking of the square. The sequence of actions to which the machine is directed is made up of one or more of the following acts:

(1) printing a new symbol on the square;

(2) erasing the square;

(3) moving to the left one square;

(4) moving to the right one square;

(5) staying in place (doing nothing).

Each sequence is then terminated with the act of changing to a specified new machine configuration (state). The act of erasing can be eliminated by considering it to be equivalent to the printing of a blank (Post [4]), and "staying in place" or "doing nothing" can be thought of as the act of printing the same symbol that was scanned. This would reduce the possible acts to three plus a change of state.

Turing used two conventions to restrict the formation of the aforementioned sequences. The first convention [20, p. 239] is

basic to the development given by Kleene [8] while the second convention [20, p. 251] is basic to that used by Post[15] and Davis[5]. A formal proof of the logical equivalence of the two conventions in the context of the notion of a computational scheme is given by Anderson [1]. Where Turing considered a tape which was infinite in one direction only, Kleene[8] and Davis[5] follow Post[15] in using a tape that is infinite (or infinitely extensible) in two directions, but this presents no logical difficulty. In this thesis we shall follow the development by Kleene[8] in restricting all sequences to a triple:

(1) print a symbol(to include printing either a blank or the same symbol);

(2) move left, stay in place (center move) or move right;

(3) change to a specific state (to include remaining in the same state).

The moves will be designated by "L", "P", or "R" respectively.[1] A formal definition of a Turing machine will be given in Chapter III following essentially that definition given by Anderson[1].

To carry out a computation or the steps in an algorithm, the corresponding Turing machine is started in a particular state on an "input" tape which is all blank except for at most some finite number of squares. The computation is finished when the machine stops, i.e.,

[1] Kleene uses "C" for "center" in place of the "P" we use here. Turing used "P" to stand for "print."

when it reaches an explicit command to halt (e. g. the triple q P s with the machine in state q scanning symbol s) or a state-symbol configuration for which no commands are defined.

Associated with the collection of all Turing machines are some undecidable problems among which is the well-known halting problem: there does not exist an algorithm (or equivalently a Turing machine) which can act upon an arbitrary Turing machine and its input tape to determine whether or not the Turing machine will eventually halt its operation on the tape. This is true in particular for the universal Turing machine.[1]

A particular "well-defined" integer function has been given by Rado [16] which is defined in terms of the class of two-symbol (binary) Turing machines. This function of the positive integer k is defined as the maximum number of (non-blank) marks which can appear on the output tape of an arbitrary two-symbol, k-state machine which eventually stops after being started on an all blank tape. The determination of the value of this function depends upon the solution of the halting problem for blank tapes which is unsolvable (see Chapter III). It is not clear what is to prevent us from determining in general the values of this and other associated functions. The value of this function for k = 2 is known[16], but after considerable

[1] A simple proof of this fact developed in terms of the self-applicability of a Turing machine is given by Trakhtenbrot[22].

effort had been expended utilizing digital computers, the value for $k = 3$ was not known at the time of publication of Rado's original paper [16] on the subject.[1] Neither were any values known for $k > 3$. Accordingly, research was undertaken to devise a means to effectively compute the value of this function for the particular argument $k = 3$ and perhaps shed some light upon the difficulties involved in "calculating" the value of the function for even greater values of k.

---

[1] Prior to the completion of the research reported in this thesis it was learned that the problem for $k = 3$ had been solved [11] by Dr. Shen Lin, a former student of Rado's[17]. The solution of the problem to be presented here was determined independently of the efforts of Dr. Lin and gratifyingly agreed completely with his particular result. I am indebted to Dr. Lin for his later communication of a copy of his paper pertaining to this result which is to be published in collaboration with Prof. Tibor Rado.

II. TURING MACHINES AND SOME SIMPLE HALTING THEOREMS

2.1 Basic Definitions

Given two sets A and B, a function, written $f: A \rightarrow B$, is a subset of $\{A \times B\}$ such that if $(a, b) \in f$ and $(a, b') \in f$, then $b = b'$. f will be called finite if it contains a finite number of elements.

Let Q be an enumerable set, M be the set $\{-1, 0, +1\}$, and S be any pre-defined set. The elements of Q will be called states, the elements of M moves, and the elements of S symbols. A Turing machine is a function $Z: \{Q \times S\} \rightarrow \{Q \times M \times S\}$, finite in the sense described above.[1] The finite set A of elements $s \in S$ determined by the existence of a quintuple element $z = (q_1, s', q_2, m, s)$ or an element $z = (q_1, s, q_2, m, s')$ in Z is called the alphabet of Z. The finite set of elements $q \in Q$ similarly determined will be referred to as the states of Z.

A tape t is a function mapping the integers I (negative and non-negative) into the alphabet A. An element of A which is the image of an infinite set of integers will be called a "blank". Only tapes which define one blank element shall be admissible.

A triple (z, t, i) where z is a quintuple in some Turing

---

[1] The definitions of Turing machine, tape, etc. given here follow essentially that of Anderson [7] but we do not include the tape with the Turing machine in the definition.

machine Z, t is an admissible tape, and $i \varepsilon I$, is called a complete configuration [5]. Let T be the collection of admissible tapes. We define a function on the set of complete configurations (denoted by the product set $\{ Z \times T \times I \}$) into itself which we shall call the Turing operating function:

$$O(q_1, s_j, q_k, m, s, t_1, n) = (q_k, t_2(n+m), q', m', s', t_2, n+m)$$

where q', m', and s' are determined uniquely by the Z correspondence $(q_k, t_2(n+m)) \rightarrow (q', m', s')$ and t_2 is determined by

$$t_2(n) = s; \quad t_2(r) = t_1(r), \quad r \neq n.$$

If $Z(q_k, t_2(n+m))$ is not defined then O is not defined.

We can now prove the following

LEMMA 2.1. If $O(z, t, n) = (z, t, n)$, then $z = (q, s, q, 0, s)$ for some state q of Z and some $s \varepsilon A$.

PROOF. Suppose $O(z, t, n) = (z, t, n)$. Then taking $z = (q_i, s_j, q_k, m, s_\ell)$ we have from the definition of O:

$$n = n + m \Longrightarrow m = 0.$$

Thence

$$s_j = t(n+0) = t(n) = s_\ell,$$

and

$$q_i = q_k .$$

Therefore, $z = (q_i, s_j, q_i, 0, s_j)$.

Using the preceding lemma we prove the following

THEOREM 2. 2. $O^n(z, t, i) = (z, t, i)$ for all $n \geq 1$ if and only if $z = (q, s, q, 0, s)$ for some q of Z and $s \in A$.

PROOF. Proof that the condition is necessary follows from the lemma above. Suppose $z = (z, s, z, 0, s)$ and let $(z', t', i') = O(z, t, i)$. Then

$$O(q, s, q, 0, s, t, i) = (q, t'(i+0), q', m', s', t', i+0),$$

but $t'(i+0) = t'(i) = s$, and therefore, $z' = (z, s, q', m', s') = z$ by the definition of Z. Thus the condition is sufficient for $n = 1$ and by induction holds for all $n \geq 1$.

A quintuple of the form $z_s = (q, s, q, 0, s)$ is called a <u>stop</u>, and a complete configuration of the form (z_s, t, i) where $t(i) = s$ is called a <u>stop configuration</u>.

The sequence of complete configurations defined by successive applications of the Turing operating function O to an initial complete configuration (z_0, t_0, i_0) will be called the <u>history of operation</u> of the Turing machine Z upon the tape t_0. When speaking of complete configurations we will admit a configuration (z, t, i) as <u>proper</u> if $t(i) = s$, where $z = (q, s, q', m, s')$. It is obvious that only proper

complete configurations will appear in the history of operation if the initial configuration is proper. In the halting theorems to follow later we will be interested in establishing conditions of "not-stopping" or equivalently that there does not exist an $n \geq 0$ such that $O^{n+1}(z, t, i) = O^{n}(z, t, i)$. If for some configuration C, O(C) is not defined, then the history of operation terminates; if a stop configuration occurs, then the history of operation will also be considered to terminate. Thus a <u>defined</u> stop will be considered equivalent to a terminal situation.

It will be convenient in some situations to restrict our attention to only certain portions of a tape. The restricted range will be written in brackets following the symbol for the tape function. Thus

$$t[\alpha, \beta] \equiv t \bigcap \{ \{i \mid \alpha \leq i \leq \beta\} \times S\} ;$$

$$t[-\infty, \alpha] \equiv t \bigcap \{ \{i \mid i \leq \alpha\} \times S\} ;$$

$$t[\alpha, \infty] \equiv t \bigcap \{ \{i \mid i \geq \alpha\} \times S\} .$$

Composite tapes will be defined by the union of tapes with restricted ranges:

$$t_i[-\infty, \alpha]\, t_j[\alpha, \beta]\, t_k[\beta, \infty] \equiv t_i[-\infty, \alpha] \bigcup t_j[\alpha, \beta] \bigcup t_k[\beta, \infty]$$

The next theorem follows from the definition of O.

THEOREM 2. 3 If $\alpha < i < \beta$ and $O(z, t, i) = (z', t', i')$, then

$$O(z, t_m[-\infty, \alpha]\, t[\alpha, \beta]\, t_n[\beta, \infty], i) = (z', t_m[-\infty, \alpha]\, t'[\alpha, \beta]\, t_n[\beta, \infty], i')$$

for any tapes t_m and t_n.

A linear shift of the characters on the input tape t_0 is of no consequence if we correspondingly shift the starting location i_0. The next theorem states that the computations are "relatively" equal if the initial tapes are likewise.

THEOREM 2. 4. If $t_0[\alpha, \beta](j) = t_0'[\alpha+k, \beta+k](j+k)$ for some fixed k, and if $\alpha \leq i_\ell \leq \beta$, $\ell = 0, 1, 2, \ldots, n$, then $t_n[\alpha, \beta](j) = t_n'[\alpha+k, \beta+k](j+k)$ where t_n and t_n' are the tapes in the configurations determined by $O^n(z, t_0, i_0)$ and $O^n(z, t_0', i_0+k)$ respectively.

PROOF. The proof depends upon the preceding theorem and the definition of O.

2. 2 Halting

In this section we shall examine some theorems which are of use in recognizing when an operating Turing machine is in a "cycle" or "loop" and will not stop. It should be pointed out that in his original paper Turing [20] was concerned with computable "infinite" sequences which can be constructed digit by digit by a machine which

will keep moving down the tape without ever coming to a "stop". Machines which behave in this fashion were termed "circle-free" and machines which print only a finite number of non-blank symbols or else halt in the sense defined in §2.1 were termed "circular." He showed that there does not exist a (Turing) machine which can distinguish between the two types. This is equivalent in our case to saying that there does not exist an algorithm (Turing machine) to determine whether or not an arbitrary Turing machine will ever stop once it has been started on an arbitrary tape. It is pointed out in Chapter III that "an arbitrary tape" may be replaced by "a blank tape" and the preceding statement still holds.

In this section we shall be primarily interested in theorems which can be mechanically implemented. While in some pure sense it may seem no more difficult to "remember" all the terms in a sequence of tape configurations (especially storing them in a Turing machine with its "infinite" memory) than to keep track of the sequences of locations and of state-symbol coordinate pairs, this will not be the case on an actual digital computer. The non-mathematical factor of realizable time must also be considered.

The simplest type of non-stopping degeneracy is the "fixed loop." We have the first

THEOREM 2.5. If there exist n and m with $n > m$ such that

$O^n(C_0) = O^m(C_0)$ where C_0 is the initial configuration, then the sequence of operations will not terminate.

PROOF. $O^{n+k(n-m)}(C_0) = O^m(C_0)$ for $k = 1, 2, \ldots$.

A not very practical test for the preceding condition would be the exhaustion test. If the locations of operation of a Turing machine on a particular tape appear to be bounded, then we need only to allow the machine to run through a number of operations which exceeds the number of possible configurations realizable within this bounded region of the tape. Suppose the alphabet of our Turing machine consists of n symbols and the machine has r states. Let its operation be restricted to a segment of the tape consisting of ℓ squares. Then there are n symbols for each of ℓ squares, ℓ positions to be scanned, and r states for each position for a maximum total of $r\ell n^{\ell}$ configurations. For a three-state, two-symbol machine confined to a four-square segment of tape this exhaustion test would require 193 moves while the cycle might repeat itself in, say, six moves. We desire a more "practical" procedure.

Before we proceed we should note that the complete configurations in a history of operation are not necessary to reconstruct the steps in a computation. We have the

THEOREM 2.6. Let $\{c_1, c_2, \ldots, c_n\}$ be the sequence of coordinate

pairs in the history of $O^n(z_0, t_0, i_0) = (z_n, t_n, i_n)$ and let it serve likewise in the history of $O^n(z_0', t_0', i_0') = (z_n', t_n', i_n')$. If $(z_n', t_n', i_n') = (z_n, t_n, i_n)$ then $(z_0', t_0', i_0') = (z_0, t_0, i_0)$.

PROOF. For $n = 1$ this follows from the definition of O. For $n > 1$ the proof is by induction.

We see then that the entire history of operation can be reconstructed given the final complete configuration and the sequence of coordinate pairs. But without actually "reconstructing" we can draw some conclusions from a limited amount of information:

THEOREM 2.7. Let $\{c_0, c_1, \ldots, c_n\}$ be identical with $\{c_n, c_{n+1}, \ldots, c_{2n}\}$ and let $i_n = i_{2n}$ in the location history of $O^{2n}(z_0, t_0, i_0)$. Then $O^n(z_0, t_0, i_0) = (z_0, t_0, i_0)$ and we have a loop.

The proof of this theorem depends upon the

LEMMA 2.8. Let $\{c_0, c_1, \ldots, c_n\} = \{c_0', c_1', \ldots, c_n'\}$ in the histories of $O^n(C)$ and $O^n(C')$ respectively. Let $\alpha = \min\{i_j \mid k = 1, 2, \ldots, n\}$ and $\beta = \max\{i_j\}$ in the location history of $O^n(C)$; α', β' the corresponding extremes in the history of $O^n(C')$. If $C = (z_0, t_0, i_0)$ and $C' = (z_0', t_0', i_0')$, then

$$\alpha + i_0' = \alpha' + i_0, \quad \beta + i_0' = \beta' + i_0, \text{ and } t_0(\alpha + k) = t_0'(\alpha' + k),$$

for all k such that $|k| \leq \beta - \alpha$.

PROOF. The proof is by induction from the definition of O.

PROOF of Theorem 2.7. Let α and β be the minimum and maximum location extremes in the history of $O^n(C_0)$, where $C_0 = (z_0, t_0, i_0)$, and let α' and β' be the corresponding extremes in the history of $O^n(O^n(C_0))$. Since the coordinate pair histories satisfy the conditions of the preceding lemma, and since the final locations are equal, $i_n = i_{2n}$, we have

$$\alpha = \alpha' \qquad \text{and} \qquad \beta = \beta',$$

and since the tape region outside $[\alpha, \beta]$ is unchanged,

$$t_0 = t_n. \qquad \text{QED}$$

By definition an input tape (or a tape at some point in the operation history) must be all blank except at a finite number of squares. The exhaustion test tells us that the region of tape being scanned must eventually grow if a machine not in a fixed loop does not stop. The simplest (in an intuitive sense) form of this type of degeneracy is described by the term traveling loop and is defined by the behavior described in the

THEOREM 2.9. Let $O^n(z_0, t_0, i_0) = (z_n, t_n, i_n)$, $k = i_n - i_0 > 0$, and let i_{min} equal the minimum location in the sequence $\{i_0, i_1, \ldots, i_n\}$. If $t_0(i) = t_n(i+k)$ for all $i \geq i_{min}$, then the operation sequence will

not terminate.

PROOF. The proof is by induction using theorem 2.4.

We may refer to the behavior described in this theorem specifically as a "right-traveling" loop and note that a corresponding theorem holds for a "left-traveling" loop. The determination that $t_0(i) = t_n(i+k)$ for all $i \geq i_{min}$ does not present a problem if we know the maximum location, i_{max}, in the history of $O^n(z_0, t_0, i_0)$, for $t_n(i)$ will be blank for all $i \geq i_{max}$ in, for instance, the case of an initially all blank tape. At at least one point in the operation sequence the maximum excursion will obviously have to change, and the traveling loop conditions will repeat themselves at another change in the location maximum occurring n operations later. In a "practical" Turing machine (by "practical" is meant a deliberately contrived Turing program for some purpose of calculation) one may by convention establish end-blocking symbols equivalent to blanks except that they are never "over-printed" unless they are "moved" up or down the tape to expand the "useable" region. If a move of such an end-blocking symbol indicates an expansion, then it makes detection of changes in the maximum or minimum excursion simpler.

Theorem 2.9 requires knowledge of tape configurations at separate points in the history of operation as did Theorem 2.5 for

detecting a fixed loop. We saw that this was not necessary in the case of a fixed loop (Theorem 2. 7), and the next theorem shows that it is not necessary in the case of a traveling loop.

THEOREM 2.10. Let $t_0(i)$ be blank for $i > i_n$. If $\{c_0, c_1, \ldots, c_n\} = \{c_n, c_{n+1}, \ldots, c_{2n}\}$ and $i_n = \max\{i_0, i_1, \ldots, i_n\}$ and $i_{2n} = \max\{i_n, i_{n+1}, \ldots, i_{2n}\}$ in history of $O^{2n}(z_0, t_0, i_0)$, then we have a right-traveling loop if $i_{2n} > i_n$.

PROOF. Define $k = i_{2n} - i_n$ and then by lemma 2. 8 and the fact that $t_n(i)$ and $t_{2n}(i+k)$ are blank for all $i > i_n$ we see that the conditions for theorem 2. 9 are satisfied.

Another kind of degenerate (non-stopping) behavior is observed in three- and four-state, two-symbol Turing machines which have started on blank tapes and involves a "growing string" of characters in which a particular symbol configuration is repeated along the string as in the case of the "traveling-loop" behavior. In the "traveling-loop" the pattern segments are "added on" so-to-speak, while in this new situation the Turing machine sweeps back and forth from one end of the string to the other, causing the string to grow systematically at one or both ends. In some cases (in the three- and four-state, two-symbol machines) the segment pattern is converted back and forth between two different patterns depending upon the

direction of sweep. After observing many such machines one might be led to the following conjecture in order to avoid the separate inductive proof each case seems to require (cf. Fig. 1 for an example of the sort which inspired it):

Conjecture. Let Z be a Turing machine with the two-symbol alphabet S (blank) and I(mark). Let $t_0(i) = I$ for $0 \leq i \leq k$ and $t_0(i) = S$ otherwise. Let $\{c_0, c_1, \ldots, c_m\}$ be the coordinate pair sequence in the history of $O^m(C_0) = C_m$. Let $C_0 = (z_0, t_0, 0)$ and $C_m = (z_0, t_m, 0)$ where $t_0(i) = t_m(i)$ for $i \neq k+1$ and $t(k+1) = I$. Let $O^n(C_m) = C_n$ where $C_n = (z_0, t_n, 0)$ and $t_n(i) = t_m(i)$ for $i \neq k+2$ and $t_n(k+2) = I$. If the coordinate pair sequence in the history of $O^n(C_m)$ is one of the form

$$c_1, S_1, c_2, S_2, \ldots, c_n, S_n$$

where S_i is an empty sequence, or else a sequence of the form

$$c_{i-r}, \ldots, c_{i-1}, c_i$$

such that each S_j is empty for $i-r \leq j < i$, then the conditions will be repeated in $O^p(z_0, t_n, 0)$ for some $p > n$.

This conjecture is based upon the assumption that only a repetition of the "local" configuration encountered in sweeping across the growing string of marks will satisfy the conditions. The fallacy

	S	I
1	2LI	1RI
2	3RS	2LI
3	===	1RI

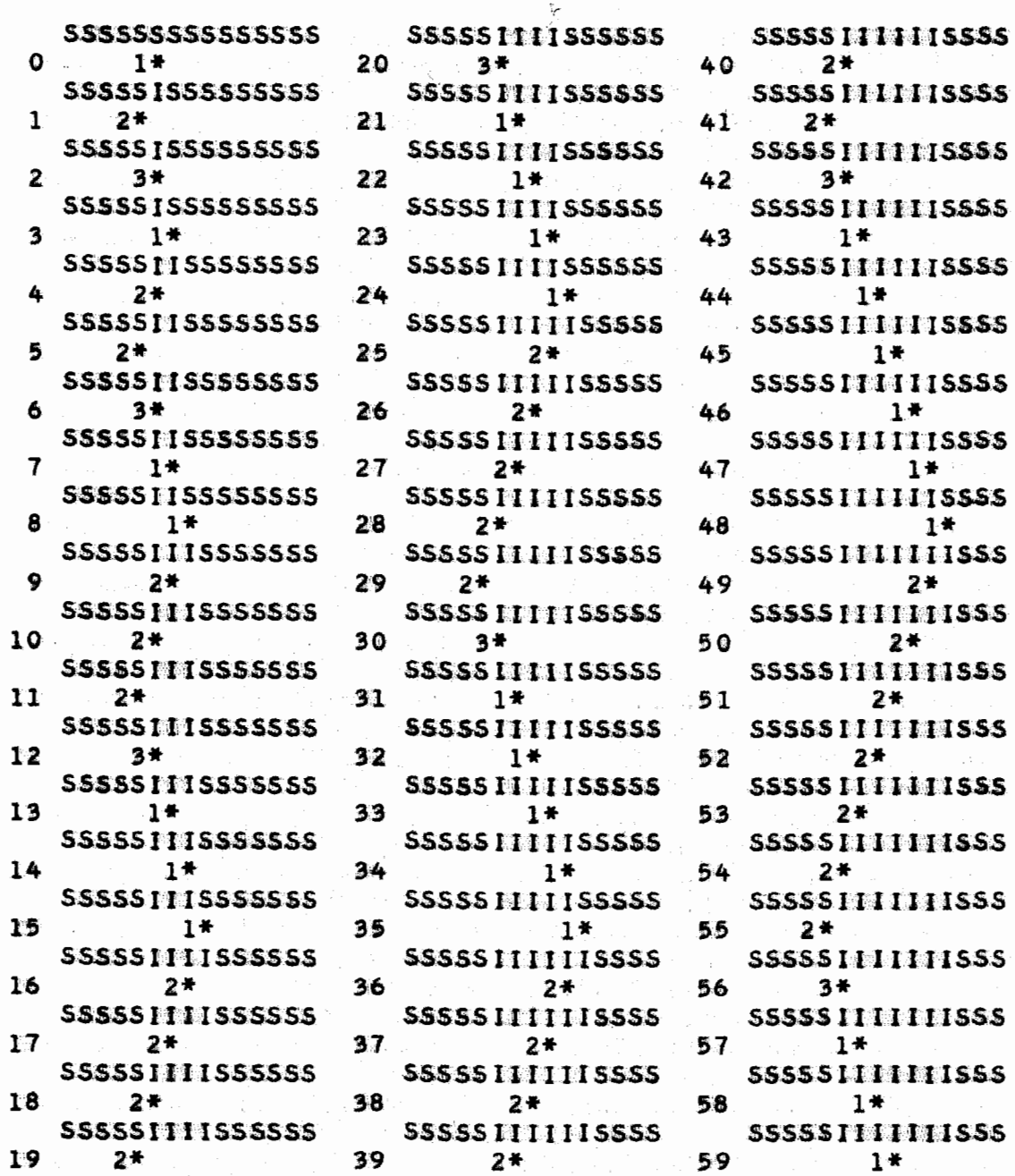

FIGURE 1. MACHINE EXHIBITING A SIMPLE SWEEPING PATTERN.

lies in the fact that other constructions might take place away from the string of marks which are afterwards erased and do not appear on t_0, t_m or t_n. Such a counterexample can be constructed: a machine which copies the string to the left each time and then erases it. When the copy grows into the original the conditions are not repeated. Imposing a maximum and a minimum on the machine locations is still no restriction, for a counterexample can be constructed in which the "digression" takes place within the string:

I I I I I I I I S	Start at left end.
I I I I I I I I I	Move right and add I to string.
I I S S S S S S I	Move left erasing six spaces leaving end mark.
I I S S I I S S I	Copy left portion two spaces to right and
I I S S I I S S I	Return to string.
I I S S S S S S I	Go right erasing copy and continue right to I on end.
I I I I I I I I I	Fill in and return to beginning of string.
	Repeat.

As can be seen, the above procedure will be repeated the second time, and the machine, when coded in a straightforward manner, will satisfy all the conditions in the conjecture, but on the third repetition the copy will merge with the right most mark and the

conditions change.

A generalization of the behavior of the machine in Figure 1 may be made as follows: The machine Z starts in state q_0 on the left end of a repeating string of symbols (not necessarily distinct), the entire tape on either side of the string being blank (b).

$$\ldots bb\underset{q_0}{a_1}a_2\ldots a_k a_1 a_2\ldots a_k\ldots \quad \ldots a_1 a_2\ldots a_k bb\ldots$$

The machine moves generally from the left end to the right end of the string. It may move left but never left into the preceding segment after going from a (former) position of a_k to a position of a_1. After "m" moves we have

$$\ldots bb\beta_1\beta_2\ldots\beta_k\underset{q_0}{a_1}a_2\ldots a_k\ldots \quad \ldots a_1 a_2\ldots a_k bb\ldots$$

It is obvious that eventually we will have

$$\ldots bb\beta_1\beta_2\ldots\beta_k\beta_1\beta_2\ldots\beta_k\ldots \quad \ldots\beta_1\beta_2\ldots\underset{q_0}{\beta_k}bb\ldots$$

Suppose that now the machine converts the right most $\underline{k}$ blanks to a segment $\beta_1\beta_2\ldots\beta_k$ ending up in state q_0' facing the right most β_k:

$$\ldots bb\beta_1\beta_2\ldots\beta_k\beta_1\beta_2\ldots\beta_k\ldots \quad \ldots\beta_1\beta_2\ldots\beta_k\beta_1\beta_2\ldots\underset{q_0'}{\beta_k}bb\ldots$$

The machine now converts the "β-segments" into "α-segments" in a manner analogous to that before, while moving generally right to left

$$\ldots bb\beta_1\beta_2\ldots\beta_k\beta_1\beta_2\ldots\beta_k\ldots \quad \ldots\beta_1\beta_2\ldots\underset{q_0'}{\beta_k}\alpha_1\alpha_2\ldots\alpha_k bb\ldots$$

Eventually the machine will be at the left end facing a blank in state q_0'.

$$\ldots \underset{q_0'}{bb}\alpha_1\alpha_2\ldots\alpha_k\alpha_1\alpha_2\ldots\alpha_k\ldots \quad \ldots\alpha_1\alpha_2\ldots\alpha_k\alpha_1\alpha_2\ldots\alpha_k bb\ldots$$

Continuing with the analogy let the left most k blanks be converted to an α-string ending up in the configuration:

$$\ldots bb\underset{q_0}{\alpha_1}\alpha_2\ldots\alpha_k\alpha_1\alpha_2\ldots\alpha_k\alpha_1\alpha_2\ldots\alpha_k\ldots \quad \ldots\alpha_1\alpha_2\ldots\alpha_k\alpha_1\alpha_2\ldots\alpha_k bb\ldots$$

It is clear that this operation will never stop as long as the conversion of tape segments is independent of the remainder of the tape, for all the necessary information has been given for an induction proof of this fact. Formally it may be stated in a somewhat cumbersome theorem which we shall call

PROCEDURE 2.11. Given a Turing machine Z, let $t_\alpha[i_0, i_0+k-1]$ be defined by

$$t_\alpha(i_0) = \alpha_1,$$

$$t_\alpha(i_0+1) = \alpha_2,$$

$$\cdots$$

$$\cdots$$

$$t_\alpha(i_0+k-1) = \alpha_k,$$

and let it be determined that for some m

$$O^m(q_0, \alpha_1, q_\alpha, \mu_\alpha, \alpha, t_\alpha, i_0) = (q_0, t_\alpha(i_0+k), q'_\alpha, \mu'_\alpha, \alpha', t'_\alpha, i_0+k),$$

where $t'_\alpha(i) = t_\alpha(i)$ for all $i < i_0$ and all $i \geq i_0+k$, and

$$t'_\alpha(i_0) = \beta_1,$$

$$t'_\alpha(i_0+1) = \beta_2,$$

$$\cdots$$

$$\cdots$$

$$t'_\alpha(i_0+k-1) = \beta_k .$$

Let it also be determined that, for some m' and any t_β such that

$t_\beta[i_0, i_0+k-1] = t'_\alpha[i_0, i_0+k-1]$,

$$O^{m'}(q'_0, \beta_k, q_\beta, \mu_\beta, \beta, t_\beta, i_0+k-1) = (q'_0, t_\beta(i_0-1), q'_\beta, \mu'_\beta, \beta', t'_\beta, i_0-1)$$

where

$$t'_\beta[i_0, i_0+k-1] = t[i_0, i_0+k-1].$$

Further, let it be determined that (1) if t is any tape such that $t[a,\infty](i) = b$ (blank) for some value of a, then there is an integer r such that

$$O^r(q_0, t(a), q_\gamma, \mu_\gamma, \alpha_\gamma, t, a) = (q_0', \beta_k, q_\beta, \mu_\beta, \beta, t', a+k-1)$$

where $t'(a+i-1) = \beta_i$, $i = 1, 2, \ldots, k$, $t'(i) = t(i)$ otherwise, and q_0 and q_0' are the states mentioned previously; and (2) if t is any tape such that $t[-\infty, a'](i) = b$ for some integer a', then there is an integer ℓ such that

$$O^\ell(q_0', t(b), q_\delta, \mu_\delta, \beta_\delta, t, a') = (q_0, \alpha_1, q_\alpha, \mu_\alpha, \alpha, t', a'-k+1)$$

where $t'(a'-i+1) = \alpha_{k-i+1}$, $i = 1, 2, \ldots, k$, $t'(i) = t(i)$ otherwise, and q_0 and q_0' are the states mentioned previously. Now, if for some n_0 and some d,

$$t(kj+d) = \alpha_1,$$

$$t(kj+1+d) = \alpha_2,$$

$$\ldots$$

$$\ldots$$

and $$t(kj+k-1+d) = \alpha_k$$

for all j such that $0 \leq j < n_0$; and $t(i)$ is blank otherwise, then $O^n(z_0, t, d)$ will not terminate, where z_0 is defined by $c_0 = (q_0, \alpha_1)$.

Twenty-five of the 27 examples in Appendix D are more-or-less covered by the above procedure. However, to cover all

those cases it should be generalized further to allow the possibilities:

(1) growth at only one end of the string;

(2) the addition of non-blank end segments not a part of the patterns;

(3) growth after each sweep by more than one repeating segment.

A fourth contingency which might be useful for application to the higher order cases of the Busy Beaver problem of the next chapter is the "traveling, growing string":

(4) addition on one end of i segments and the removal on the other end of j segments, $j < i$.

The machine given in the table below will demonstrate this last behavior when started on all blank (S) tape.

	S	I
1	2 R I	1 L I
2	3 L I	4 R I
3	---	1 L I
4	---	2 R I

The general stopping problem of some seemingly complicated machines can be solved by working the machines "backwards" and making an individual demonstration that no possible terminal configuration can be reached. One might refer to the application of this

process as a heuristic, it actually being the method of reductio ad absurdum. Two particular proofs will be shown, but no generalization of the process will be made, although certain simple situations can arise where it is apparent that the process (of going backwards) can never terminate in an impossible configuration.

THEOREM 2.12. The Turing machine defined by the table

	S	I
1	2 R I	---
2	3 L I	2 R S
3	3 L S	4 L S
4	1 R I	1 R I

where S is the blank symbol can never reach a terminal configuration when started on a blank (S) in state 1.

PROOF. The only possible terminal situation is in state 1 scanning a mark:

$$\underset{1}{\ldots\ \overline{\mathrm{I}}\ \ldots}$$

State 1 can be reached only from the state 4 in two situations:

(a) $\underset{4}{\ldots\ \mathrm{S}\,\overline{\mathrm{I}}\ldots}$

(b) $\underset{4}{\ldots\ \mathrm{I}\,\overline{\mathrm{I}}\ldots}$

Both of these situations are compatible with the situation from which they were derived, where the strike above a symbol corresponds to the location of the assumed terminal situation. The situation (a) cannot be reached from the partial configuration

$$\ldots \underset{3}{S\,\overline{I}} \ldots$$

which is the only possible situation which can precede it. Similarly the situation (b) cannot be derived from the partial configuration

$$\ldots \underset{3}{I\,\overline{I}} \ldots$$

Hence the assumption of stopping leads to contradicting circumstances.

The behavior of the machine of the last theorem can be seen (perhaps not easily) by starting the machine in state 1 on a blank tape. The machine in the next theorem requires several levels of "reverse operation" to demonstrate that it never stops.

THEOREM 2.13. The Turing machine defined by the table

	S	I
1	2 R I	3 L I
2	1 L I	1 R S
3	2 R S	---

can never reach a terminal situation when started on an all blank (S) tape.

PROOF. If the machine is to stop it must be in state 3 scanning the mark I:

$$\ldots \underset{3}{\overline{I}} \ldots$$

To reach the above partial configuration it must have been in state 1 scanning a mark on the square to the immediate right:

$$\ldots \underset{1}{\overline{I}}\, I \ldots$$

This configuration could have been preceded by

$$\text{(a)} \quad \ldots \underset{2}{\overline{I}}\, I\, S \ldots \quad ,$$

but not by the other possible configuration

$$\ldots \underset{2}{\overline{I}}\, I \ldots \quad .$$

Similarly, the configuration (a) could only be preceded by

$$\text{(b)} \quad \ldots \underset{1}{\overline{I}}\, S\, S \ldots$$

and not the partial configuration

$$\ldots \underset{3}{\overline{I}}\, S\, S \ldots$$

The only possible configurations that could precede (b) are

$$\ldots \overline{1}_2 S S \ldots$$

and

$$\ldots \overline{1}_2 S S \ldots$$

which lead to contradictions under the operation of the given Turing machine. Therefore the theorem must hold.

The complete stopping problem for each of the machines in the above theorems has actually been solved since the only possible conditions leading to a stop are enumerated in the proofs. It should be added that proofs of the form given in these theorems certainly lend themselves more readily to mechanization than the Procedure 1.11.

As was stated earlier we know that there exist particular Turing machines for which the halting problem has no solution, namely any universal Turing machine. It has been stated that there does not exist a two-symbol, two-state universal Turing machine [13]. This was shown by reducing the problem to a small number (<100) of cases and investigating each one separately (D. Bobrow and M. Minsky, private communcations, 1964). It is reasonable to assume that the investigators solved the individual halting problems to arrive at their conclusion.

An example of the "solution" of the stopping problem for a particular two-symbol, two-state machine is shown here. We assume that the machine starts in state one on (1) a blank S and (2) a mark I and likewise for state two. Then we work each case forward to the point of a new binary choice which must eventually be the case if a loop does not occur, the latter being a decideable issue as we have shown earlier. For the machine

	S	I
1	2 R I	2 L I
2	1 L I	---

one can easily verify that the following chains from the choice tree yield the only tape constructions on which the given machine will continue operation (the other choices yield a stop):

... S ...	... I ...	... S ...
1	1	2
... I S ...	... S I ...	... I I ...
2	2	1
... S I I ...	... I S I ...	... S I I ...
2	1	2
... I I I I ...	... S I I I ...	... I I I I ...
1	2	1
...	...	...

The chains yield eventually the same construction. Thus we see that the machine will start and continue (moving to the left) only

with

$$\ldots S I \ldots S I S I S S S \quad (1)$$

or

$$\ldots S I \ldots S I S I \quad (1)$$

or

$$\ldots I S \ldots I S I S \quad (2) \quad .$$

Since eventually the tape must be all blanks, the machine must eventually stop on all possible input tapes.

This constructive approach is awkward to say the least, and one can envision its getting rapidly out-of-hand for cases with several states. Of course, if while applying this technique one (unknowingly) encounters a universal Turing machine, then he cannot possibly succeed in solving the halting problem.

III. THE RECURSIVELY UNSOLVABLE BUSY BEAVER PROBLEM OF T. RADO

3.1. The Busy Beaver Problem

Consider the class of all k-state Turing machines with an alphabet consisting of two symbols: a blank (O) and a mark (I). This is a finite class having

$$N_k = (6k)^{2k}$$

members.[1] When a Turing machine of this class is started (conventionally in state one; cf. § 4.2) on an all blank tape it will either eventually stop or not stop. Neither subclass is empty, for consider the k-state machines of the forms given by the partial tables

	O	I
1	1 R O	. . .
2	. . .	. . .
	. . .	

and

	O	I
1	1 P I	1 P I
2	. . .	. . .
		. . .

Of the subclass whose members stop there are machines which will

---

[1] This is allowing the center move and defining a "stop" as a quintuple of the form (q, s, q, P, s) (cf. Chapter II). Rado [16] considers a class with

$$(4(k+1))^{2k}$$

members where a "stop" is defined as a branch to an additional state called the "stop state." He does not allow the center move. This will be discussed in Chapter IV.

leave a number of marks on their tapes. The number of such marks left by a given machine which stops will be called its score, and the least upper bound of this finite set of scores will be called the Busy Beaver number of k (Rado, [16]), and we shall write it BB(k).

As Rado [16] points out this is, by present standards, an "exceptionally well-defined" function. One is tempted to think that the determination of BB(k) for a particular value of k would be a trivial problem for a digital computer given sufficient time to generate the N_k members for a given value of k and to determine the score for each. The deception lies in the fact that the halting problem for blank input tapes is equivalent to the general halting problem: it is undecideable. That is,

THEOREM 3.1. There does not exist an algorithm (Turing machine) which will decide whether or not an arbitrarily given Turing machine will eventually stop after being started on an all blank tape.

PROOF. Given any Turing machine and its input tape, it is clear that there is another Turing machine which will construct, starting on a blank tape, the code for the given Turing machine along with its input tape. This machine will require ℓ states, ℓ being the total number of symbols to be found in the largest contiguous region of the output tape bounded by non-blank symbols. One can also construct a universal Turing machine in, say, m states. The combined

machine of $n = m + \ell$ states which first constructs the code and input of some Turing machine on a blank tape and then presents this as an input to a universal Turing machine is a member of the class of Turing machines with blank input tapes. The stopping problem cannot be solved for all such machines for arbitrary n else the halting problem for the universal Turing machine could be solved.

The statement of the theorem is not restricted to binary (two-symbol) Turing machines. It has been shown by Shannon [18] that every Turing machine with m symbols and n states can be converted (by a Turing machine!) to a binary Turing machine of not more than

$$6mn + n(2^{\ell} - 7)$$

states where ℓ is the smallest integer such that $m \leq 2\ell$. Thus there exists a two-symbol universal Turing machine. For the logical purposes here, then, we may restrict our discussion to binary machines and Theorem 3.1 still holds.

It is not of course completely clear that solution of the halting problem is necessary for the determination of BB(k) for arbitrary k. That it is necessary follows from the proof that BB(k) is not computable, a proof which rests upon the demonstration of the remarkable rate of growth of this function of the integer k.

3. 2 Rado's Proof of the Non-computability of BB(k)

Let f and g be functions in $\{I \times I\}$ where I is the set of positive integers. If there exists some $x_0 \in I$ such that $f(x) > g(x)$ for all $x > x_0$, we shall say that $f(x)$ is eventually greater than $g(x)$ written

$$f(x) >_{ev} g(x)$$

The elementary properties of $>_{ev}$ such as transitivity are clear.

THEOREM 3. 2 (Rado[16]). If f is any computable function (of the integers)

$$BB(n) >_{ev} f(n).$$

PROOF. Given any f define

$$F(x) = \sum_{i=0}^{x} (f(i) + i^2).$$

Then, if $f(i)$ is computable, so is $F(x)$, and F has the properties:

(1) $F(x) \geq f(x)$

(2) $F(x) \geq x^2$

(3) $F(x+1) > F(x)$, i. e. F is strictly monotone.

Since $F(x)$ is computable there exists a (binary) Turing machine Z_F with a fixed number of states C which, when given an input

consisting of x I's, will compute an output of F(x) I's.

Now, for any x we can make the following machine with x states which prints x consecutive marks on a blank tape and stops at the right end of this string:

Z_x:

	O	I
q_1	q_2 R I	---
q_2	q_3 R I	---
...	...	
q_x	q_x P I	q_x P I

We next construct the composite Turing machine Z_c consisting of Z_F operating in turn on the result of Z_F operating on the result of Z_x, so that we calculate F(F(x)) from a blank tape by means of a Turing machine of $2C + x$ states. F(F(x)) is therefore a <u>score</u> for the class which determines $BB(x + 2C)$, i.e.

$$(4)\quad BB(x+2C) \geq F(F(x)) .$$

Now

$$x^2 >_{ev} x + 2C$$

and we have $F(x) \geq x^2$ by (2) so that

$$F(x) >_{ev} x + 2C .$$

But by (3) F(x) is strictly monotone increasing so that

$$F(F(x)) >_{ev} F(x+2C),$$

and substitution of this into (4) yields

$$BB(x+2C) >_{ev} F(x+2C) .$$

By the construction of F

$$F(x+2C) \geq f(x+2C),$$

and since x is arbitrary we take $n = x+2C$ and obtain

$$BB(n) >_{ev} f(n)$$

for any computable f.

Since it cannot be that $f(x) >_{ev} f(x)$ for any function f, it follows from Theorem 3.2 that $BB(n)$ is not computable. Rado [16] points out that there are other related non-computable functions:

(a) the "shift-number", $SN(k)$, i.e. the maximum number of moves which a k-state machine can make after starting on a blank tape and eventually stopping;

(b) $N_e(k)$, the number of k-state machines which stop after being started on a blank tape.

These functions have the interesting properties:

(1) $SN(k) \geq BB(k)$;

$$(2) \quad N_e(k) \leq N_k \ .$$

Since N_k is computable we see that N_e is a non-computable function which does not possess the rate-of-growth properties of SN and BB.

Another function which is also not computable is X(k), the maximum tape excursion of all k-state machines which stop. If we restrict ourselves to machines whose first lateral move is, say, to the right, then we can define RX(k) and LX(k) as the corresponding maximum right excursion and maximum left excursion respectively. These functions are of interest in Chapter IV.

THEOREM 3. 3. At least one[1] of the functions RX(x) and LX(x) is not computable.

PROOF. If it were otherwise the value of BB(k) would be determined employing the exhaustion test (Chapter II).

3. 3 A Discussion of the Growth of the Function BB(k)

The function BB(k) is obviously monotone[2], and considering the possible cases in adding one or two states to a k-state machine

---

[1] Because a one-way infinite tape is sufficient (Turing[20]) we actually can say both are not computable.

[2] Using Rado's convention of allowing a change of symbol in the act of stopping would make BB(k) strictly monotonic increasing.

one can see that

$$BB(k+2) > BB(k)$$

for any k. It is also of interest to note that

$$BB(k) \geq 2k-1 \quad \text{for all } k,$$

and

$$BB(k) \geq 2k \quad \text{for all } k \geq 2 .$$

The last two inequalities can be seen from the trivial fact that $BB(1) = 1$, the fact that $BB(2) = 4$, and the constructions which follow:

2k-1 I's:
(construction for $k \geq 2$)

	O	I
q_1	q_2 R I	q_1 L I
q_2	q_3 R I	q_2 L I
...		
...		
q_{k-1}	q_k R I	q_k L I
q_k	q_1 L I	q_k P I (STOP)

2k I's:
(construction for $k \geq 3$)

	O	I
1	q_2 L I	q_k R I
2	q_1 R I	q_2 L I
3	q_2 R I	q_3 P I (STOP)
...		
...		
q_{k-1}	q_{k-2} R I	q_{k-1} L I
q_k	q_{k-1} R I	q_k L I

While the "scores" for machines with only several states are not spectacular in an astronomical sense, extremely large lower bounds for BB(k) can be demonstrated for values of k = 100. Rado originally contributed

$$BB(100) \geq (((7!)!)!)!$$

from the knowledge that a machine for computing x! can be constructed with no more than 26 states and combining several such machines. He also stated [17] that C. Y. Lee has shown that

$$BB(100) \geq \exp_{10}(\exp_{10}(\exp_{10}(50,000))) .$$

It is an understatement to say that this latter value is too conservative. In fact for k = 100 it could very well get to be a difficult task just to compare variously contributed lower bounds. Perhaps this difficulty may occur well below the value of k = 100. We shall see in the discussion which follows what can be obtained for as few as 14 states and then consider multiple recursion using a "factorial" machine of 24 states.

Consider the following Turing machine where O is the blank space:

	O	I
q_1	q_2 L I	q_1 R I
q_2	q_3 R I	q_2 L I
q_3	...	q_4 R I
q_4	STOP	q_1 R O

O	I
Add I at right	Move to right end
Fill space	Move left to space
(not used)	Skip a mark
Stop if space	Erase mark

It is easily verified that starting on an input tape with an n-mark string in state

q_1 on the string yields 2n+2 marks;

in state q_2 at the space to the left of the string: 2n marks;

and in state q_3 at the left most mark of the string: 2n-1 marks.

By adding only four states one can obtain a machine which computes 2n+2 recursively[1] for a second argument (string of marks) two spaces to the left of the original string:

	O	I
q_4	q_5 L O	q_1 R O
q_5	q_6 L O	q_5 L I
q_6	q_6 L O	q_7 L O
q_7	STOP	q_8 R I
q_8	q_8 R O	q_1 R I

O	I
End of recursion	...
Go to left string	Go left over right string
Go to left string	Delete right most mark
Halt if right string erased	Otherwise continue
Go to right string	Repeat computation

[1] By using (q_4, O, q_5, L, I) instead of (q_4, O, q_5, L, O) one can compute 2n+3 recursively instead of 2n+2.

The following six-state machine (Lee[9]) will mark 31 contiguous I's on a blank (O) tape after 297 moves starting in state one:

	O	I
q_1'	q_2' R I	q_3' R I
q_2'	q_3' R O	q_6' R I
q_3'	q_4' R I	q_2' R I
q_4'	q_5' L O	q_1' R I
q_5'	q_1' L O	q_5' L I
q_6'	STOP	q_4' R I

By replacing $(q_6', O, \text{"STOP"})$ by (q_6', O, q_3, R, O), inserting (q_3, O, q_4, R, I) in the previous machine, and also replacing (q_4, O, q_5, L, I) by (q_4, O, q_5, L, O) in the previous machine [1] one obtains a 14-state machine whose operation on a blank tape gives the inequality

$$BB(14) \geq 5 \cdot 2^{30} = 5,368,709,120 .$$

In Appendix B is a machine using two symbols and 24 states to compute $x!$, $x > 1$. One can cause a k-fold recursion of the computation by the addition of k+1 marks two spaces to the left

[1] See the note at bottom of page 39.

of the input string of x marks. Thus with this machine it becomes an interesting contest for a lower bound of, say, BB(35) to obtain the most in the way of an input string and number of recursions. Also, of course, there is no proof that the factorial machine of 24 states is minimal. With probably no more than 45 states, we could use the BB(14) entry above to construct a recursion argument to the right of which we could construct three marks, and then compute factorial of 3 recursively so that

$$BB(45) \geq 3\underbrace{!!!\ \ldots\ !!!}_{5\cdot 2^{30}-1 \text{ times.}}$$

Compared with Lee's example this may be thought of as going from the sublimely incomprehensible to the ridiculously incomprehensible.

IV. THE SOLUTION OF RADO'S PROBLEM FOR k = 2 AND 3 AND LOWER BOUNDS OF THE SOLUTION FOR k = 4

4.1. Preliminary Remarks on the Problems of Small Order

In a 1963 paper [7] House and Rado wrote:

> Many computer programs were written to [solve the halting problem of three-state, binary Turing machines on a blank tape]; these programs grew larger and larger as more and more [stopping criteria] were covered. These programs were the results of cooperative efforts of experienced mathematicians and skilled programmers, and were run on some of the finest existing computers. Yet this extremely primitive-looking problem is still unsolved, and probably most of the participants in these studies feel that perhaps it will never be solved.[1]

It is not clear just exactly what constitutes the limitation on our ability to "compute" the values of BB(k) and SN(k) for a given value of k. Is it merely the supra-astronomical number of cases to be considered? We have seen in Chapter III that given unlimited time and memory (neither of these being logical restrictions for a Turing machine) there is no method (i. e. algorithm) which will guarantee the solution for arbitrary k. What then is the physical or real limitation as opposed to the logical limitation on our ability

---

[1] See the footnote on page 5 of Chapter I.

to solve this problem for reasonably small values of k? As has been noted earlier the number of k-state binary Turing machines is given by the expression

$$(6k)^{2k}$$

which for several values of k yields:

k = 2:	12^4 =	20, 736;
k = 3:	18^6 =	34, 012, 224;
k = 4:	24^8 =	109, 812, 874, 176;
k = 5:	30^{10} =	594, 090, 000, 000, 000.

To consider (or say just to generate) all the possibilities for k = 2 or k = 3 by manual means would appear to be unfeasible. To do the same for k = 4 or k = 5 using a present day "high speed" digital computer would be even more so. If such a computer required 10 microseconds to execute a single instruction, then just the execution of exactly 24^8 instructions would require more than ten days of continuous operation. If we multiply this by the necessary number of instructions to be executed in the generation program, we will undoubtedly get an impractical computer time requirement. By the same argument consideration of a value of five for k would appear to be hopeless even allowing a thousand fold increase in computer capabilities in the near future.

It happens, however, that manual solution of the problem for

$k = 2$ is entirely feasible, and using the procedure to be presented in this chapter it turns out to be trivial. Needless-to-say, the long evening spent originally in solving the two-state problem by hand led to the developments of this chapter.

4.2 Reduction of Cases: Center Move and Symmetric Redundancies

What is of interest normally in a computation of a Turing machine and particularly in the case of the Busy Beaver Problem is the output or terminal tape. Consider for a moment the "P" or "in place" move. A P-move entry has at most two "dynamic" capabilities: (1) a change of symbol and (2) a change of state. (As was mentioned in Chapter II, a P-move entry which has neither capability is taken to be a stop command.) It is apparent from the definition that a P-entry determines uniquely the subsequent entry in the history of operation of a Turing machine on any input tape. Thus, it is the subsequent entry which is of consequence, and the P-entry can effectively be replaced by it. Formally this can be stated in

THEOREM 4.1. Given a Turing machine Z and an input tape t_0, let $O^n(z_0, t_0, i_0) = (z_n, t_n, i_n)$ where $z_n = (q, s, q_r, O, s_j)$ and is distinct from all z_k, $k < n$. Let (q_r', m, s_j') be the entry determined by the pair (q_r, s_j) and define Z' as the Turing machine

obtained from Z by the replacement of $(q, s, q_r, 0, s)$ by (q, s, q_r', m, s'). Then if t_{n+2} is the tape determined by $O^{n+2}(z_0, t_0, i_0)$, then $t_{n+2} = t'_{n+1}$, where t'_{n+1} is determined by $O^{n+1}(z', t_0, i_0)$, and where $z \in Z$ and $z' \in Z'$ correspond to identical initial states.

The implication of the above theorem should be clear: by iterating the process of replacement of P-entries by their uniquely determined subsequent entries we can, without the addition of any state, eliminate all P-moves in a Turing machine Z except those which comprise a stop or immediately precede a stop. If the state in which we stop is of no consequence (it may be of consequence in certain "practical" situations) then the P-moves which immediately precede a stop can be replaced by "stop" if they do not change the symbol being scanned. If we define a unique "stop" entry (such as no entry) then all P-moves can be replaced by the addition of one more state in which all entries are "stop." [1]

We may apply this then to the Busy Beaver problem for k states: the only effective P-move that can possibly occur in a k-state machine which writes $BB(k)$ marks on a blank tape and then stops will be in an entry which places a mark on a blank space just before branching to a stop. This possibility can easily be

---

[1] Perhaps this could be taken as an argument in favor of Rado's stopping convention [16]. Cf. § 4. 3.

considered separately for every k-state machine which we generate without P-moves. This yields a considerable immediate reduction in the number of cases involved for a particular value of k. Taking account of the necessary fact that <u>at least</u> one stop entry must appear in a k-state machine which we would consider for a potential BB(k)-machine, we can say that the number of machines will not exceed the number of such combinations given by

$$\sum_{i=1}^{2k-1} (4k)^{i}(4k+1)^{2k-i} = (4k+1)^{2k} - (4k)^{2k}$$

$$\cong (\sqrt{e} - 1)(4k)^{2k} .$$

We now consider another factor: symmetry. Every machine without P-moves possesses a mirror image obtained by the exchange of the right move "R" and the left move "L" (except of course the trivial case in which all entries are stops). This neatly divides the non-trivial cases by two! We have so far reduced the number of machines for k = 2, 3, 4, and 5 to:

k = 2: 1, 232;

k = 3: 920, 412;

k = 4: 1, 340, 395, 072;

k = 5: 3, 219, 940, 489, 100 .

For k = 3 the number is not unreasonably large but for k = 4 and

$k = 5$ it appears still to be unmanageable.

We have yet to take into account the evident fact that there is nothing of importance in the naming or numbering of the k states. First it is clear that we need restrict ourselves to starting each of the above machines in one state, say state one, for starting in state n is equivalent to starting in state one of the (not necessarily distinct) machine obtained by permuting state one and state n throughout the quintuples comprising the machine. So immediately we can discard all machines of the form

	S	I
1	stop	...
2	...	...

...

...

and furthermore, it is obvious that we can discard any machine Z_k containing a quintuple of the form (1, S, 1, m, a) where $m = \pm 1$ (L or R) and a = S or I. Therefore, the first active quintuple must be of the form (1, S, q, m, a) with $q > 1$. We can also require that $m = +1$ (move right) which takes care of the R-L symmetry already noted. If a = S (space) then the net effect of the first move is to shift right and into a new state on what is still an all blank tape! This contributes nothing to the solution of the halting problem for we eventually must reach an entry which marks I or reach a stop, or else the

machine will never stop. It is clear that this can be determined in k or fewer steps where k is the total number of states. Hence, for our problem we may require that a = I.

If $1 \neq q \neq q' \neq 1$, the class of machines containing the quintuple (1, S, q, R, I) will obviously be in one-to-one correspondence with the class of machines containing the quintuple (1, S, q', R, I) by permuting q and q'. Therefore, if we are going to choose a state q distinct from 1, then we just as well choose q = 2. This reduces our problem to considering only those machines of the form

	S	I
1	2 R I	...
2	...	...

...

with previously mentioned restrictions still in effect. At this point we see that the (2, S) entry must be considered next. It should be apparent by now that we can consider similar arguments to diminish the number of cases even further:

(1) If (2, S) contains a "stop" then we need not consider other machines with identical entries in (1, S) and (2, S).

(2) With either of the entries 2 R S or 2 R I the machine will <u>not</u> stop independent of all other (undetermined) entries.

(3) If we consider an entry which branches to a new state q > 2, then we may take q = 3 by the same argument as before.

The steps above suggest a simple algorithm for successive generation of distinct machines. The only drawback is in the fact that the determination of all entries which lead to situations which will never stop as in (2) above requires another algorithm to make this decision. Such an algorithm does not in general exist according to Theorem 3.1, but the procedure does allow us to make better estimates of the number of cases we must ultimately consider, the approximation being improved by the addition of each new solution of the stopping problem for particular cases.

We can incorporate the reductions thus far mentioned with the suggested method of generation to obtain an upper bound on the number of three-state machines to be considered. It is based on the assumption that all combinations of entries will stop. Consider in order the entries in

(1, S) (1, I) (2, S) (2, I) (3, S) (3, I)

Beneath each of the above pairs we shall consider various possibilities followed by a subtotal of the potential combinations. In (1, S) will always appear 2 R I. Suppose first that we have a stop in (1, I). Then 1 L S or 1 L I in (2, S) lead immediately to the stop, so these we will not consider. Suppose we have 2 L S or 2 L I in (2, S). Then state three must be reached in (2, I), and thence follow 13 possible entries in each of (3, S) and (3, I):

2 R I	STOP	2 L S I	3 L S R I	[13] (stops possible)	[13]	1, 352

If state three is reached from (2, S) then we can consider all 13 possible entries in (2, I):

2 R I	STOP	3 L S R I	[13]	[13]	[13]	8, 788

We continue with a STOP in (2, I) but not in (1, I), noting all potential combinations: first with state three not mentioned in (2, S) and then with only state three mentioned in (2, S):

2 R I	3 L S R I	1 L S 2 I	STOP	[13]	[13]	2, 704
2 R I	[12] (no stop)	3 L S R I	STOP	[13]	[13]	8, 112

Finally we consider stops in (3, S) and (3, I) but not before, taking into account the appropriate mentioning of the various states:

2 R I	3 L S R I (not state 3)	1 L S 2 I	12 (no stop)	STOP	13	2, 496
2 R I	1 L S 2 R I (not state 3)	1 L S 2 I	3 L S R I	STOP	13	1, 664
2 R I	12 (no stop)	3 L S R I	12 (no stop)	STOP	13	7, 424

With the STOP in (3, I), the combinations in (1, S) through (2, I) will repeat those of the previous three situations, considering only in (3, S) and (3, I) the respective possibilities:

Same as preceding three cases	12	STOP	2, 304
	12	STOP	1, 536
	12	STOP	6, 912

This gives a final total of 43, 292 combinations as an upper bound to the number of three-state machines. A similar estimate can be made for $k = 4$ to obtain an upper bound of approximately 15, 000, 000 which is certainly not an unreasonable number. As it has finally turned out, these estimates are too large by factors of about 12 and 30 respectively!

4. 3 A Heuristic Computer Program to Solve the Busy Beaver Problem for $k \leq 4$.

On the assumption that the methods and theorems of Chapter II are of sufficient power to solve the stopping problem on blank tapes for binary Turing machines of small order, a computer program was written for an electronic digital computer based on the generation procedure suggested in the last section.[1] In an intuitive sense perhaps it could be called an algorithm, but as we have seen this is not

[1] While only a general description of the program will be given here, its logical details are given in Appendix C.

entirely accurate. A more fitting term would be a heuristic procedure (cf. Polya [14]) or a heuristic program (cf. Minsky[12]) to describe its incorporation of mathematical "experience" in dealing with similar problems. A degree of "artificial intelligence" could possibly be attributed to the program since it was guided to some extent in its choice of potentially applicable theorems by the gross pattern of the operational behavior of each Turing machine. However, the program did not benefit from its own experience with previously encountered cases, although it became apparent that such ability incorporated into the program might well have improved its efficiency for the number of states $k = 4$. It would now seem reasonable that a degree of such ability might be necessary before there is any hope of solving the problem for $k = 5$.

The main input data for the program consists of the following:

(1) k, the number of states;

(2) a proper k-state Turing machine;

(3) the "order-of-entry" of the input machine;

(4) relevant parameters from the solution of the problem for k-1 states.

A proper input Turing machine, Z_k, is one which might have been generated by the program starting with the machine consisting of the sole quintuplet (1, S, 2, R, I). It depends upon the meaning of

order-of-entry: the sequence of the 2k-1 or fewer coordinate pairs ordered in their appearance in the history of the machine Z_k on a blank tape, starting with (1, S) and such that (q, a) can appear if and only if a pair of the form (q-1, a') has appeared earlier. A proper machine then is one in which the most recent entry is of the form q m a, where m and a are the move and symbol respectively, and q is the state determined by the fact that $q \leq q_{max}+1$, where q_{max} is the largest state appearing in the order-of-entry sequence.

The generation "algorithm" works from a proper Turing machine and uses the information that either a new entry is required or else that the most recent entry yields a machine which will not stop. If a new entry is required the algorithm inserts 1 L S, and otherwise increases the most recent entry to the next triple in the sequence ordered as follows:

$$1\ L\ S < 1\ L\ I < 1\ R\ S < 1\ R\ I < 2\ L\ S < \ldots < k\ R\ I$$

If the branch state in the new entry exceeds $q_{max}+1$, then the most recent entry is cleared (left undefined) and a retrace is made to the previous entry in the order-of-entry, where the procedure is repeated. If the first entry, (1, S, 2, R, I), is reached in the process of retracing, the program stops. If the branch state does not exceed $q_{max}+1$, the generation algorithm goes into a procedure which checks

for the existence of obvious or intrinsic loops, i.e., a situation where the most recent entry is part of a closed system of entries in which there is no stop possible (all entries are defined) and out of which there is no branch possible. If an intrinsic loop is present, the algorithm is repeated starting at the point where the most recent entry is increased. If no such loop is present, then the algorithm relinquishes control to the heuristic section of the program which (hopefully) solves the blank input tape stopping problem for the generated machine.

If the heuristic program fails to solve the stopping problem, the machine is treated as one which does not stop, and the "failure" is retained for later consideration. Note that the generation process does not suffer if it subsequently turns out that a failure actually stops, for the generation can later be started from the erroneous "failure" and then terminated when the generation algorithm retraces back to this starting point. As it has turned out, all of the failures for the three-state problem contained five entries, and thus they could not produce additional machines if they did stop. For the four-state case only 452 failures contained exactly six entries, and there were none containing fewer entries. If any one of the six entry failures turns out eventually to stop, then it can generate only 16 additional machines for consideration.

To solve the stopping problem for k states the heuristic

program requires the values of SN(k-1), RX(k-1) and LX(k-1)[1] which allow a rapid decision on the stopping problem for machines in which the maximum state used is less than k. If a machine turns out to stop (reach an undefined entry) a reckoning procedure notes the values of those parameters pertaining to SN(k), RX(k), LX(k), and BB(k) and prints out any tying or superior values along with the description of the associated machine. It also determines whether or not a P-move entry is possible which changes a symbol prior to a stop, and if so, whether or not a new or tying value for BB(k) is obtained.

All machines in which the k^{th} state is mentioned are run for a fixed number of moves on a Turing machine simulator which saves the sequences of tape locations, maximum and minimum excursions, and entry coordinates (state-symbol pairs). This information is then used to determine whether or not a loop is present by means of a mechanization of Theorems 2.7 and 2.9. If no loop is detected, the given machine is run in reverse in the manner of Theorems 2.12 and 2.13 until a contradiction is obtained or until either the allocated storage or the computer word size is exceeded. Additional moves on the simulator are made if all attempts above fail to solve the problems, and again an attempt is made to detect

---

[1] These are the shift number, maximum right excursion and maximum left excursion respectively. Cf. Chapter III.

looping conditions. As already mentioned, machines for which the stopping problem is not solved are retained for later consideration.

The program was run for the number of states k = 2, 3, and 4, and the results obtained follow.

No. of states k = 2:

No. of machines generated for consideration: 33

No. of above machines which stopped: 15

Function:	BB(k)	SN(k)	LX(k)	RX(k)
Value:	4	6	1	2
No. distinct machines	1	5	(not determined)	

No. of machines on which program failed: none

No. of successful applications of "reverse" procedure: none

Thus as it turns out the problem for k = 2 is trivial. The high scoring machines are given in Figure 2.

No. of states k = 3:

No. of machines generated for consideration: 3, 361

No. of above machines which stopped: 1, 379

Function:	BB(k)	Sn(k)	LX(k)	RX(k)
Value:	6	21	4	3
No. of distinct machines:	5	1	(not determined)	

No. of machines on which program failed: 27

No. of successful applications of "reverse" procedure: 14

SCORE = 4 = BB(2)
SHIFT NO. = 6

	O	I
1	2RI	2LI
2	1LI	2PI

THE MACHINES BELOW HAVE A SHIFT NUMBER = 6
(COUNTING THE ACT OF STOPPING)

	O	I
1	2RO	1PI
2	1LI	2RI

	O	I
1	2RI	2LO
2	1LI	2PI

	O	I
1	2RI	1PI
2	2LO	1LI

	O	I
1	2RI	1PI
2	2LI	1LI

THE HIGH SCORING TWO-STATE MACHINES.

FIGURE 2.

The 27 failures were examined individually, and it was determined that none of these will stop when started on a blank tape. This is discussed in Appendix D. The "high scorers" are given in Figure 3.

No. of states k = 4:

No. of machines generated for consideration: 526, 517

No. of above machines which stopped: 183, 979

Function:	BB(k)	SN(k)	LX(k)	RX(k)
Value:	≥ 11	≥ 84	≥ 10	≥ 10
No. distinct machines:	7	1	(not determined)	

No. of machines on which program failed: 10, 818

No. of successful applications of "reverse" procedure: 10, 306

The high scoring machines are given in Figure 4.

The failures for k = 4 were stored on a magnetic tape, and if all of the 452 failures involving only six entries stop eventually (which does not seem likely), then $16 \times 452 = 7,242$ machines remain to be considered in addition to the 10, 366 failures with seven entries. The four-state problem was thus reduced to fewer than 18, 060 machines.

The experience with the three-state case showed that some loops were not detected on the first run of the simulator when the

LARGEST SHIFT NUMBER
= SN(3) = 21

	0	I
1	2RI	1PI
2	2LI	3RO
3	3LI	1LI

SCORE = 6 = BB(3)
SHIFT NO. = 13

	0	I
1	2RI	3LI
2	1LI	2RI
3	2LI	3PI

SCORE = 6
SHIFT NO. = 13

	0	I
1	2RI	3RI
2	3LI	2PI
3	1RI	2LO

SCORE = 6
SHIFT NO. = 12

	0	I
1	2RI	1RI
2	3LI	2PI
3	1RI	2LI

SCORE = 6
SHIFT NO. = 14

	0	I
1	2RI	1PI
2	3RO	2RI
3	3LI	1LI

SCORE = 6
SHIFT NO. = 11

	0	I
1	2RI	3LI
2	3RI	2PI
3	1LI	2LO

THE HIGH SCORING THREE-STATE MACHINES.

FIGURE 3.

LARGEST SHIFT NUMBER
(INCL. STOP) = 84

	O	I
1	2RI	4LI
2	3LO	3RO
3	3LI	1LI
4	4PO	1LO

SCORE = 11
SHIFT NO. = 53

	O	I
1	2RI	2LI
2	3LI	4RI
3	1RI	1LI
4	4PO	3RO

SCORE = 11
SHIFT NO. = 40

	O	I
1	2RI	3LO
2	1LI	4RI
3	2LO	2RI
4	1RI	4PI

SCORE = 11
SHIFT NO. = 63

	O	I
1	2RI	1LO
2	3RI	2LI
3	2LI	4RI
4	4PO	1RO

SCORE = 11
SHIFT NO. = 53

	O	I
1	2RI	4RO
2	3LI	1LO
3	1RI	2LI
4	4PO	3RO

SCORE = 11
SHIFT NO. = 46

	O	I
1	2RI	3RO
2	3LI	2PI
3	1RI	4LI
4	1RI	2LI

SCORE = 11
SHIFT NO. = 59

	O	I
1	2RI	4LI
2	3RI	2PI
3	1LI	1RO
4	3RI	4LO

SCORE = 11
SHIFT NO. = 43

	O	I
1	2RI	1LO
2	3RI	2PI
3	4LI	1RO
4	2LI	4LI

THE HIGHEST SCORING FOUR-STATE MACHINES FOUND AMONG THE MACHINES KNOWN TO STOP.

FIGURE 4.

number of moves allowed was 20 which is close to the value of SN(3). The figures shown for k = 4 involved a run of the program in which the maximum number of moves permitted was 90, and the maximum left and right excursions permitted were 12 and 11 spaces respectively. The fact that the best values for SN(4), LX(4) and RX(4) obtained were 84, 10, and 10 respectively immediately leads one to suspect that not only might better values exist for machines included in the collection of failures but that some of the failures could still be handled by the original heuristic program if the above restrictions were relaxed.

It was noted in § 4.2 that a k-state machine containing the quintuple (1, S, 2, R, S) is equivalent to some k-state machine containing the quintuple (1, S, 2, R, I) as far as the blank tape stopping problem and BB(k) are concerned. However, such a machine could conceivably yield a larger value for the "shift number" SN(k), but since it obviously cannot exceed by more than k-1 the value of SN(k) obtained by the generation procedure discussed, it was a simple matter to check for such a possibility among the machines which stopped. None was discovered for k = 3 or 4, but one case exists for k = 2 (Fig. 2).

Another point should be mentioned in line with Rado's "rules" for the BB(k) "contest" [16]. He evidently permits a change of symbol in the act of stopping since he branches to an added stop

state. When this possibility is allowed, no such machines are to be found among the tying entries for BB(3), nor were there any such machines arising from the 44 distinct three-state machines which had scores of exactly five marks. For k = 4 it is seen in Figure 4 that two of the machines stop on a space and therefore with Rado's stopping convention add one to their scores making $BB(4) \geq 12$ for the purposes of the "contest."

4.4 Remarks on the Efficiency of the Program

As it turned out the efficiency of the described generation procedure coupled with the heuristic solution of the stopping problem far exceeded the most optimistic hopes. While intuitively the method of generation appears to be the most efficient possible, it should be noted that incorporation of the intrinsic loop checking increased the efficiency further, and there are obviously checks for other redundancies which might have been incorporated. It seems likely that a most efficient method would depend upon the ability to recognize equivalent algorithms which in turn involves the halting problem.

It might be wondered now whether or not an attack upon the five-state problem is feasible. Solution of the three-state problem required about one minute except for the machines which failed. Time required for the four-state problem was 3 hours 45 minutes giving a ratio of 225:1 in time versus a ratio of 156:1 in the number

of machines generated. One would guess that a much larger ratio holds for the five-state case against the four-state case, and therefore that the time required would be prohibitive at least with present day digital computers.

V. CONCLUSION

In a later paper in which he discusses the non-computability of BB(n) and SN(n), Rado [16] asks whether or not there might exist a particular positive integer n_0 for which $BB(n_0)$ or $SN(n_0)$ are not effectively computable. While this might be objected to upon some sort of conceptual ground, it would not appear to be logically disputable without contradicting the established non-computability of these functions for all n. Not being satisfied with what he terms "the mere existence of a Turing machine" to compute $BB(n_0)$ and $SN(n_0)$, Rado (and also Lin [10]) suggests that there may possibly exist here some basis for redefining our concept of computability, at least in so far as the computability of a particular value of a "well-defined" integer is concerned. We have seen that there does exist an effective method for the determination of SN(3) and BB(3), and with the reduction thus far obtained it may be possible to effectively determine the values of SN(4) and BB(4). Of course this is not yet done, but in attacking the latter problem we found that while it appeared to be a very large jump in terms of magnitude, fewer than two percent of the cases could not be handled by the heuristics used in solving the three-state case.

Magnitude considered alone is not an obstacle in the formulation of the concept of computability of an integer function. We can

easily devise an algorithm (Turing machine) to effectively calculate any value of Ackermann's function [8, p. 272] which is given by the diagonal values of the function defined recursively by

$$\xi(0, b, a) = a + b,$$

$$\xi(n+1, 0, a) = \alpha(n, a),$$

$$\xi(n+1, b+1, a) = \xi(n, \xi(n+1, b, a), a),$$

where α is defined by

$$\alpha(n, a) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ a & \text{otherwise.} \end{cases}$$

Yet, for even relatively small values of the arguments it is not physically possible to carry out the computation of the value of this function nor even to comprehend the magnitude of the number involved.

It might seem that complexity becomes the real issue, although there is no precise logical definition of the meaning of the term. Among the 27 three-state machines on which the computer program failed to make a decision were two whose behavior was camouflaged to the extent that an inductive proof of the fact that they will never stop on a blank tape was achieved before the utter simplicity of their behavior became evident. But at that, the sequence of tapes in the operation history demonstrated a humanly recognizable pattern.

Consider what becomes involved when one is confronted with a Turing machine which writes, unknown to the observer, a never-ending computable sequence (in Turing's sense [20]) for some irrational or transcendental number. A proper illustration of this point may not even exist among the machines of small order, but we can describe a simple Turing machine of not very large order which serves to illustrate the matter in another way. The well-known "last theorem" of Fermat that the equation

$$x^n + y^n = z^n$$

does not possess integer solutions for $n > 2$ can be made equivalent to the stopping problem on a blank tape for a Turing machine. We need only to establish a suitable enumeration of all integer quadruples (x, y, z, n), $n > 2$, and construct the Turing machine to evaluate the given formula for each quadruple, stopping only if identity is ever achieved. Such a machine might be fairly elaborate, but it would not be difficult to construct and certainly could not be considered to be sophisticated.[1] While Fermat is said to have claimed knowledge of a simple proof of this theorem it has defied the attempts of mathematicians over the last several centuries to find a proof [2, p. 237]. Would we not be wise in assuming that the solution to the blank tape

---

[1] The reader is asked not to infer that a simplification of the theorem has been suggested!

stopping problem for the Turing machine described above would be similarly elusive?

House and Rado [15] have suggested the optimization of small (several state) Turing machines as a problem study source for the field of "artificial intelligence." They also mention attempts to solve the BB(3) problem in this light.[1] While it now appears that there are no particular three-state machines whose stopping problems on a blank input tape present any great difficulties for a suitably experienced human, the construction of a computer program to do this was, to say the least, not as trivial as some of the individual problems. While the absolute determination of BB(4) and SN(4) has not been made, it is expected from sample studies that a great reduction in the cases remaining might be achieved through mechanization of Procedure 2.11 and the modifications suggested for it in Chapter II. But it would not be a surprise to encounter other machines whose non-stopping behavior is camouflaged in some subtle way as opposed to possessing a pattern of uncomprehensibly large dimensions.

An attack on the problem of the determination of BB(5) and even BB(6) should be recommended as a source for study in the artificial intelligence field (or the field of <u>adaptive programming</u> if

---

[1] This was prior to the solution of the problem by Lin. See the footnote on page 5.

another term is desired) whether there exists any hope of obtaining the solution or not. Efforts at "mechanical proofs" of mathematical theorems starting from systems of axioms have had some success (cf. for instance Wang[23] or Gerlernter [6]). But what may be particularly needed for the continuation of this study is a collection of appropriate heuristics for the application of mathematical induction. The induction that was used here is implicit in the theorems that were applied and it was likewise implicit in the procedures recommended for future mechanization. Efforts in attempting to solve this "well-defined" problem for special cases may also serve to aid us in the determination of the limits of "intelligent automata" even if it be in only some statistical sense.

BIBLIOGRAPHY

1. Anderson, Sandra E. Some computational schemes equivalent to Turing machines. Master's thesis. Corvallis, Oregon State University, 1964. 31 numb. leaves.

2. Bell, Eric Temple. Mathematics, queen and servant of science. New York, McGraw-Hill, 1951. 437 p.

3. Brady, A. H., W. R. Stahl and H. E. Goheen. The calculation of a recursive function on a simulated Turing automaton. Paper submitted for presentation at the Congress of the International Federation for Information Processing, New York, 1965. (Summary, 4 p.)

4. Coffin, R. W., H. E. Goheen and W. R. Stahl. Simulation of a Turing machine on a digital computer. In: Proceedings of the Fall Joint Computer Conference, Las Vegas, 1963. Baltimore, Spartan Press, 1963. p. 35-43.

5. Davis, Martin. Computability and unsolvability. New York, McGraw-Hill, 1958. 210 p.

6. Gerlertner, H. Realization of a geometry-theorem proving machines. In: Proceedings of the International Conference on Information Processing, Paris, UNESCO, 1959, p. 273-282.

7. House, R. W. and T. Rado. An approach to artificial intelligence. In: Artificial Intelligence. New York, Institute of Electrical and Electronics Engineers, January 1963. p. 6-15. (IEEE Special Publication S-142)

8. Kleene, S. C. Introduction to metamathematics. Princeton, D. Van Nostrand, 1952. 550 p.

9. Lee, C. Y. Lecture notes. In: University of Michigan Engineering Summer Conference on Automata Theory, 1963. Ann Arbor, 1963. 368 p. (Paper bound collection)

10. Leeson, Daniel N. and Donald L. Dimitry. Basic programming concepts and the IBM 1620 computer. New York, Holt, Rinehart and Winston, 1962. 368 p.

11. Lin, Shen. Computer studies of Turing machine problems. Ph. D. thesis. Columbus, The Ohio State University, 1963. 35 numb. leaves.

12. Minsky, M. L. Some methods of artificial intelligence and heuristic programming. In: Proceedings of the Symposium on Mechanisation of Thought Processes, National Physics Laboratory, Teddington, England, 1958. Vol. I. London, H. M. Stationery Office, 1959. p. 5-27.

13. Minsky, M. L. Size and structure of universal Turing machine using tag systems. In: Proceedings of Symposia in Pure Mathematics. Vol. V. Recursive Function Theory. Providence, R. I., American Mathematical Society, 1962. p. 229-238.

14. Polya, G. Mathematics and plausible reasoning. Princeton, Princeton University Press, 1954. 2 vols.

15. Post, Emil L. Recursive unsolvability of a problem of Thue. The Journal of Symbolic Logic 12(1):1-11. 1947.

16. Rado, T. On non-computable functions. The Bell System Technical Journal, May 1962, p. 877-884.

17. Rado, T. On a simple source for non-computable functions. In: Proceedings of the Symposium on Mathematical Theory of Automata, New York, 1962. Ed. by Jerome Fox. Brooklyn, Polytechnic Press, 1963. p. 75-82. (Microwave Research Institute Symposia Series, Vol XII)

18. Shannon, Claude E. A universal Turing machine with two internal states. In: Automata Studies. Ed. by C. E. Shannon and J. McCarthy. Princeton, Princeton University Press, 1956. pp. 157-165.

19. Stahl, Walter R., Robert W. Coffin and Harry E. Goheen. Simulation of biological cells by systems composed of string-processing finite automata. In: Proceedings of the Spring Joint Computer Conference, Washington, D. C., 1964. Baltimore, Spartan Press, 1964. p. 89-102.

20. Turing, A. M. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, ser. 242:230-265. 1936. (43:544-546. 1937)

21. Turing, A. M. Computability and λ -definability. The Journal of Symbolic Logic, 2(4):153-163. 1937.

22. Trakhtenbrot, Boris A. Algorithms and automatic computing machines. Boston, D. C. Heath, 1963. 112 p.

23. Wang, Hao. Toward mechanical mathematics. IBM Journal of Research and Development. 4:2-22. 1960.

24. Von Neumann, John. Collected works. Vol. V. Ed. by A. H. Taub. New York, Macmillan, 1963. 784 p.

APPENDICES

APPENDIX A

A TURING MACHINE SIMULATOR FOR THE IBM 1620 COMPUTER

Background

In order to carry out research on the programming and behavior of Turing machines it is useful and possibly necessary to have a reliable means of simulating the Turing machines one wishes to describe. The human simulator is prone to error after not very many moves in a Turing computation. For this reason a program to simulate the Turing machine was written for the IBM 1620 computer. This computer was chosen primarily for its availability and secondly because it is a machine with a character-organized or variable word length memory (cf. Leeson and Dimitry[10]). This particular computer and others similar to it[1] that are presently in use come the closest to being a mechanization of Turing's hypothetical universal machine, where the instructions and "data" appear as a string of characters in the memory ("tape").

The first description of a general Turing machine simulation for a digital computer was published in 1963 by Coffin, Goheen,

---

[1] Such as the International Business Machine model 1401.

and Stahl [4]. This was done on a computer[1] with the more conventional word organized memory with the primary purpose of the simulation of algorithmic models of biological cells [19] although other uses have become evident [19]. Its "power" as a symbol manipulation device has also been discussed [3].

There are two essential differences between the earlier simulation technique [22] and that described here, in that the former (1) stores the Turing program as an unordered list of quintuples and (2) permits the calling of particular Turing algorithms to be applied to the tape.

The simulator for the 1620 computer stores the Turing program as an ordered table of triples, and the Turing instructions are "looked up" directly using their computed position in the table. This organization suffers from two disadvantages not possessed by the earlier published simulator. First, the instructions cannot be randomly ordered, a feature unique to the earlier simulator [4], and apparently not to be found in other types of programming languages. Second, the table type of organization wastes all of the space occupied by undefined entries, and in large Turing algorithms of a practical nature, experience will verify that the amount of such waste space

---

[1] The Scientific Data Systems model 920, which is also the computer used for the heuristic porgram described in Appendix C.

can be considerable.

However, the disadvantage may not be so real when one considers that the earlier simulation used a word computer with the most compact storage consisting of one quintuple per word, while in the 1620 simulator each quintuple would require 11 digits of memory versus 6 digits of memory for the triple, and the additional space requirements of the quintuples must be considered in contrast to the space "wasted" by the table of triples. It is also much less efficient to use the character-organized computer in a search of a list of quintuples than is the case for the word-organized computer. In the word-organized computer one examines a single word using a "compare on mask and skip if equal" type of instruction (Robert W. Coffin, private communication, 1964) while in the character organized computer one must compare several characters and then branch on a separate "test" instruction, the entire operation being much slower.

The second difference between the two simulators is due primarily to their original purpose: a specialized application intended for the earlier simulation and the purpose of studying particular Turing algorithms and Turing machine behavior intended for the latter.

Using the Simulator

The normal sequence of operation of the simulation program is the execution of the following routines in the order shown.

1. Load a new Turing machine.
2. Load table entries.
3. Load an "input tape."
4. Start the Turing machine on the tape.
5. Halt. Read data to determine the new point of entry into this operation sequence.
6. Dump out the tape configuration.
7. Halt. Same as before.

The general function of each step is as follows:

1. The entire area of computer storage not allocated to the simulator program is cleared and set to alphanumeric blanks. A data record is read to obtain the number of characters in the alphabet followed by the actual characters of the alphabet given in the exact order to be used in the Turing table. The alphabet must be a subset of the 48 alphanumeric character set of the 1620 computer (cf. Leeson and Dimitry [10]).

2. Data records containing the table entries are read. If no triple is defined in the data, the triple already appearing in the table is left undisturbed (assuming the table has not been cleared by step 1 above). A "halt" is a triple of the form q P s appearing in the (q, s) position of the table. If an undefined triple is encountered during "operation" of a Turing machine, this is considered to be an error, and a message is so printed. A special record is used to indicate that all of the table entries have been loaded.

3. A record is read to indicate where the input tape sequence is to be loaded relative to that portion of computer storage which has been allocated as tape for the Turing machine. A continuous string of symbols is then read into the computer until a special character called a "record mark" is encountered. This defines the end of the tape. Note that in practice a tape has a beginning and an end which requires the Turing programmer to know something in advance of the amount of "tape" required for his computation.

4. The Turing machine must be started in a particular state and on a particular tape location. A data record containing this information must be supplied. The machine

commences operation.

5. The machine continues operation until a "halt" command is encountered or until an error occurs (undefined symbol on tape, undefined table entry, tape limits exceeded). The program then must obtain a data record to direct its entry back into some point of this operation sequence.

6. Output consists of the following information:

(a). The number of "moves executed by the Turing machine since starting.

(b). The state the machine was last in.

(c). The tape location being scanned.

(d). The complete sequence of symbols on the tape between the defined limits of the tape.

7. As before, a record must be read to direct the entry of the program back into a particular point of its operating sequence.

Certain program options can be obtained through settings of the four console switches on the computer as follows:

Sw. 1 ON "Debugging" monitor in effect (permits intermittent output of tape configurations starting and ending at a specified number of moves and at a specified interval of moves).

	OFF	"Debugging monitor not in effect.
Sw. 2	ON	Input records taken from console typewriter.
	OFF	Input records taken from punched cards.
Sw. 3	ON	Output to console typewriter.
	OFF	Output to cards.
Sw. 4	ON	An output occurs after each move of the Turing machine during operation.
	OFF	No output during Turing machine operation.

Intermittent use of switch 4 is permitted even while the "debugging" monitor (switch 1) is effective.

The sequence of tape configurations shown in the various figures in this thesis were obtained using this simulator program. In Figure 5 is shown a sample input deck for the operation of the BB(2) machine on a blank input tape (where the "blank" in this case is the character "S" and not the alphanumeric blank character). In the lower portion of the figure are the tape configurations as they appear during the operation of the machine.

Internal Organization of the Simulator

The simulation program including input (compiling) routines, output routines, and the addition and multiplication tables used

INPUT

```
02 SYMBOLS SI                          TWO-STATE BUSY BEAVER MACHINE
STATE 001 CARD NO 1    2RI   2LI
STATE 002 CARD NO 1    1LI   2PI
STATE 003 CARD NO 1 Z
LOAD TAPE AT 0001
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSZ                    Z
START STATE 001 TAPE 0030
5 DUMP
```

OUTPUT

```
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSZ
 000000                                          001*
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSISSSSSSSSSSSSSSSSSSSSSSSSSSSSZ
 000001                                           002*
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSIISSSSSSSSSSSSSSSSSSSSSSSSSSSZ
 000002                                          001*
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSIISSSSSSSSSSSSSSSSSSSSSSSSSSSZ
 000003                                         002*
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSIIISSSSSSSSSSSSSSSSSSSSSSSSSSZ
 000004                                        001*
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIISSSSSSSSSSSSSSSSSSSSSSSSSZ
 000005                                        002*
FROM 00001 TO 00060 SSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIISSSSSSSSSSSSSSSSSSSSSSSSSZ
 000006                                        002*
MOVE 000006 SYMBOL (I) STATE 002 TAPE 0029
SSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIISSSSSSSSSSSSSSSSSSSSSSSSSZ
```

FIGURE 5. INPUT DECK FOR SIMULATOR AND MONITOR OUTPUT.

by the computer occupy approximately 5000 digits of storage. The simulator itself, i. e. that portion active during an unmonitored Turing machine operation, requires only 24 instructions and thus uses only about 300 digits of storage, a trivial amount compared to the remainder of the program.

During compilation the program, in addition to checking for the occurrence of an improper "move" (not L, P, or R), has to rearrange each triple slightly and load it into the proper table location. Each triple occupies 6 digits of storage:

$$\overline{x}\ x\ x\ s\ s\ m$$

The leading digit is flagged to define the start of the "word" containing the "triple", and the first three digits comprise the "branch state." The fourth and fifth digits comprise the alphanumeric code of the replacement character, and the sixth digit denotes the move: "2" for "R"; "0" for "P"; "-2" ("2" with a flag) for "L". The numeric value of the move is added to a number corresponding to the storage location of the tape "square" being scanned.

When the alphabet is originally defined, each symbol is assigned a number corresponding to its order of appearance in the declared alphabet. The assignment number times two is stored in a special table (the symbol table) according to the alphanumeric code of the symbol [10]. The assignment number is then used along with

the branch state and the number of symbols to compute the location of the next triple in the Turing table.

No further burden was put on the compiler even though it is evident that greater operation speed can be obtained by not using the standard alphanumeric codes internally. (The simulator processes approximately 100 entries per second during Turing machine operation.) However, this simulator was intended to be used normally in situations requiring considerable output and its efficiency in that respect would be very much reduced and furthermore the program would occupy even more storage to the detriment of the amount of space available for the Turing table and input tape.

APPENDIX B

A 24 STATE MACHINE TO COMPUTE $x!$

A binary Turing machine of 24 states was constructed to compute factorial of x employing the following strategy (shown for the specific value $x = 4$):

$$\ldots\ 0\ 0\ 0\ 0\ I\ I\ \underset{1}{\underline{I}}\ I\ 0\ldots$$

Starting at the second mark from the right in a contiguous string of x marks the machine "opens up" the string inserting spaces between the marks:

$$\ldots\ 0\ \underset{18}{\underline{I}}\ 0\ I\ 0\ I\ 0\ I\ 0\ I\ 0\ldots$$

The string is then closed as the $x-1$ spaces are tallied to construct the multiplier $x-1$ on the left:

$$\ldots\ I\ I\ \underset{22}{\underline{I}}\ 0\ 0\ I\ I\ I\ I\ 0\ldots$$

The string of x marks is then copied on the right a total of $x-2$ times, or until the multiplier $x-1$ has been totally deleted, losing one mark before each copy is made:

$$\ldots \underset{\overline{16}}{I}\ 0\ 0\ 0\ 0\ I\ I\ I\ I\ 0\ I\ I\ I\ I\ 0\ I\ I\ I\ I\ 0\ 0 \ldots$$

The x-2 spaces enclosed by the copies of x are then tallied while the string is closed, constructing the multiplier x-2 on the left of the string of x·(x-1) marks:

$$\ldots\ 0\ I\ \underset{\overline{22}}{I}\ 0\ 0\ I\ I\ I\ I\ I\ I\ I\ I\ I\ I\ I\ I\ 0\ 0 \ldots$$

The process continues recursively until a left hand multiplier consisting of one mark is constructed.

The table for the machine follows:

	0	I	Comments: 0	I
1	18 R 0	2 L 0	String open, condense.	Open string with a space.
2	3 R I	2 L I		
3	4 L 0	3 R I		
4	22 L 0	1 L I		
5	5 R 0	6 R I	Go right to multiplicand.	
6	7 R 0	6 R I		
7	8 L 0	6 R I	Start construction of another copy.	Continue right over right most copy.
8	8 L 0	9 R 0		
9	10 R 0	9 R I		

10 11 L I	10 R I		
11 12 L 0	11 L I		
12 13 L I	12 L I		
13 14 L 0	9 R 0	End of copy.	Continue while copying.
14 15 L 0	14 L I		Go left over copies.
15 16 L 0	14 L I		
16 16 L 0	17 L 0	Go left to the multiplier.	Delete mark from multiplier.
17 18 R 0	5 R I	End of multiplication.	Make another copy.
18 18 R 0	19 R 0		Close string.
19 20 R I	19 R I		
20 23 L 0	21 L I		
21 22 L 0	21 L I		
22 18 R I	24 L 0	Add mark to the multiplier	Delete right most mark of multiplier.
23 4 L 0	23 L I		
24 24 P 0	5 R I	Stop if multiplier is one mark.	Make copies if multiplier > 1.

APPENDIX C

A HEURISTIC COMPUTER PROGRAM FOR THE DETERMINATION OF BB(k) AND SN(k) FOR k = 2, 3, 4

We shall present here the logical details of the heuristic program which was written for the solution of the Busy Beaver problem for up to four states. This program was coded for the Scientific Data Systems model 920 digital computer, but it will be presented here in an algorithmic form that is essentially computer independent.

The generation "algorithm" was discussed in § 4. 3, but the routine to solve the stopping problem was referred to only as a "mechanization" of appropriate theorems and procedures of Chapter II. This mechanization will be covered in detail along with the simple heuristics used in directing the application of particular theorems. The various parts of the program will be numbered decimally as the need arises. It will be assumed that we are using a computer with a storage composed of binary "words", but it will be noted that this is not essential to the program.

The Program

Input.

Read into storage the following data:

The number of states k.

A "properly generated" Turing machine of not more than k states (a table of 2k-1 or fewer entries).

The sequence (array) of state-symbol coordinates of the above Turing machine given in the order of entry determined by operating it on a blank tape. This will be referred to as the "order-of-entry array."

The "best values" known for BB(k), SN(k), LX(k) and RX(k).

The total number of machines generated thus far for the k-state problem and the number that stopped (these for tallying purposes only).

The known values of BB(k-1), SN(k-1), LX(k-1) and RX(k-1).

Setup.

"Tag" the position LX(k-1)+1 places to the left of the starting point of the storage array designated as the "tape" for a Turing machine simulator. Also "tag" the position RX(k-1)+1 places to the right of the starting point.

Determine the number of entries in the input machine. (Comment. The most recent entry as determined by the order-of-entry array will be referred to as the pivotal entry.)

(1) Determine the maximum branch state mentioned among the non-pivotal entries.

Define the maximum pivotal state to be equal to the maximum state mentioned plus one.

Return to (3) if (1) was reached from "Retrace" or "Advance". Otherwise continue with the Intrinsic loop check.

(2) Here check to see that the pivotal entry does not complete a closed set of entries which does not contain a "stop" or undefined entry. If such a closed set does not exist then go the the stopping heuristics (5). Otherwise we must Increase the pivotal entry.

(Comment. An entry was actually encoded as a binary number as follows:

$$\text{space} \text{ -- } 0 \cdot 2^0$$

$$\text{mark} \text{ -- } 1 \cdot 2^0$$

$$\text{left} \text{ -- } 0 \cdot 2^1$$

$$\text{right} \text{ -- } 1 \cdot 2^1$$

$$\text{state} \text{ -- } \text{state} \cdot 2^2$$

Thus by adding one to an entry we increase it according to the ordering scheme of § 4. 3.)

(3) Increase the pivotal entry triple to the next triple in succession. (Comment. See § 4. 3.)

Does the branch state of the new triple exceed the

maximum pivotal state? If not, then go to (2). If yes, then Retrace to previous entry.

(4) Clear the pivotal entry (render undefined). Subtract one from the entry count to define the previous entry in the order-of-entry as the pivotal entry.

If the pivotal entry is the first entry then the program is through. Go to (14). Otherwise go to (1).

Heuristics for stopping.

(5) If neither the maximum state mentioned nor the branch state of the pivotal entry equal the value of k then the current machine must stop within the limits of SN(k-1), LX(k-1) and RX(k-1), or else it does not stop at all. (Comment. We here use the Turing machine simulator previously set up for this purpose.) If the machine does not stop within the prescribed limits for k-1 states then go to (3). Otherwise, we have encountered an undefined entry so we must

Advance.

(6) Add one to the entry count. If the count exceeds 2k-1 then no more entries are possible, so go to (4). Otherwise, the newly reached undefined entry is the pivotal entry so set it equal to the triple 0 R I and go to (1).

If either the maximum state mentioned or the pivotal branch state equals k, then run the machine for not more than 45 moves. Save in appropriate arrays the history of state-symbol pairs, maximum right and left excursions, and (redundantly) the tape configurations.

(6.1) If the machine stops (reaches an undefined entry) before 45 moves have occurred then

Reckon score for BB(k), SN(k), LX(k), RX(k).

If the "shift number" exceeds the "best" current value then revise the "best" value and print out the machine. Also print out any machine whose "shift number" is within k-1 of the "best " value.

Revise the "best" value for LX(k) or RX(k) if either is exceeded and print out the machine.

Count the number of marks on the tape. If the count exceeds the best value of BB(k), revise the best value and print out the machine. Also print out the machine if the tally ties the best value.

If the tally exceeds the best value of BB(k) less two, then determine whether or not the "stop" occurred on a "space." If so, determine whether or not there exists an undefined entry under the mark position of

the table, and if there is, then a tie or greater value for BB(k) can be attained using a P-move with a change of symbol before a "halt" (see § 4. 2). Print out such a machine.

Go to (6) (Advance).

If the machine does not stop before 45 moves or before exceeding the confines of a 24 square tape, then we go to the

Choice of theorem heuristics.

(7) If the machine position exceeded the right hand limit of the tape before exceeding the limit on the number of moves, go to (9). If similarly the machine exceeded the left limit of the tape, go to (9). Compare the most recent right maximum location with that of 1 0 moves earlier. If it has changed, then a right traveling loop is possible; so go to (1 0). Otherwise, compare the most recent left maximum location with that of 1 0 moves earlier. If it has changed, then a left traveling loop is likely so go to (11). Otherwise, no change on either end for 1 0 or more moves suggests we make a

Fixed loop conditions check.

(8) Search back through the location history not already considered, but not prior to half the number of moves that have occurred, until a scan location is found which equals the most recent location. If no such equality occurs go to (12). Otherwise, compare the sequences of state-symbol pairs starting back from the most recent configuration to that which starts back with the pair corresponding to the equal location just found. If at some point prior to reaching the second overlapping pair, the two sequences differ, then go to (8). Otherwise, it has been determined that a fixed loop exists; the machine will not stop, so go to (3) to increase the pivotal entry.

Right or left traveling loop possibility check.

(9) Compare the most recent right maximum location with that of 10 moves earlier. If they are equal go to (11) to check for a left traveling loop. Otherwise compare the most recent left maximum location with that of 10 moves earlier. If they are not equal the string is growing on both sides so go to the "Reverse" procedure (12). Otherwise make a

Right traveling loop check.

(10) Search back through the sequence of right maximum locations until the most recent change in maximum occurs.

(10.1) Search back through the sequence of right maximum locations from the earliest change in maximum thus far discovered until the next (earlier) change in maximum occurs but not prior to half the number of moves that have occurred. If no such change is discovered, then go to the "Reverse" procedure (12). Otherwise, compare the sequence of state symbol pairs ending at the most recent change in maximum and beginning at that pair corresponding to the earliest change in maximum thus far discovered with the sequence of the same length which ends at the point where the former begins. If the sequences compare we have a right-traveling loop so go to (3). Otherwise, continue with this procedure at (10.1).

Left traveling loop check.

(Comment. This procedure is the mirror image of the right traveling loop check procedure.) If a left traveling loop occurs go to (3). Otherwise this test fails so go to (12), the

Reverse running procedure.

(12) Has this procedure been applied to this machine already? If yes, then all tests have failed, so print out this fact along with the machine in question, assume the machine will not stop and go to (3). If not, subtract the entry count from 2k to determine the number of undefined entries in the table of this machine.

(12.1) Search for the first missing (undefined) entry not previously found. If there are no more missing entries then the test has succeeded so go to (3). Otherwise, set the maximum level of the tree equal to 11 (determined by the word size of the computer) and the maximum number of derived nodes equal to 23 (determined by the amount of storage allocated.) Store the state-symbol pair of the missing entry in the first word of the first node. Set the scan bit in the center of the second word of the first node. Set the center bit of the third word(which is the tape segment) equal to 0 or 1 accordingly as the symbol of the pair is a blank or a mark. Set the center bit of the fourth word (the defined tape region mask) equal to 1 with all other bits equal to 0. (Comment. The

first node is comprised of four words which now appear as follows:

s-s pair:	00000000 00000000 0000sssx
scan location:	00000000 00001000 00000000
tape segment:	00000000 0000x000 00000000
defined region:	00000000 00001000 00000000

The 24 nodes allowed comprise an array totaling 96 words.) Designate the first node as the originating node. Point to the second node designating it as the new node. Set the count of originating nodes equal to one. Set the count of new nodes and the count of all nodes equal to zero.

(12.2) Search the table for a triple with a branch to the state of the originating node. If none is found that has not already been considered, then point to the next originating node and go to (12.2). If the originating nodes are exhausted go to (12.3). If a branch is found, then place the corresponding state-symbol pair in the first word of the new node. Next form the scan location of the new node from that of the originating node by shifting left if the move of the triple is right and shifting right if the move is left. Form the tape

segment of the new node from that of the originating node by inserting the symbol from the state-symbol pair into the position corresponding to the new scan location. Form the new defined area mask by (logically) merging the mask from the originating node with the new scan location.

With the machine in the configuration of the new node, execute one move (forward) and compare the resulting tape segment with the tape segment of the originating node over the defined region (mask) of the originating node. If the tape segments so compare, then the configuration of the originating node could have been derived from that of the new node, so add one to the count of the new nodes and designate the next node following the present new node as the new node. If the total count of nodes exceeds 23, then terminate this test and go to (13). If the segments do not compare, then the derivation is not possible, so leave the new node as designated and go to (12. 2).

(12. 3) If the count of new nodes is zero, then further expansion of the tree from the present undefined entry is not possible, so go to (12. 1). If the count is not zero then set the count of originating nodes equal to

the count of new nodes and designate the next node (in the array) following the originating node as the originating node. Set the count of new nodes to zero. Increase the count of tree levels. If the level is less than 11 then go to (12. 2). Otherwise

Terminate the test.

(13) (Comment. All tests thus far tried have failed.) Proceed to run the machine for not more than 45 additional moves repeating the heuristic procedure starting at (6. 1).

Output.

(14) The following data is given as output when the program ends:

The best values of BB(k), SN(k), RX(k), and LX(k).

The total number of machines generated.

The number of the above machines known to stop.

The number of machines for which the program failed to make a decision on the blank input tape halting problem.

(Comment. Other items of transient interest were also determined but these will not be discussed. Also, the facility to terminate the program at any

point between generated machines was incorporated. Thus, the program is able to resume with the last "properly" generated machine as input.)

The program as written for the SDS 920 computer encompasses nearly 2, 000 instructions, and in addition, a "trace" routine and an "octal storage correcting" routine were written to use in the debugging of the program. Since the main effort was involved in the writing and debugging of the program (execution time for the $k = 3$ case was slightly in excess of one minute!) a trace option was included within the program to be actuated by an appropriate setting of the "break point" switches.

It is interesting to note that the computer time involved in debugging the program exceeded even the running time of 3 hours 45 minutes for the four-state case. There did not appear to have been any compromises in the program which might have traded a significant amount of programming or debugging effort for the expense of additional running time during final execution. While use of one of the present day scientific programming languages might have been possible and would have saved the effort required to learn the "machine language", it would not have simplified the logic of the program in any material way and would almost certainly have decreased the efficiency of execution to such a degree that the

program could not have been run economically for the four-state case. There in fact do not seem to be any programming languages in use today which might have aided significantly in the writing of this program. It is not a constructive thing to say but it is true, never-the-less, that in the application of digital computers we are a long way from the possession of a mechanical means of translation of an algorithm or program from a form similar to that presented here into an operational computer program.

APPENDIX D

THE 27 THREE-STATE MACHINES ON WHICH THE COMPUTER PROGRAM FAILED TO MAKE A DECISION

Of the 3, 361 machines generated for consideration for the case of three states, on only 27 did the heuristic computer program fail to make a decision on the blank input tape halting problem. These failures are shown in Figure 6. Each of the 27 machines was run on the Turing machine simulator discussed in Appendix A to obtain the details of their behavior for the first 60 moves. Of the 27 machines all but two demonstrated the simple "sweep-across" pattern covered by the variations on Procedure 2.11 discussed in Chapter II. (This procedure had not, of course, been mechanized.) An example of this behavior is shown in Figure 7 for the machine which is failure number 6 of the group which is shown in Figure 6.

The remaining two machines (numbers 2 and 21 of Figure 6) demonstrated a completely different type of behavior which resulted in a much slower rate of growth. The first 60 moves of the machine (failure number 2 of Figure 6)

	O	I
1	2 R I	3 L I
2	1 L O	2 R O
3	1 L I	---

01		
	O	I
1	2RI	3LO
2	1LO	1RO
3	1LI	===

02		
	O	I
1	2RI	3LI
2	1LO	2RO
3	1LI	===

03		
	O	I
1	2RI	1LI
2	1LI	3RI
3	===	2RI

04		
	O	I
1	2RI	2LO
2	1LI	3RO
3	2RI	===

05		
	O	I
1	2RI	3LO
2	1LI	1RO
3	1LI	===

06		
	O	I
1	2RI	3LO
2	2LI	1RI
3	===	1LI

07		
	O	I
1	2RI	3LO
2	3LO	1RI
3	===	1LI

08		
	O	I
1	2RI	===
2	3LO	2RO
3	3LI	1LI

09		
	O	I
1	2RI	===
2	3LO	1RO
3	1LI	2LI

10		
	O	I
1	2RI	===
2	3LO	1RO
3	1RO	2LI

11		
	O	I
1	2RI	===
2	3LO	2RI
3	1LI	3LI

12		
	O	I
1	2RI	1RI
2	3LO	===
3	1RO	3LI

13		
	O	I
1	2RI	1RI
2	3LO	===
3	1RI	3LI

14		
	O	I
1	2RI	===
2	3LI	2RO
3	2LI	1LI

15		
	O	I
1	2RI	===
2	3LI	2RO
3	3LI	1LI

16		
	O	I
1	2RI	2LO
2	3LI	2RI
3	===	1LI

17		
	O	I
1	2RI	3LI
2	3LI	2RI
3	===	1LI

18		
	O	I
1	2RI	3RI
2	3LI	===
3	1RO	2LO

19		
	O	I
1	2RI	===
2	3LI	1RO
3	2RO	2LO

20		
	O	I
1	2RI	===
2	3LI	1RO
3	2RI	2LO

21		
	O	I
1	2RI	===
2	3LI	1RI
3	1RO	3LO

22		
	O	I
1	2RI	===
2	3LI	2RI
3	1LI	3LI

23		
	O	I
1	2RI	1RI
2	3LI	===
3	1RO	3LI

24		
	O	I
1	2RI	1RI
2	3LI	===
3	1RI	3LI

25		
	O	I
1	2RI	===
2	3RO	2RO
3	3LI	1LI

26		
	O	I
1	2RI	1LI
2	3RI	2RI
3	1LO	===

27		
	O	I
1	2RI	1LI
2	3RI	2RI
3	1LI	===

FIGURE 6. THE 27 THREE-STATE FAILURES.

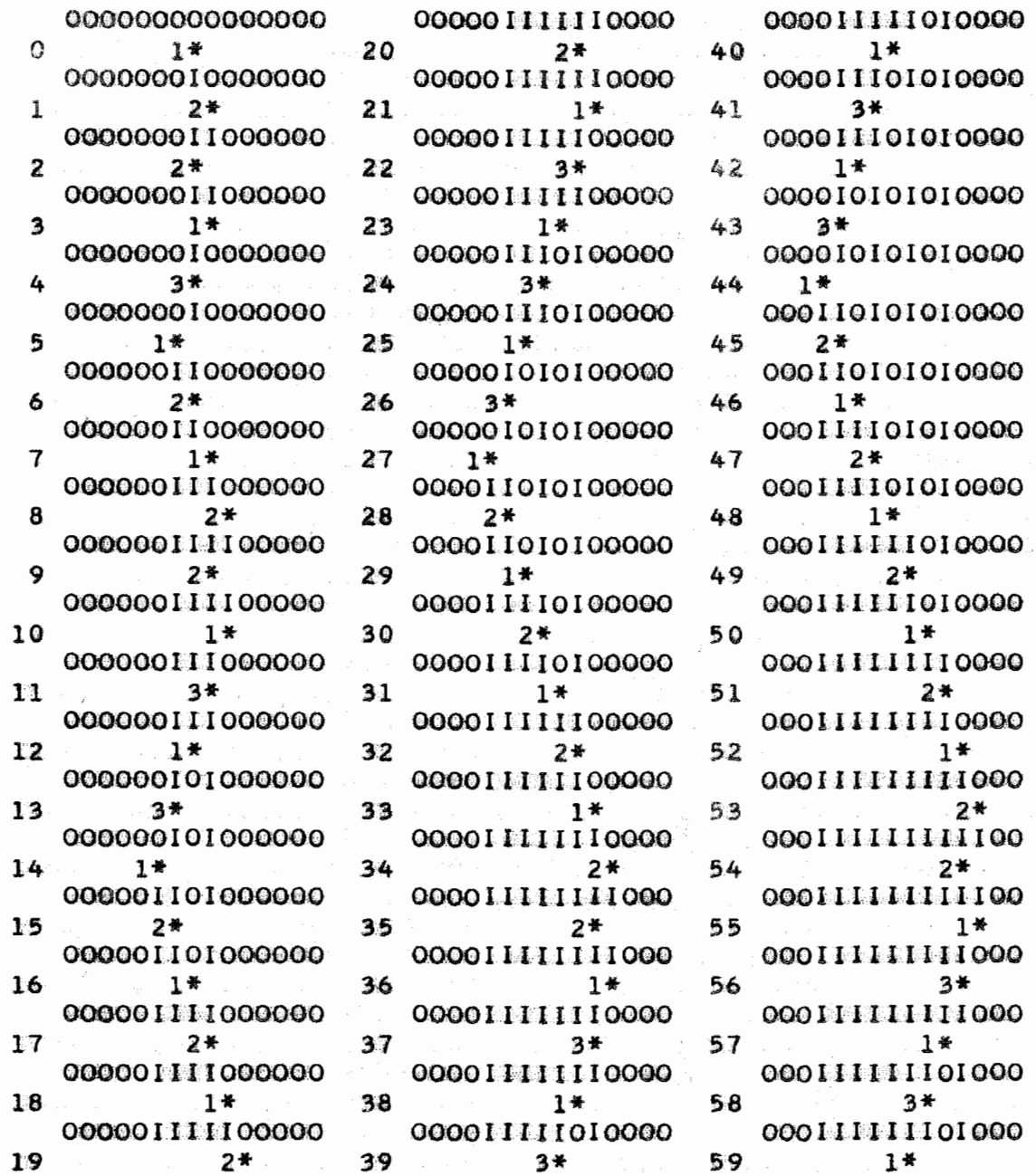

FIGURE 7. BEHAVIOR OF FAILURE NUMBER SIX.

are shown in Figure 8. It was not until after an algebraic expression for the recursion was discovered and a proof by induction was made to demonstrate that the machine will never stop, that the really simple behavior of the machine became evident. Although an argument which was discovered later makes the decision on these two machines trivial, an inductive proof that the above machine will never stop will be given for its instructive value. The less obvious but more elegant hindsight argument will also be given.

Assume a correspondence of the squares of the tape with the integers similar to the correspondence of the definition given in § 2.1. The expression $[a, b]$ which denotes all integers k, $a \leq k \leq b$, will be used to denote the ath through bth squares of the tape. Suppose that initially $[-2n, 1]$ are all blank squares, and after starting in state one on square zero let us assume that after $(3 \cdot 2^{n+1} - 2n - 7)$ moves in $[-2n, 1]$ we arrive at the configuration

$$\ldots 0\,0\,0 \ldots 0\,0\underset{3}{\underline{0}}\,I\,I\,I \ldots I\,I\,I\,0\,0 \ldots 0\,0\,0 \ldots$$

2n-1 marks

where square number one corresponds to the symbol 0 on the right of the string of marks. Then in two subsequent moves we obtain

$$\ldots 0\,0\,0 \ldots I\underset{2}{\underline{I}}\,I\,I\,I \ldots I\,I\,I\,0\,0 \ldots 0\,0\,0 \ldots$$

which is the configuration now after $(3 \cdot 2^{n+1} - 2n - 5)$ moves. We

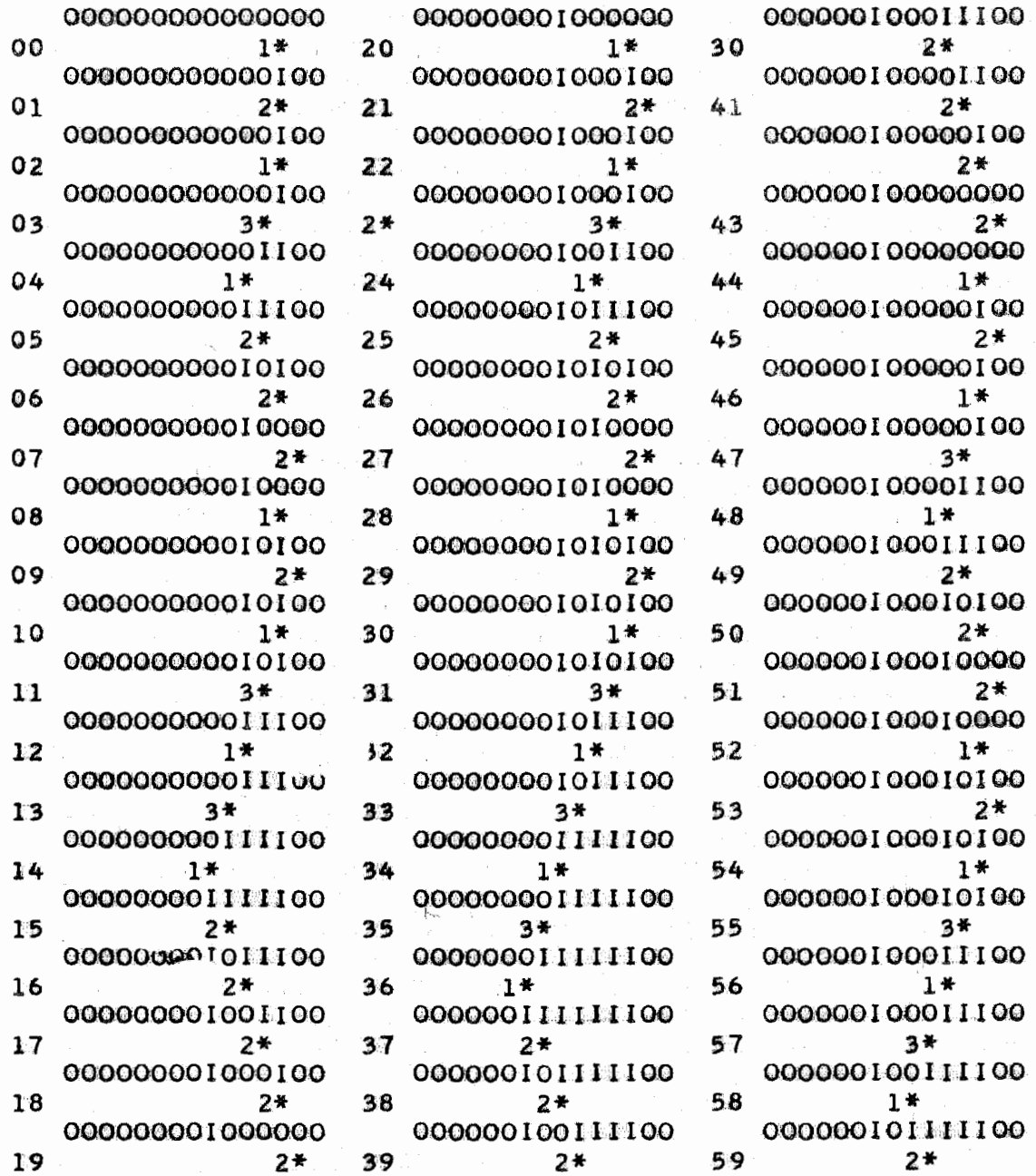

FIGURE 8. BEHAVIOR OF FAILURE NUMBER TWO.

are now in state 2 looking at a mark with 2n-1 contiguous marks to the right of the scanned symbol. The mark now being scanned will be replaced by a blank (0) and the machine will move right one square and remain in state 2. This will be repeated until all of the marks on the right are erased and we are at square 1 in state 2 looking at the space. Thus after an additional 2n moves we arrive at the configuration:

$$\ldots\ 0\,0\,0\ldots\ I\ 0\,0\,0\,0\ldots\ 0\,0\,0\,\underset{\overline{2}}{0}\,0\ldots\ 0\,0\,0\ldots$$

having made a total of $(3\cdot 2^{n+1}-5)$ moves. We then leave the space, move left and go into state 1:

$$\ldots\ 0\,0\,0\ldots\ I\ 0\,0\,0\,0\ldots\ 0\,0\,\underset{\overline{1}}{0}\,0\,0\ldots\ 0\,0\,0\ldots$$

having made a total now of $(3\cdot 2^{n+1}-4)$ moves. Now, by hypothesis, in $(3\cdot 2^{n+1}-2n-7)$ moves we will arrive at the configuration

$$\ldots\ 0\,0\,0\ldots\ I\ \underset{\overline{3}}{0}\,I\,I\,I\ldots\ I\,I\,I\,0\,0\ldots\ 0\,0\,0\ldots$$

having made a total number of $(3\cdot 2^{n+2}-2n-11)$ moves. In two more moves we arrive finally at the configuration:

$$\ldots\ 0\,0\,0\ldots\ \underset{\overline{3}}{0}\,I\,I\,I\,I\,I\ldots I\,I\,I\,0\,0\ldots\ 0\,0\,0\ldots$$

having made a final number of $(3\cdot 2^{n+2}-2n-9)$ moves. This shows that in $(3\cdot 2^{n+1+1}-2(n+1)-7)$ moves always staying inside the interval

$[-2(n+1), 1]$ we will arrive at the configuration

$$\ldots\ 0\ 0\ 0\ \ldots\ \underset{3}{0}\ I\ I\ I\ I\ I\ \ldots\ I\ I\ I\ 0\ 0\ \ldots\ 0\ 0\ 0\ \ldots$$

$2(n+1)$ -1 marks

It can be verified from Figure 8 that the hypothesis is true for a particular value of n, and thus we have demonstrated that the machine will not stop. One may prove the same result in the following artificial way. If one considers a binary number encoded two symbols to the "bit" as follows:

"0" -- 0 0,

"1" -- I 0,

one will note immediately that whenever the Turing machine is scanning the square number one that a binary number encoded in the above fashion lies to the left. The immediate act of the machine is to add "1" to this "binary" string changing all "1's" from the code I 0 to the intermediate code I I as the carry is propogated left to the first "0", encoded 0 0. When the first "0" is encountered it is changed to the redundant code I I also, but as the machine returns right to its starting point the last I I is changed to I 0, the code for digit "1", and all the redundant codes I I are changed to 0 0, the code for "0".

While a mathematician might feel that both of the above arguments in proof are so trivial as to not warrant further discussion,

to the heuristic programmer or other investigator concerned with the concept of "machine intelligence" it would be interesting to decide which of the two arguments is the more "sophisticated." Considered in this light the simplicity of the second argument is perhaps deceptive!