

AN ABSTRACT OF THE THESIS OF

DALE ROBERT COMSTOCK for the Ph. D. in Mathematics
(Name) (Degree) (Major)

Date thesis is presented January 13, 1967

Title AN OPERATOR SCHEME FOR COMPUTATION WITH TURING
MACHINES

Abstract approved Redacted for privacy
(Major professor)

An operator scheme for composition of Turing machines is developed and applied to the computation of the recursive functions over an arbitrary alphabet.

AN OPERATOR SCHEME FOR COMPUTATION
WITH TURING MACHINES

by

DALE ROBERT COMSTOCK

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

DOCTOR OF PHILOSOPHY

June 1967

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

January 3, 1967

Typed by Carol Baker

ACKNOWLEDGMENT

The author wishes to thank Professor Goheen for his helpful advice and encouragement during the past several years of graduate study. His patience and support during a critical period of illness in my family has been an inspiration.

Financial support of the National Science Foundation in the form of institutes for mathematics teachers is gratefully acknowledged. Without this support, the author may never have begun his graduate studies. The author also wishes to acknowledge the support of the administration of Central Washington State College for making the computing facilities available.

To my wife goes infinite thanks for the hardships she has carried during my graduate studies.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
II. TURING MACHINES AND COMPUTABILITY	3
Introduction to the Concept of a Turing Machine	3
The Definition of a Turing Machine	5
The Composition of Turing Machines	17
Some Elementary Turing Machines	22
An Operator Scheme for Composing Turing Machines	24
Turing Computability	
III. RECURSIVE FUNCTIONS OVER AN ARBITRARY ALPHABET S AND THEIR COMPUTABILITY	31
Definition of Recursive Functions over S	31
Further Turing Machines	34
Computability of Recursive Functions by Turing Machines	38
IV. CONCLUSION	44
BIBLIOGRAPHY	46

AN OPERATOR SCHEME FOR COMPUTATION WITH TURING MACHINES

CHAPTER I

INTRODUCTION

As a result of the work of Turing, Post, Kleene and Church [12, 10, 8, 4], it is now widely accepted that the concept of "computable" as applied to a function (not necessarily defined for all arguments) of natural numbers is correctly identified with the concept of "partial recursive". The usual proofs [6, 7, 8] that all partial recursive functions of natural numbers are computable have consisted of proofs that all recursive functions can be computed by Turing machines over an alphabet with the stroke symbol, 1 , by representing the natural number n by $n+1$ strokes.

In this paper, we develop the concept of partial recursive function over an arbitrary alphabet S (not necessarily the natural numbers, although our definition agrees with the definition of recursive function of natural numbers in the special case where S consists of just the stroke and space symbols) and show the computability of such functions by means of some particularly elementary Turing machines. We first define Turing machines and Turing computability and develop an operator scheme for composition of machines in Chapter II. The concept of partial recursive function over an arbitrary alphabet is

developed in Chapter III. Then the computability of these functions is shown by means of the operator scheme developed in Chapter II.

CHAPTER II

TURING MACHINES AND COMPUTABILITY

Introduction to the Concept of a Turing Machine (TM)

In 1936, Turing [12] devised a computational scheme (it has become known as a Turing machine) having the concept of an "algorithm" as a strong intuitive basis. In fact, algorithm is often replaced mathematically by the more precise concept of Turing machine or other logically equivalent scheme such as the λ -definability of Church [4].

Turing considered a "tape" divided into squares, each capable of bearing a "symbol". The machine is only capable of a finite number of conditions (states). The machine is supplied with a tape running through it in such a way that at any moment there is just one square ("the scanned square") bearing a symbol which is "in the machine". The symbol on the scanned square is called the "scanned symbol". The scanned symbol is the only one of which the machine is directly aware. However, by altering its states, the machine effectively remembers some of the symbols which it has "seen" (scanned) previously. The possible behavior of a machine at any moment was uniquely determined by the state and the scanned symbol and could be of the following types:

- i. Mark a new symbol on the scanned square;
- ii. Erase the scanned symbol;
- iii. Change the scanned square by shifting one place to the right or left;
- iv. Change the state of the machine.

Some of the symbols written down on the tape will form the sequence of symbols of the value being computed. Others are notes to "assist the memory".

It was Turing's contention that the above operations include all those which are used in the computation of a number.

The act of erasing can be eliminated by including the "blank symbol" as part of the set of symbols and consider erasing as equivalent to the printing of a blank [11].

Where Turing considered a tape infinite in one direction only [11], Kleene [8], Davis [6], and Hermes [7] follow Post [10] (as we will) in using a tape that is infinite in two directions. This presents no essential restriction.

In our development we shall follow Kleene [8] in restricting the behavior of a machine in a given state scanning a given symbol to the following triple:

- i. Mark a new symbol on the scanned square (which may be the same symbol);

- ii. Change the scanned square by shifting left one square, shifting right one square, or staying in place (center move);
- iii. Change to a new state (which may be the same state).

The moves described above will be designated by "l", "r", "h" for left, right and center moves respectively.

To carry out the computation of a Turing machine, the machine is started in a particular state over an "input tape" which is blank except for at most some finite number of squares. The computation is finished when the machine stops, i. e. when it reaches an explicit command to halt (e. g. the triple qhs with the machine in state q scanning symbol s) or a state symbol pair for which no commands are defined.

With the above intuitive discussion as a basis, we proceed to give a formal definition of a Turing machine in the next section following Anderson [1] although some modifications and shifts of emphasis are made.

Definition of a Turing Machine

Given two sets A and B , by a function f from A to B (designated by $f: A \rightarrow B$), we mean a subset of $A \times B = \{(a, b) \mid a \text{ in } A, b \text{ in } B\}$ such that if (a, b) is in f and (a, c) is in f , then $b = c$.

Definition 2.1. Let Q, M, S be disjoint finite sets as follows:

$$Q = \{q_0, q_1, \dots, q_m\};$$

$$M = \{r, l, h\};$$

$$S = \{s_1, s_2, \dots, s_n\}.$$

Q is the set of states, M the set of move orders and S the set of symbols. Let s_0 denote the blank symbol, \square , and let $A = S \cup \{s_0\}$. A is called an alphabet. Let δ, μ, λ be functions defined on $Q \times A$ as follows:

$$\delta: Q \times A \rightarrow Q;$$

$$\mu: Q \times A \rightarrow M;$$

$$\lambda: Q \times A \rightarrow A.$$

δ is the next state function, μ is a move order, and λ is the new symbol function. Let $T: Q \times A \rightarrow Q \times M \times A$ be a function defined on $Q \times A$ by means of δ, μ, λ as follows: for (q_i, s_j) in $Q \times A$ define

$$T(q_i, s_j) = (\delta(q_i, s_j), \mu(q_i, s_j), \lambda(q_i, s_j)).$$

T is called a Turing table.

T is conveniently represented as a matrix with $m+1$ rows and $n+1$ columns as follows:

T	s_0	s_1	\cdots	s_j	\cdots	s_n
q_0	$T(q_0, s_0)$	$T(q_0, s_1)$	\cdots	$T(q_0, s_j)$	\cdots	$T(q_0, s_n)$
q_1	$T(q_1, s_0)$	$T(q_1, s_1)$	\cdots	$T(q_1, s_j)$	\cdots	$T(q_1, s_n)$
\vdots	\vdots	\vdots	\cdots	\vdots	\cdots	\vdots
\vdots	\vdots	\vdots	\cdots	\vdots	\cdots	\vdots
q_i	$T(q_i, s_0)$	$T(q_i, s_1)$	\cdots	$T(q_i, s_j)$	\cdots	$T(q_i, s_n)$
\vdots	\vdots	\vdots	\cdots	\vdots	\cdots	\vdots
\vdots	\vdots	\vdots	\cdots	\vdots	\cdots	\vdots
q_m	$T(q_m, s_0)$	$T(q_m, s_1)$	\cdots	$T(q_m, s_j)$	\cdots	$T(q_m, s_n)$

s_0, s_1, \dots, s_n serve as column indices and q_0, q_1, \dots, q_m as row indices.

For notational convenience, we will often write $q_i m s_j$ for an arbitrary element (q_i, m, s_j) in $Q \times M \times A$.

Definition 2.2. q_0 will be called the initial state. If $T(q_i, s_j) = q_i h s_j$ for some i and j then q_i is called a terminal state.

q_0 will sometimes be denoted by q_T . q_T , by convention, will always be the first state appearing in the table T .

Definition 2.3. Let $\pi: I \rightarrow A$ be a function defined on the integers I with values in A . π is called a tape. A tape π such that $\pi(i) = s_0$ for all but a finite number of i in I is called a Turing tape.

An alternative definition of the blank symbol might be obtained from the functions π as follows:

Let $\pi: I \rightarrow A$ be a function defined on I with values in A . An element in A which is the image of an infinite set of integers will be called a "blank". Only tapes which define one blank element would be admitted as Turing tapes.

The arguments i in I of π may be regarded as squares of a tape as shown

...	$\pi(i-2)$	$\pi(i-1)$	$\pi(i)$	$\pi(i+1)$	$\pi(i+2)$...
...	$i-2$	$i-1$	i	$i+1$	$i+2$...

considering cell i as containing the symbol $\pi(i)$. A square containing s_0 is said to be blank.

From the above definition, we see that a Turing tape contains a finite word over S or a finite sequence of words over S separated by blanks. In the remainder of this paper, we will consider

only Turing tapes and will denote the set of all Turing tapes over A by πA .

In order to introduce change into our so far static picture of a Turing machine, we want to indicate in a single expression the entire condition of a Turing machine at a given instant. We do this with the concept of a complete configuration.

Definition 2.4. A complete configuration is a triple (q_i, π, n) with q_i in Q , π in πA and n in I .

Let $X = Q \times \pi A \times I$, i. e. X is the set of all complete configurations. (q_i, π, n) gives a description of the machine at a given instant, i. e. the internal state q_i , the tape π upon which the machine acts, and the cell n under scan at the instant.

A complete configuration $x \equiv (q_i, \pi, n)$ in the history of a Turing machine leads to the application of $T(q_i, \pi(n))$ of the Turing table T to determine the next complete configuration. This results in the natural definition of the function $F: X \rightarrow X$ defined on X by means of the Turing table T as follows:

Definition 2.5. Let $F: X \rightarrow X$ be defined by

$$F(q_i, \pi, n) = (\delta(q_i, \pi(n)), \bar{\pi}, \bar{n}) \text{ if } T(q_i, \pi(n)) \text{ is defined}$$

where

$$\bar{n} = n-1 \quad \text{if} \quad \mu(q_i, \pi(n)) = \ell ;$$

$$\bar{n} = n \quad \text{if} \quad \mu(q_i, \pi(n)) = h ;$$

$$\bar{n} = n+1 \quad \text{if} \quad \mu(q_i, \pi(n)) = r ;$$

and

$$\bar{\pi}(k) = \pi(k) \quad \text{if} \quad k \neq n ;$$

$$\bar{\pi}(k) = \lambda(q_i, \pi(n)) \quad \text{if} \quad k = n ;$$

$F(q_i, \pi, n)$ is undefined if $T(q_i, \pi(n))$ is undefined. F is called the consecutive configuration function.

Given a Turing tape π and a Turing table T , successive applications of F beginning with the initial state q_T describes the transformation of a given (input) tape into another (output) tape (provided F reaches a complete configuration involving a terminal state q_i and symbol s_j such that $T(q_i, s_j) = q_i h s_j$).

If q_i is terminal and $T(q_i, s_j) = q_i h s_j$ where s_j is the scanned symbol, then further application of F does not change the tape. This suggests the following theorem:

Theorem 2.1. If q_i is terminal and $T(q_i, s_j) = q_i h s_j$ where $\pi(n) = s_j$; then $F^m(q_i, \pi, n) = (q_i, \pi, n)$ for $m \geq 1$ where $F^1 = F$ and $F^m(q_i, \pi, n) = F(F^{m-1}(q_i, \pi, n))$ for $m \geq 2$.

Proof: Let q_i be terminal and $T(q_i, s_j) = q_i h s_j$ where $s_j = \pi(n)$. Then $F(q_i, \pi, n) = (q_i, \pi, n)$ by Definition 2.5. Further, $F^2(q_i, \pi, n) = F(F(q_i, \pi, n)) = F(q_i, \pi, n) = (q_i, \pi, n)$. The result now follows by induction on m .

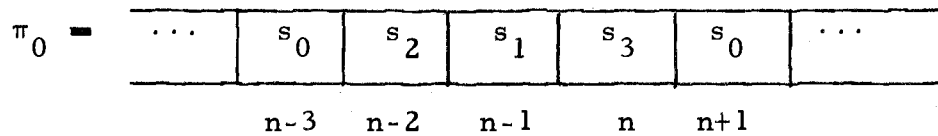
Theorem 2.1 leads us to make the following definitions.

Definition 2.6. If $F(q_i, \pi, n) = (q_i, \pi, n)$ then the Turing machine is said to have halted. The tape at the time of a halt is called the output tape.

Definition 2.7. The triple $\langle T, \pi A, F \rangle$ is called a Turing machine over the alphabet A.

To illustrate the above definitions, consider the Turing machine $Z_1 = \langle T, \pi A, F \rangle$ with alphabet $A = \{s_0, s_1, s_2, s_3\}$, Turing table T , input tape π_0 and initial configuration x_0 as shown below.

T	s_0	s_1	s_2	s_3
q_0	$q_2^h s_0$	$q_1^l s_1$	$q_1^l s_2$	$q_1^l s_3$
q_1	$q_2^h s_1$	$q_0^l s_1$	$q_0^l s_2$	$q_0^l s_3$
q_2	$q_2^h s_0$	$q_2^h s_1$	$q_2^h s_2$	$q_2^h s_3$



$$x_0 = (q_0, \pi, n).$$

The remaining squares of π_0 contain the symbol s_0 and the initial scanned symbol is s_3 in square n . Then applying F as determined by Definition 2.5, we have the following sequence of transformations of π_0 into an output tape:

$$\pi_1 = \pi_0$$

$$x_1 = (q_1, \pi_1, n-1) = F(x_0) \quad \text{since} \quad T(q_0, s_3) = q_1^l s_3;$$

$$\pi_2 = \pi_0$$

$$x_2 = (q_0, \pi_2, n-2) = F(x_1) \quad \text{since} \quad T(q_1, s_1) = q_0^l s_1;$$

$$\pi_3 = \pi_0$$

$$x_3 = (q_1, \pi_3, n-3) = F(x_2) \quad \text{since} \quad T(q_0, s_2) = q_1 s_2 ;$$

$$\pi_4 \equiv \begin{array}{|c|c|c|c|c|c|} \hline \cdots & s_1 & s_2 & s_1 & s_3 & s_0 & \cdots \\ \hline \end{array}$$

$$x_4 = (q_2, \pi_4, n-3) = F(x_3) \quad \text{since} \quad T(q_1, s_0) = q_2 h s_1 ;$$

$$\pi_5 = \pi_4$$

$$x_5 = x_4 = F(x_4) \quad \text{since} \quad T(q_2, s_1) = q_2 h s_1 .$$

If the Turing machine Z_1 is in state q_0 scanning the right most symbol of a non-empty finite sequence w of symbols of S on an otherwise blank tape, then Z_1 will halt after a finite number of steps over s_0 or s_1 depending upon whether the number of symbols in w is even or odd. This can be observed by examining the Turing table T . Z_1 passes to the left over the symbols of w alternating between states q_0 and q_1 searching for s_0 . It will reach s_0 in either state q_0 or q_1 . Depending on which, s_0 or s_1 will be marked on the tape and Z_1 will pass to state q_2 which is terminal for all s_j . Note that $T(q_2, s_2)$ and $T(q_2, s_3)$ need not be defined in order that Z_1 operate properly.

As a further example, consider the problem of evaluating the

function f (a variation of Ackerman's function [7: 82-88]) for given natural numbers m, n where

$$f(0, n) = n+1$$

$$f(m+1, 0) = f(m, 1)$$

$$f(m+1, n+1) = f(m, f(m+1, n))$$

and the natural number n is represented by $n+1$ strokes.

(m, n) is initially on an otherwise empty tape as shown below for the case $(3, 3)$. The machine is to replace (m, n) with the value of $f(m, n)$, halting over the blank square in front of $f(m, n)$.

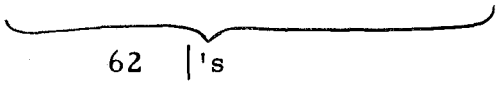
We proceed to describe a machine to compute the function f .

Let $Z = \langle T, \pi A, F \rangle$ with alphabet $A = \{s_0, (,), ', |, U\}$, Turing table T , input tape π_0 and initial configuration x_0 as shown below.

Some remarks about T are necessary. U is used as a dummy symbol in place of $|$. $T(q_1, |)$ is used to determine if $m = 0$; if not, Z changes into state q_7 which leads to an application of $T(q_9, |)$ if $n = 0$; otherwise Z changes into state q_{10} and proceeds to open the string of symbols on the tape in order to develop $f(m, f(m+1, n))$; then the entire strategy is repeated on $f(m+1, n)$ by means of $T(q_{16}, ($). The process continues very straightforward much as a hand calculation of f would. The remaining squares of π_0 contain the symbol s_0 and the initial scanned symbol is $($ in square $n-5$.

Applying F 230,021 times we transform π_0 into an output tape $\pi_{230,021}$ and terminal configuration $x_{230,021}$ as follows:

$$\pi_{230,021} \equiv \begin{array}{cccccccccccc} \cdots & s_0 & | & | & \cdots & | & | & | & s_0 & \cdots \\ \cdots & n-6 & n-5 & n-4 & \cdots & n+54 & n+55 & n+56 & n+57 & \cdots \end{array}$$



62 |'s

$$x_{230,021} \equiv (q_{16}, \pi_{230,021}, n-6)$$

One probably suspects that the determination of $\pi_{230,021}$ was not carried out by hand. When the author first began to program and study the behavior of Turing machines, it became evident that it is useful and necessary to have a reliable way of simulating the action

of Turing machines. The human as a simulator is prone to error after not too many applications of F in a Turing computation. Further the number of applications of F to transform an input tape to an output tape may be very large (e. g. in determining $f(3, 3)$, F was applied 230, 021 times).

The first description of a Turing machine simulator for a digital computer is due to Coffin, Goheen and Stahl [5]. It was designed for the Scientific Data Systems 920 Computer for the primary purpose of simulation of the operation of biological cell systems. A later effort was due to Brady [3] and made use of the International Business Machines 1620 Computer. The simulator of Brady was used in our study of Turing machines primarily because of availability of the IBM 1620 during the preparation of this paper. Brady's simulator was modified for our use because of differences in the memory size and the instruction set of the particular model of the 1620 that we were using.

The Composition of Turing Machines

It is very difficult to understand the behavior of a Turing machine from its Turing table, especially by an individual unfamiliar with the strategy of the designer. Therefore it becomes desirable to formulate complex Turing machines from more simple machines.

Our aim in this section is to describe a technique for the

composition of Turing machines. This technique will be developed later as an operator scheme. Some ideas of Hermes [7] are used, but conditional branching is greatly restricted. In fact, we allow branching only on the basis of the answer to the question: Is the final scanned symbol s_0 ?

Definition 2.8. Let X and Y be Turing machines with a common alphabet A and disjoint state sets Q_X and Q_Y respectively. Define a Turing machine XY as follows:

$$XY: (Q_X \cup Q_Y) \times A \rightarrow (Q_X \cup Q_Y) \times M \times A$$

where

$$XY(q_i, s_j) = \begin{cases} q_Y h s_0 & \text{if } j = 0 \text{ and } X(q_i, s_0) = q_i h s_0 \\ X(q_i, s_j) & \text{if } j = 0 \text{ and } X(q_i, s_0) \neq q_i h s_0 \\ X(q_i, s_j) & \text{if } j \neq 0 \text{ and } q_i \text{ in } Q_X \\ Y(q_i, s_j) & \text{if } q_i \text{ in } Q_Y \end{cases}$$

Intuitively, as a Turing table for XY from the above definition, we have

XY	A
Q_X	$X(Q_X \times A)$
Q_Y	$Y(Q_Y \times A)$

except that all entries of the form $X(q_i, s_0) = q_i h s_0$ are replaced with $q_Y h s_0$. The remaining entries remain unchanged. Thus XY operates as X would, but then follows Y if the final scanned symbol under X is s_0 ; otherwise a halt under X occurs in the usual manner.

Definition 2.9. Let X, Y and Z be Turing machines with common alphabet A and disjoint state sets Q_X, Q_Y and Q_Z , respectively. Define a Turing machine $T \equiv \overbrace{X Y Z}$ as follows:

$$T: (Q_X \cup Q_Y \cup Q_Z) \times A \rightarrow (Q_X \cup Q_Y \cup Q_Z) \times M \times A$$

where

$$T(q_i, s_j) = \begin{cases} q_Y h s_0 & \text{if } j = 0 \text{ and } X(q_i, s_j) = q_i h s_j \\ q_Z h s_j & \text{if } j \neq 0 \text{ and } X(q_i, s_j) = q_i h s_j \\ X(q_i, s_j) & \text{if } X(q_i, s_j) \neq q_i h s_j \\ Y(q_i, s_j) & \text{if } q_i \text{ in } Q_Y \\ Z(q_i, s_j) & \text{if } q_i \text{ in } Q_Z \end{cases}$$

Intuitively, as a Turing table for T from the above definition, we have

T	A
Q_X	$X(Q_X \times A)$
Q_Y	$Y(Q_Y \times A)$
Q_Z	$Z(Q_Z \times A)$

except that all entries of the form $X(q_i, s_j) = q_i \cdot h s_j$ are replaced with $q_Z \cdot h s_j$ provided $s_j \neq s_0$ and $X(q_i, s_0) = q_i \cdot h s_0$ is replaced with $q_Y \cdot h s_0$. The remaining entries are left unchanged. Thus T operates as X would, but then follows Y if the final scanned symbol under X is s_0 or Z if the final scanned symbol is not s_0 .

Definition 2.10. Let X, Y and Z be Turing machines with a common alphabet A and disjoint state sets Q_X, Q_Y and Q_Z , respectively. Define a Turing machine $T \equiv \overline{X} Y Z$ as follows:

$$T: (Q_X \cup Q_Y \cup Q_Z) \times A \rightarrow (Q_X \cup Q_Y \cup Q_Z) \times M \times A$$

$$T(q_i, s_j) = \begin{cases} q_Z h s_0 & \text{if } j = 0 \text{ and } X(q_i, s_j) = q_i h s_j \\ q_Y h s_j & \text{if } j \neq 0 \text{ and } X(q_i, s_j) = q_i h s_j \\ X(q_i, s_j) & \text{if } X(q_i, s_j) \neq q_i h s_j \\ Y(q_i, s_j) & \text{if } q_i \text{ in } Q_Y \\ Z(q_i, s_j) & \text{if } q_i \text{ in } Q_Z \end{cases}$$

Intuitively, Definition 2.10 serves to reverse the condition for a jump in Definition 2.9, i. e. the passage to Z occurs if the final scanned symbol under X is s_0 , otherwise Y is performed.

Definition 2.11. Let X be a Turing machine. Define a Turing machine $[X^n]$, $n \geq 1$ as follows:

$$[X^1] = X, \quad [X^2] = \overrightarrow{X X}, \quad [X^n] = \overrightarrow{X [X^{n-1}]}$$

Intuitively, Definition 2.11 yields a Turing machine $[X^n]$ which acts just as an n -fold application of X without regard to whether the terminal scanned symbol is s_0 or not. The output tape of X (if it halts) becomes the input tape of X in the next step.

Definition 2.11 can be extended to include $n = 0$ by considering $[X^0]$ to be the Turing machine with Turing table

$[X^0]$	s_0	s_1	\dots	s_j	\dots	s_n
q_0	q_0hs_0	q_0hs_1	\dots	q_0hs_j	\dots	q_0hs_n

$[X^0]$ halts on the initial scanned square and does not change state or symbols. $[X^0]$ serves as an identity machine.

Some Elementary Machines

In this section, we introduce three especially simple Turing machines from which more complicated machines will be later constructed. These machines can be constructed for any alphabet.

Definition 2.12. Let $A = \{s_0, s_1, \dots, s_n\}$ be an alphabet and $Q = \{q_0, q_1\}$ a state set. Define 3 Turing tables for Turing machines R, L, and P as follows:

i.

R	s_0	s_1	\dots	s_j	\dots	s_n
q_0	q_1rs_0	q_1rs_1	\dots	q_1hs_j	\dots	q_1rs_n
q_1	q_1hs_0	q_1hs_1	\dots	q_1hs_j	\dots	q_1hs_n

ii.

L	s_0	s_1	\dots	s_j	\dots	s_n
q_0	q_1ls_0	q_1ls_1	\dots	q_1ls_j	\dots	q_1ls_n
q_1	q_1hs_0	q_1hs_1	\dots	q_1hs_j	\dots	q_1hs_n

iii.	P	s_0	s_1	\dots	s_j	\dots	s_n
	q_0	q_1hs_1	q_1hs_2	\dots	q_1hs_{j+1}	\dots	q_1hs_0
	q_1	q_1hs_0	q_1hs_1	\dots	q_1hs_j	\dots	q_1hs_n

R is called the right machine. L is called the left machine. P is called the print machine.

R when initially scanning an arbitrary square of a Turing tape, moves one square to the right and halts. There is no change in the tape.

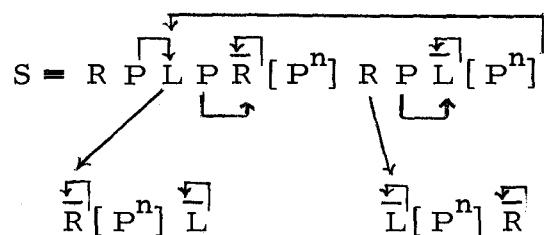
L when initially scanning an arbitrary square of a Turing tape, moves one square to the left and halts. There is no change in the tape.

P when initially scanning an arbitrary square of a Turing tape containing the symbol s_j replaces s_j with s_{j+1} modulo $n+1$ (s_n is replaced with s_0) and halts still scanning the initial square. There is no change in the remainder of the tape. This machine was suggested by an idea of Böhm [2] about a partially defined function in the set of tape configurations.

Note that $[P^{n-i+1}]$ has the effect of erasing a square (printing s_0) containing the symbol s_i .

Using the three machines R, L, P of Definition 2.12 and the composition techniques of Definitions 2.8-2.11, we can formulate a

Turing machine called the search machine as follows:



S searches out a square on an arbitrary tape that contains a symbol $s \neq s_0$, i.e. S finds a non-blank square if one exists. The search alternates to the right and then the left, back and forth until a square containing a symbol s different from s_0 is encountered at which time S halts scanning the symbol s .

An Operator Scheme for Composing Turing Machines

In this section we develop an operator scheme involving just the three elementary Turing machines R, L and P, in which we can compose any arbitrary Turing machine. This operator scheme relates in some ways to Lyapunov's operator programming method [9].

Definition 2.13. Let $E = \{R, L, P, \bar{R}, \bar{L}\}$, $E_1 = \{X_i^i \mid X \in E, i \in \mathbb{N}\}$, $E_2 = \{X^i \mid X \in E, i \in \mathbb{N}\}$, $E_3 = \{X_j^i \mid X \in E, i, j \in \mathbb{N}\}$. The set $O = E \cup E_1 \cup E_2 \cup E_3 \cup \{;\}$ is called the set of operators over E.

We interpret a lower subscript as the beginning of an arrow and the superscript as the end of an arrow as used in Definitions

2.8-2.11. Further a semicolon (;) appearing after a machine indicates that there is no transfer to the machine on its right as is normally indicated by juxtaposition (see Definition 2.8). It is clear that in view of Definition 2.13 and its interpretation, we can eliminate the arrows used in Definitions 2.9-2.11. Compositions of Turing machines become words over the infinite alphabet O of operators.

Definition 2.14. Let O be the set of operators as defined above. A finite word over O is called an operator description of a Turing machine if for every subscript i there is a unique appearance of a superscript i and for every superscript i there is at least one appearance of a subscript i .

For example the search machine S described earlier now appears as

$$S = R P_1 L_2^1 P_3 \bar{R}_3^3 [P^n] R_4 P_5 \bar{L}_5^5 [P^n]_1; \bar{R}_2^2 [P^n]_6 \bar{L}_6^6; \bar{L}_4^4 [P^n]_7 \bar{R}_7^7$$

It is evident from the definitions and examples above that the following theorem holds:

Theorem 2.2. Let $X = \langle T, \pi A, F \rangle$ be any Turing machine over an alphabet A . Then there is a word over O which is an operator description of X .

Proof: Let X be in the complete configuration (q_i, π, j)

and let $T(q_i, \pi(j)) = q_{ij} m_{ij} s_{ij}$ so that $F(q_i, \pi, j) = (q_{ij}, \bar{\pi}, \bar{j})$ where $\bar{\pi}(k) = \pi(k)$ if $k \neq j$, $\bar{\pi}(j) = s_{ij}$ and $\bar{j} = j+1$ if $m_{ij} = r$, $\bar{j} = j-1$ if $m_{ij} = \ell$, $\bar{j} = j$ if $m_{ij} = h$. Further let $|s_j| = j$. Then the portion of X involving the passing from (q_i, π, j) to $F(q_i, \pi, j)$ is given by

$$P_t^{(|s_{ij}| - |\pi(j)|) \bmod n+1} U_{m_{ij}}^t$$

where $U_r = R$, $U_\ell = L$, $U_h = [R^0]$. In this way X can be built as a word over O .

Turing Computability

Intuitively, a Turing machine calculates values of certain functions. For example, see page 11-13 (Z_1), where $f(w) = s_0$ or s_1 depending on whether the number of symbols is even or odd. We want to formalize these intuitive notions about computing functional values in this section.

We generally follow the definition of Turing computability given by Hermes [7] (called standard Turing computability by Hermes when treating recursive functions of natural numbers), although modifications in definitions and terminology are made.

Definition 2.14. A word w over the set of symbols S is

a finite sequence $s_{i_1} s_{i_2} \cdots s_{i_r}$ ($r \geq 1$) of symbols in S .

For example, if $S = \{s_1, s_2, s_3, s_4\}$ then $s_2 s_1 s_3$ and s_1 and $s_2 s_2 s_3$ are words over S .

Definition 2.15. Let $\Sigma'(S)$ be the set of all words over S . Let Λ be the empty word and let $\Sigma(S) = \Sigma'(S) \cup \{\Lambda\}$. $\Sigma(S)$ is the free semigroup under concatenation generated by S with identity Λ ($w\Lambda = \Lambda w = w$).

Since S is used throughout this paper for the set of symbols, it will be omitted so that Σ is written for $\Sigma(S)$.

It will be convenient later to exclude the use of the empty word for arguments and values of functions. We can do this without loss of generality in view of

Theorem 2.3. There exists a 1-1 mapping of Σ onto Σ' .

Proof: Let $f: \Sigma \rightarrow \Sigma'$ be defined by $f(\Lambda) = S_1; f(w) = s_1 w$ if w is of the form $w = s_1 s_1 \cdots s_1$; $f(w) = w$ otherwise. Clearly f has the properties required.

We proceed to define what we mean when we say a certain Turing machine computes the value of a function.

Let $S = \{s_1, s_2, s_3\}$. Let $w = s_2 s_1 s_3$ be a word in $\Sigma(S)$. We say w is on an otherwise blank Turing tape to mean, there is a Turing tape π such that $\pi(i) = s_2$, $\pi(i+1) = s_1$, $\pi(i+2) = s_3$,

$\pi(k) = s_0$ if $k \neq i, i+1, i+2$ for some i in I .

Definition 2.16. i. Let $f: \Sigma \rightarrow \Sigma$ be a function defined on Σ .

f is a function of a single variable. f is Turing computable if there exists a Turing machine X over an alphabet A with $S \subseteq A$ such that if w in Σ is on an otherwise blank tape and X is scanning a given square of the tape, then X will halt after finitely many configurations behind the word $f(w)$ in Σ so that the output tape has the form $\cdots s_0 w s_0 f(w) s_0 \cdots$ and is otherwise blank.

ii. Let $f: \underbrace{\Sigma \times \Sigma \times \cdots \times \Sigma}_{n \text{ factors}} \rightarrow \Sigma$ be a function defined

on $f: \underbrace{\Sigma \times \Sigma \times \cdots \times \Sigma}_{n \text{ factors}}$. f is a function of $n > 1$ variables. f is

Turing computable if there is a Turing machine X over an alpha-

bet A with $S \subseteq A$ such that if w_1, w_2, \cdots, w_n in Σ are on

an otherwise blank tape in the form $w_1 s_0 w_2 s_0 \cdots s_0 w_n$ and X

is scanning a given square of the tape, then X will halt after

finitely many configurations behind the word $f(w_1, w_2, \cdots, w_n)$ in

Σ so that the output tape has the form

$w_1 s_0 w_2 s_0 \cdots s_0 w_n s_0 f(w_1, w_2, \cdots, w_n)$ and is otherwise blank.

iii. Let $f: \Sigma^0 \rightarrow \Sigma$ be a function defined on the

empty class Σ^0 . f is a function of 0-variables with a single

value w (i.e. f is a constant) which may be any word in Σ .

f is Turing computable if there exists a Turing machine X over

an alphabet A with $S \subseteq A$ such that if X is initially scanning an arbitrary square of a blank tape, then X will halt after finitely many configurations behind w in Σ so that the output tape has w on it, but otherwise blank.

Before turning to the development of the recursive functions, we prove two theorems which will simplify our results on Turing computability. In view of these theorems, without loss of generality, we can assume our Turing machines are initially scanning the blank cell to the right of w_n where $w_1 s_0 w_2 s_0 \cdots s_0 w_n$ appears on an otherwise blank tape.

Theorem 2.4. There is a Turing machine X which when started operating, scanning an arbitrary square of a tape containing only $w_1 s_0 w_2 s_0 \cdots s_0 w_n$, will halt behind (to the right) the last symbol of w_n without altering the tape.

Proof: Let $X \equiv SR_1^1 R_1 L$. Clearly X is the desired machine. S finds a marked square; $R_1^1 R_1$ passes over $w_1 s_0 w_2 s_0 \cdots s_0 w_n$ ending on the second blank square to the right of w_n ; L moves back to the left one square to the first blank square behind w_n .

Theorem 2.5. Let X be a machine which computes $f(w_1, w_2, \dots, w_n)$ provided the initial scanned square is an arbitrary

square $j(w_1, w_2, \dots, w_n)$ defined in terms of w_1, w_2, \dots, w_n and let J be a Turing machine which locates j when it is started initially scanning the square behind $w_1 s_0 w_2 s_0 \dots s_0 w_n$. Then f is Turing computable by a machine Z which starts scanning initially an arbitrary cell.

Proof: Let $Z = S_1 R_1^1 R_1 L_2 J_3^2 X^3$. $S_1 R_1^1 R_1 L_2$ leads to the square behind $w_1 s_0 w_2 s_0 \dots s_0 w_n$ as indicated by Theorem 2.4; J_3^2 leads to the square $j(w_1, w_2, \dots, w_n)$ which X initially scans in order to compute $f(w_1, w_2, \dots, w_n)$.

CHAPTER III

RECURSIVE FUNCTIONS OVER AN ARBITRARY ALPHABET S
AND THEIR COMPUTABILITYDefinition of the Recursive Functions over S

We want to consider certain classes of functions over S (i. e. defined on Σ^0 or Σ^n) by generalizing previously stated definitions of recursive functions of natural numbers. There are several equivalent formulations available, but we prefer that of Kleene. In particular, we modify the concept of partial recursive function (general recursive if defined for all n -tuples) of natural numbers as formulated in the Corollary to Theorem XIX, Part III of Kleene [8] to allow the domain of definition to be words over the arbitrary set S of symbols, yet still agree with partial recursive functions of natural numbers (where S contains only the stroke symbol).

Definition 3.1. Let g_1, g_2, \dots, g_m be given functions of n variables over Σ and h a given function of m variables. Let f be given by

$$f(w_1, w_2, \dots, w_n) = h(g_1(w_1, w_2, \dots, w_n), \dots, g_m(w_1, w_2, \dots, w_n)).$$

Then f is said to be defined by the composition of g_1, g_2, \dots, g_m

with h_i .

Definition 3.2. Let g be a given function of n variables, h_i be a given function of $n+2$ variables. Let f be given by

$$f(w_1, w_2, \dots, w_n, \Lambda) = g(w_1, w_2, \dots, w_n)$$

$$f(w_1, w_2, \dots, w_n, ws_i) = h_i(w_1, w_2, \dots, w_n, w, f(w_1, w_2, \dots, w_n, w)).$$

Then f is said to be defined by primitive recursion from g and h_i .

Note that f depends on i , that is, there is an f for each s_i in S .

Definition 3.3. The initial functions are the following:

- i. $f_i(w) = ws_i \quad (i = 1, 2, \dots, n)$
- ii. $f_i(w_1, w_2, \dots, w_n) = w_i \quad (i = 1, 2, \dots, n)$
- iii. $f_n^w(w_1, w_2, \dots, w_n) = w$

The functions given in i are called the successor functions, those in ii are called projection functions, and those in iii are called constants. We allow $n = 0$ in iii so as to include functions of 0 variables with a constant value.

Definition 3.4. A function over S (i. e. with arguments in

Σ^0 or Σ^n and values in Σ) is a primitive recursive function if it can be generated from the initial functions by a finite number of applications of composition and primitive recursion.

Definition 3.4. Let g be a given function of $n+1$ variables. Let f be given by

$$f(w_1, w_2, \dots, w_n) = \mu_i w [g(w_1, w_2, \dots, w_n, w) = \Lambda] \quad (i = 1, 2, \dots, n)$$

where $\mu_i w [g(w_1, w_2, \dots, w_n, w) = \Lambda]$ means the shortest word $w \neq \Lambda$ of the form $w = s_i s_i \dots s_i$ such that $g(w_1, w_2, \dots, w_n, w) = \Lambda$ and $g(w_1, w_2, \dots, w_n, w')$ is defined and not Λ for all w' having the same form as w , but shorter than w . Then f is said to be defined by an application of the μ -operator.

We could define a μ -operator yielding the first word w in some fixed ordering of Σ which satisfied the condition $g(w_1, w_2, \dots, w_n, w) = \Lambda$. But this would require us to assign order to Σ . We prefer not to do this here.

Definition 3.5. A function over S (with arguments in Σ^0 or Σ^n and values in Σ) is a recursive function if it can be generated from the initial functions by a finite number of applications of composition, primitive recursion, and the μ -operator.

Further Turing Machines

Using the operator scheme developed in Chapter II, we now introduce some further Turing machines to be used in establishing the Turing computability of the recursive functions over S .

The first machine required is one to copy a word to the right of a given word (without altering it) leaving s_0 between the original word and its copy, assuming the tape is blank to the right of the original word.

Lemma 3.1. There exists a Turing machine K (copy machine) which copies a given word to the right of it with a blank square gap between the original word and its copy, when it is started initially scanning the square behind the word to be copied.

Proof: Let K be given by the following word over O :

$$\begin{aligned}
 K \equiv & L_1^1 R_2^4 R_3^3; P_5^2 [(R_6^6)^2] [P^n]_7 [(L_7^7)^2] [P^n]_4; \\
 & P_8^5 [(R_9^9)^2] [P^{n-1}]_{10} [(L_{10}^{10})^2] [P^{n-1}]_4; \\
 & \vdots \\
 & P^{3n-1} [(R_{3n+3}^{3n+3})^2] P_{3n+4} [(L_{3n+4}^{3n+4})^2] P_4.
 \end{aligned}$$

L_1^1 leads to the square in front of the given word w ; R_2^4 leads

to the machine scanning the first symbol s_i in w ; successive applications of $P_5^2, P_8^5, \dots, P_{3(n-i+1)+2}^{3(n-i+1)-1}$ change s_i to s_0 ; $[(R_{3(n-i+1)+3}^{3(n-i+1)+3})^2]$ leads to the second blank square to the right of s_i 's original position; $[P^{n-(n-i)}]$ prints s_i on this cell; $[(L_{3(n-i+1)+4}^{3(n-i+1)+4})^2]$ returns the machine to scanning the square where s_i was originally located (it presently contains s_0); $[P^{n-(n-i)}]$ replaces s_0 with s_i ; R_2^4 starts the cycle again on the next symbol to the right in w unless a blank symbol is reached in which case R_3^3 leads to a halt behind the copied word.

Another useful machine operates similarly to K , but copies the j^{th} word w_j in a string of words of the form $w_n s_0 w_{n-1} s_0 \dots s_0 w_j s_0 \dots s_0 w_1$ over to the right in the form $w_n s_0 w_{n-1} s_0 \dots s_0 w_j s_0 \dots s_0 w_1 s_0 w_j$ leaving the original string of words unaltered, assuming the tape is blank to the right of the original word.

Lemma 3.2. There exists a Turing machine K_j (j -copy machine) which copies the j^{th} word (from the right) in a string of words $w_n s_0 w_{n-1} s_0 \dots s_0 w_j s_0 \dots s_0 w_1$ to the right of the string with a blank square gap between w_1 and the copied word w_j .

Proof: Let K_j be given by the following word over O :

$$K_j^- = [(L_1^1)^j] R_2^4 [(R_3^3)^j]; \quad P_5^2 [(R_6^6)^{j+1}] [P^n]_7 [(L_7^7)^{j+1}] [P^n]_4;$$

$$P^{3n-1} [(R_{3n+3}^{3n+3})^{j+1}] P_{3n+4} [(L_{3n+4}^{3n+4})^{j+1}] P_4.$$

K_j^- works in a fashion analogous to the working of K , but the j^{th} word is copied instead of the first word encountered.

We use one further machine for computing the recursive functions. This machine is used for cleaning up intermediate calculations and bringing the result forward so that it appears in the correct position in agreement with the definition of Turing computability.

Lemma 3.3. Let a sequence of words w_1, w_2, \dots, w_n, w be on a tape in the form $s_0 s_0 w_1 s_0 w_2 s_0 \dots s_0 w_n s_0 w$ with the tape blank to the right of w . Then there exists a Turing machine C (clean up machine) which erases the words w_1, w_2, \dots, w_n and moves the word w so that its left most symbol is contained in the square in front of w_1 .

Proof: Let C be given by the following word over O :

$$C \equiv L_1^1 L_2 L_0 R \bar{R}^{2(n+2)} R_{2n+5}; R^0 P_2^2 R R_4^3 \bar{L}_1;$$

$$P_6^4 L[P^n] \bar{R}_3^5; P_8^6 L[P^{n-1}] \bar{R}_3^7; \dots; P^{2(n+1)} L P_{2n+3} \bar{R}_3^{2n+3};$$

$$P_{2n+7}^{2n+5} [L^2] [P^n]_{2(n+3)} \bar{R}^{2(n+3)}_{2(n+2)}; \dots; P^{4n+3} [L^2] P_{4(n+1)} \bar{R}^{4(n+1)}_{2(n+2)}.$$

$L_1^1 L_2$ leads to the right most symbol of the word to be erased.

$P_2^2 R R_4^3$ erases this symbol and moves to w in order to move

w one square to the left which is accomplished by

$$P_6^4 L[P^n] \bar{R}_3^5; \dots; P^{2(n+1)} L P_{2n+3} \bar{R}_3^{2n+3};$$

L_1^1 is reached when w has been moved one square to the left;

the cycle begins again with $L_1^1 L_2$; L_0 is reached when a given

word has been erased and checks for additional words to be erased;

if there are none, $R \bar{R}^{2(n+2)} R_{2n+5}$ leads to w in order to begin

the moving of w the remaining two blank squares which is accomplished by

$$P_{2n+7}^{2n+5} [L^2] [P^n]_{2(n+3)} \bar{R}^{2(n+3)}_{2(n+2)}; P^{4n+3} [L^2] P_{4(n+1)} \bar{R}^{4(n+1)}_{2(n+2)}$$

Note that intermediate computations are separated from other words to the left by at least two blank squares in order that one be able to recognize the beginning of these intermediate calculations.

With the three additional machines K , K_j , C we proceed

to prove our final results.

Computability of the Recursive Functions by Turing Machines

Using the operator scheme for computation with Turing machines as developed in Chapter II, we show in this section the computability of the recursive functions over an arbitrary alphabet S .

Theorem 3.4. If f is an initial function, there is a Turing machine X described by a word over O which computes f .

Proof: i. Let $f(w) = ws_i$. Then for X we take $X \equiv K[P_i]R$. K copies w over to the right; $[P_i]$ marks s_i to the right of the copy of w ; R moves behind $f(w)$ as required by Definition 2.16, i.

ii. Let $f(w_1, w_2, \dots, w_n) = w_i$. Then for X we take $X \equiv \overbrace{K}^{n-i+1}$. \overbrace{K}^{n-i+1} copies w_i over to the right of w_1, w_2, \dots, w_n and halts behind $f(w_1, \dots, w_n) = w_i$ as required by Definition 2.16, ii.

iii. Let $f_n^w(w_1, w_2, \dots, w_n) = w$ and let $w = s_{i_1} s_{i_2} \dots s_{i_r}$ and $|s_j| = j$. Then for X we take

$$X \equiv R[P \begin{array}{|c|} \hline s_{i_1} \\ \hline \end{array}]R[P \begin{array}{|c|} \hline s_{i_2} \\ \hline \end{array}]R \dots R[P \begin{array}{|c|} \hline s_{i_r} \\ \hline \end{array}]R$$

Clearly $[P \begin{matrix} |s_i| \\ |k| \end{matrix}]$ effects the marking of s_i . Hence X marks w on the tape, and in the terminal configuration is scanning the blank square behind w .

Note that we allow $n = 0$ in iii and that Λ is represented by s_1 (see Definition 3.3 and Theorem 2.3).

Theorem 3.5. If f is a primitive recursive function of $n \geq 0$ variables, then there is a Turing machine X described by a word over O which computes f .

Proof: i. See Theorem 3.4 for the computability of the initial functions.

ii. Let h, g_1, g_2, \dots, g_m be given as in Definition 3.1 and be computed by Turing machines H, G_1, G_2, \dots, G_m .

Let

$$f(w_1, w_2, \dots, w_n) = h(g_1(w_1, w_2, \dots, w_n), \dots, g_m(w_1, w_2, \dots, w_n)).$$

Then for X we can take

$$X = R_1 P^1 R_2 [(K_{n+1})^n]_3^2 [(L_4^4)^n]_5^3 [P^n]_6^5 R_6^6 R_6^6 L_7$$

$$(G_1)_8^7 [(K_{n+1})^n]_9^8 (G_2)_{10}^9 \dots [(K_{n+1})^n]_{2m+5}^{2m+4} (G_m)_{2m+6}^{2m+5}$$

$$(K_{m+(m-1)m})_{2m+7}^{2m+6} (K_{m+(m-2)m})_{2m+8}^{2m+7} \dots (K_m)_{3m+6}^{3m+5} H_{3m+7}^{3m+6} C_{3m+7}^{3m+7}$$

$R_1 P^1 R_2$ builds a bridge $s_0 s_1 s_0$ across which we copy w_1, w_2, \dots, w_n by means of $[K_{n+1}]_3^2$; $[(L_4^n)]^3 L_5 [P^n]_6^5$ removes the bridge producing a 3 blank square gap so that we may apply C at the end of the calculation to clean up intermediate results; $R_6^6 R_6 L_7$ leads to the cell behind the copied arguments; the calculation then proceeds with g_1, g_2, \dots, g_m , and finally h using g_1, g_2, \dots, g_m as arguments; C^{3m+7} erases the intermediate calculations halting while scanning the square behind $f(w_1, w_2, \dots, w_n)$ as required by Definition 2. 16.

iii. Let g and h_i be given as in Definition 3. 2 and be computed by Turing machines G and H_i . Let f be given by

$$f(w_1, w_2, \dots, w_n, \Lambda) = g(w_1, w_2, \dots, w_n)$$

$$f(w_1, w_2, \dots, w_n, w s_i) = h_i(w_1, w_2, \dots, w_n, w, f(w_1, w_2, \dots, w_n, w)) .$$

Then for X we can take

$$X = R_1 P^1 R_2 (K_2)_3^2 [(K_{n+3})^n]_4^3 [(L_5^{n+1})]_4^4 L_6 [P^n]_6^6$$

$$R_7^7 R_7 L_8 G_9^8 (K_{n+2})_9^9 L_{10}^{10} [P^{n-i+1}]_{11}^{11} L_{12}^{12} C^{24};$$

$$R_{14}^{13} [(K_{n+2})^n]_{14}^{14} R_{15}^{15} [P^i]_{16}^{15} R_{17}^{16} (K_{n+3})_{18}^{17} R_{19}^{18}$$

$$(H_i)_{19}^{19} (K_{n+4})_{20}^{20} L_{21}^{21} [P^{n-i+1}]_{22}^{21} L_{23}^{22} R_{24}^{23} [(K_{n+4})^{n+1}]_{25}^{24} R_{26}^{25} \bar{L}_{22}^{26} .$$

$R_1 P^1 R_2 (K_{23})^2 [(K_{n+3})^n]_4^3 [(L_5^{n+1})^4] L_6 [P^n]_6^7 R_7 R_7 L_8$ builds a bridge as in ii above; G_9^8 computes $g(w_1, w_2, \dots, w_n)$; $(K_{n+2})_9^{10} L_{11}^{10} [P^{n-i+1}]_{12}^{11} L_{13}^{12}$ copies w and erases the last symbol in the copy and moves left one square; if this square is blank, the calculation is complete and C^{24} cleans up intermediate calculations as before; if not, R_{14}^{13} takes us to the right one square to compute successive values of f using the primitive recursion process; $[(K_{n+2})^n]_{15}^{14} R_{16}^{15} [P^i]_{17}^{16} R_{18}^{17} (K_{n+3})_{19}^{18} (H_i)_{20}^{19}$ copies w_1, w_2, \dots, w_n , prints s_1 (representing Λ) behind them, copies $g(w_1, w_2, \dots, w_n)$ and computes h_i ; now w (with the last symbol erased) is copied and another symbol erased and the machine moves left one square by means of $(K_{n+4})_{21}^{20} L_{22}^{21} [P^{n-i+1}]_{23}^{22} L_{24}^{23}$; if this square is a blank, a branch to C^{24} occurs for clean up; if not we continue the primitive recursion process obtaining new arguments by $R_{25} [(K_{n+4})^{n+1}]_{16}^{25}$ and couple back to compute h_i again. In this way X computes f .

Theorem 3.6. If f is a recursive function of $n \geq 0$ variables, then there is a Turing machine X described by a word over O which computes f .

Proof: i. See Theorem 3.5 for the computability of the primitive recursive functions;

ii. Let g be given as in Definition 3.4 and let g

be computed by the Turing machine G . Let f be given by

$$f(w_1, w_2, \dots, w_n) = \mu_i w [g(w_1, w_2, \dots, w_n, w) = \Lambda]$$

where $w = s_i s_i \dots s_i$. Then for X , we can take the Turing machine

$$X \equiv ZR[P^i]R^2GL_1LC; P_1^1L_1[P^i]_2$$

where Z is given by

$$Z \equiv R_1P_1^1R_2[(K_{n+1})^n]_2^3[(L_4^4)^n]_3^4L_5[P^n]_5^6R_6^6R_6L_7.$$

Z builds a bridge $s_0s_1s_0$ over which w_1, w_2, \dots, w_n is copied and then removes the bridge producing a 3 blank square gap for use by C as described earlier in Theorem 3.5, ii; $R[P^i]R^2$ appends $w = s_i$ and leads to the blank square behind w ; GL_1 computes $g(w_1, w_2, \dots, w_n, w)$ and determines if $g = \Lambda$; if yes, LC cleans up the result producing $f(w_1, w_2, \dots, w_n) = w$; if no, $P_1^1L_1$ erases $g(w_1, w_2, \dots, w_n)$ leading to the square behind w ; $[P^i]_2$ appends another s_i to w and branches to the calculation of g again. Thus X computes f .

Theorems 3.4 - 3.6 establish the computability of the recursive functions over an arbitrary alphabet S by an operator scheme involving basically just three elementary Turing machines (R, L, P)

and a simple unary predicate, testing the terminal scanned symbol for s_0 or not.

CHAPTER IV

CONCLUSIONS

In this final section, we briefly discuss the implications of some of the results established in the previous parts of this paper and suggest a new problem for further study.

Turing's thesis [8] is that "every function which would naturally be regarded as computable (e. g. recursive functions) is computable by one of his machines". In all papers (known to the author) where evidence is given in support of this thesis, only functions with natural numbers for arguments and values are considered. From results of Chapters II and III, Turing machines apply equally well to functions of words in any language having a finite set of symbols.

Turing's thesis might be restated as "all algorithms can be formulated in terms of certain matrices (Turing tables) and executed by the corresponding Turing machines." This does not imply that all problems should be reduced to their equivalent Turing machines. However, the fact that any Turing machine can be described using the set of operators (essentially TM's R, L, P) in a relatively easy way suggests that certain problems may be profitably reduced to a Turing machine formulation [5]. Also in this connection, the operator programming of Turing machines might prove useful in teaching the

fundamentals of programming.

As far as we know the use of the presence or absence of s_0 (blank symbol) as the final scanned symbol in describing the succession of operators in our scheme is new.

For further study, one might consider the problem of using the operator scheme developed in this paper to describe Turing machines for computing e or π in the sense that these machines print the expansion of non-terminating decimals on the tape. This process is not the same as that described in Chapter III. Further the means taken to represent a real number will determine very much the outcome.

BIBLIOGRAPHY

1. Anderson, S. E. Some computational schemes equivalent to Turing machines. Master's thesis. Corvallis, Oregon State University, 1964. 31 numb. leaves.
2. Bohm, C. and G. Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Communications of the Association for Computing Machinery* 9(5): 366-371. 1966.
3. Brady, A.H. Solutions of restricted cases of the halting problem applied to the determination of particular values of a non-computable function. Ph. D. thesis. Corvallis, Oregon State University, 1965. 107 numb. leaves.
4. Church, A. An unsolvable problem of elementary number theory. *American Journal of Mathematics* 58: 345-363. 1936.
5. 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.
6. Davis, M. *Computability and unsolvability*. New York, McGraw-Hill, 1958. 210 p.
7. Hermes, H. *Enumerability, decidability, computability*. New York, Springer-Verlag, 1965. 245 p.
8. Kleene, S. C. *Introduction to metamathematics*. Princeton, Van Nostrand, 1952. 550 p.
9. Lyapunov, A. A. On logical schemes of programs. In: *Problems of cybernetics*, ed. by A. A. Lyapunov. Vol 1. Moscow, USSR, State Publishing House for Physics-Mathematical Literature, 1958. p. 46-74. (In Russian)
10. Post, E. L. Finite combinatory processes - formulation I. *Journal of Symbolic Logic* 1: 103-105. 1936.
11. Post, E. L. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic* 12: 1-11. 1947.

12. Turing, A. M. On computable numbers with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, ser. 2, 42: 230-264. 1936; 43: 544-546. 1937.