AN INFORMAL ARITHMETICAL APPROACH TO COMPUTABILITY AND COMPUTATION

Z. A. Melzak

(received May 15, 1961)

1. In 1936 A. M. Turing published his analysis of the notion of effective computability. Very roughly speaking, its object was to distinguish between numbers defined by existential statements and those defined by purely constructive ones. Guided by his schematizing of what goes on when a person calculates using paper and pencil, Turing introduced the concept of an A-machine, eventually to be called a Turing machine. Such a machine consists of a box and a linear tape divided into squares, indefinitely long in both directions. The tape passes through the box and exactly one square of it is always within the box. The system changes only at certain discrete times: $t = 0, 1, \ldots$. Each square can carry exactly one of the finite set of symbols: $s_1, s_2, \ldots, s_n$, one of which may be the blank. The box can be in one of the finite number of states: $q_1, q_2, \ldots, q_m$. At any time t the situation is specified by the current state q_a of the box and the symbol s_b on the currently scanned square. Under these circumstances three events happen, in this order: 1) s_b is changed to s_B ($b = B$ is allowed), 2) the box shifts itself by one square to the right or to the left, or it remains stationary, and 3) q_a is changed to q_A ($a = A$ is allowed). The machine is completely specified by: 1) the list of symbols and states, 2) the symbols on the tape, the state of the box and the square being scanned, all at time $t = 0$, and 3) the list of correspondences between the pairs (q_a, s_b) and the triples (q_A, s_B, ψ), where ψ can assume the values 0, 1 and 2, referring to the motion to the right, to the left, and rest;

Canad. Math. Bull., vol. 4, no. 3, September 1961

the totality of these is called the description D_A of the Turing machine A.

If the symbols include the digits 0, 1, ..., 9, then it may happen that with a suitable description D_A the machine eventually prints on its tape the decimal development of a real number x; x is then called computable. Turing has proved that there are countably many computable numbers, including all the real algebraic numbers as well as π, e, the zeros of certain transcendental functions, etc. He also advanced the thesis that a number is effectively calculable if and only if it is computable. That is to say, he assigned the precise meaning 'computable' to such phrases as 'effectively calculable' or 'possessing a well-defined finite algorithm'. Turing's great achievement was to show that a universal machine can be constructed; this is an A-machine which, on being presented with a suitably coded description D_{A_1} of any particular A-machine A_1 on its input tape, proceeds to duplicate the computation of A_1.

Actually, there are several minor complications which have been omitted in the above sketch. For instance, there is an initial and a final state - these correspond to the orders 'start' and 'stop'. Also, the tape does not have to be unbounded in both directions, it suffices to have it extend indefinitely far to the right and to make some provision against the possibility of an overshoot to the left. Further, in Turing's original description the number computed was being printed on the even-numbered squares and the intermediate calculations were carried out on the odd-numbered ones.

2. In the intervening twenty-five years it became clear that most automatic computers are equivalent to a universal Turing machine, if provided with an indefinitely large storage. In fact, it became one of the favourite occupations of the personnels of the computation laboratories to simulate in the slack hours universal Turing machines on their own computers under various restrictive conditions.

In relation to an actual automatic computer, it might be

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

said that a Turing machine represents the computer-program, while a universal Turing machine acts as the computer itself. Alternatively, it might be said that a Turing machine represents a law of digit-formation, while a universal Turing machine represents a way of generating the actual digits according to this law.

Certain features of Turing machines have induced later workers to propose alternative devices as embodiments of what is to be meant by effective computability. We shall consider briefly some of these features. In the first place, there is the indefinitely long tape. Then, a Turing machine has a certain opacity, its workings are known rather than seen. Further, a Turing machine is inflexible - it computes a single number only, and to remedy this it is necessary either to change it internally or to pass over to a universal machine. Also, a Turing machine is slow in (hypothetical) operation and, usually, complicated. This makes it rather hard to design it, and even harder to investigate such matters as time or storage optimization or a comparison between efficiency of two algorithms.

The alternative devices that have been proposed, include the finite automata (equivalent, aside from the questions of time, hardware and economics, to the actual switching circuits), the B-machines of H. Wang (these are, roughly, A-machines with two symbols, blank and mark, and an attached program using four instructions: move to the right, to the left, mark, and transfer conditionally, but the erasure of marks is not allowed), the W-machines of C. Y. Lee (which are B-machines augmented by the ability to erase), and so on.

It might be remarked that A-machines as well as most alternatives, are primarily oriented toward the logical task of symbol manipulation rather than toward the arithmetical task of calculating. For instance, the state-symbol transition table of a Turing machine is a sort of a time-dependent truth-table.

3. It is our object to describe a primitive device, to be called a Q-machine, which arrives at effective computability via arithmetic rather than via logic. Its three operations are keeping tally, comparing non-negative integers, and transferring.

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

The Q-machine is equal in computing power to a universal Turing machine, and it is completely transparent. First and foremost, however, it is so simple that its workings could probably be understood by an average school-child after a few minutes' explanation. Another feature is its similarity in the matters of programming, subroutines, etc., to the actual computers. As against these, and some other, advantages, it has some rather serious drawbacks. These matters will be considered later on.

4. Consider the following specific task: to calculate $2^{\frac{1}{2}}$. It is easily proved that if p_0 and q_0 are two arbitrary positive integers and

$$p_{n+1} = p_n + 2q_n, \quad q_{n+1} = p_n + q_n, \quad n = 0, 1, \ldots$$

then the sequence $\{p_n/q_n\}$ tends to $2^{\frac{1}{2}}$. This suggests the following simple way of calculating $2^{\frac{1}{2}}$. Imagine four locations S, A, B, C, for definiteness, say, four holes in the ground. An indefinitely large number of counters, for definiteness, say, pebbles, is present in S, while A and B contain initially p_0 and q_0 counters respectively and C is empty. Let XYZ denote the operation of taking from the location X as many counters as there are in Y, and transferring them to Z. In particular, XXY denotes the operation of transferring the contents of X into Y. Consider now the following sequence of four operations:

(1) SBA, BBC, SAB, CCA.

As this is being carried out, the contents of the locations A, B, C, change as follows:

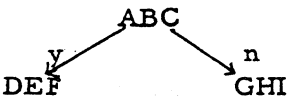
initial	p_0	q_0	0
after SBA	$p_0 + q_0$	q_0	0
after BBC	$p_0 + q_0$	0	q_0
after SAB	$p_0 + q_0$	$p_0 + q_0$	q_0
after CCA	$p_0 + 2q_0$	$p_0 + q_0$	0

Hence iterating the sequence (1) results in accumulating in A and B the numerators and denominators of the sequence $\{p_n/q_n\}$.

Most of the scheme about to be proposed will be clear by now, but it is necessary to introduce one further modification.

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

As might be guessed, it allows the operation known in programming terminology as the conditional transfer. Suppose that the contents of the locations X, Y, Z are p, q, r counters respectively. Then the operation XYZ can be carried out if and only if $p \geq q$. In particular, the operations SYZ and YYZ can always be carried out. This suggests introducing an array of operations



and interpreting it as follows: if ABC can be carried out then carry it out and proceed next to DEF, if ABC cannot be carried out leave the locations A, B, C as they were and proceed next to GHI.

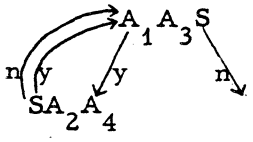
5. The Q-machine consists of an indefinitely large number of locations: S, A_1, A_2, ..., an indefinitely large supply of counters distributed among these locations, a program, and an operator whose sole purpose is to carry out the program. Initially all but a finite number from among the locations A_1, A_2, ... are empty and each of the remaining ones contains a finite number of counters. A sequence of three locations XYZ is called a command if: 1) $Y \neq S$, 2) $(X, Y, Z) \neq (A, A, A)$, and 3) $X \neq Z$. A program is a finite sequence of commands of which one, say C_1, is called the first one, and in which every command C_i has two arrows leading from it to two commands C_j and C_k; one arrow is marked 'y' and the other one 'n'. It is possible to have $i = j$, $i = k$ or $j = k$, but not $i = j = k$. An exception to these rules is an arrow which does not terminate on any command; this is to be interpreted as an order: 'stop'. A command $C = XYZ$ is carried out by taking from the location X as many counters as there are in the location Y and transferring them to the location Z, if this can be done, and proceeding then to the command along the arrow from C marked 'y'; if C cannot be carried out the locations X, Y and Z are left intact and the next command to be carried out is at the end of the arrow from C marked 'n'. A program is carried out by beginning with the first command and then proceeding as indicated.

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

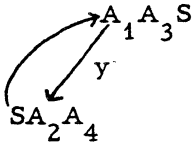
A complete description of the Q-machine is given by indicating the initial contents of the locations A_1, A_2, ..., and submitting the program. We shall occasionally not distinguish between the location A_i and the number of counters which it (currently) contains.

6. We give now some examples of programs.

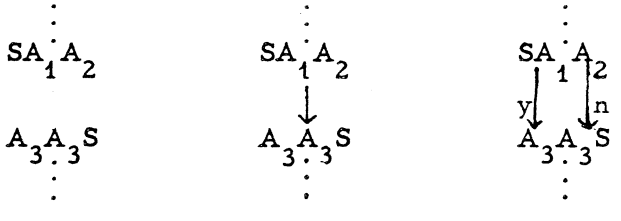
A) Multiplication.
Initial contents: p, q, 1, 0, 0, ...
Program:



Final contents: 0, q, 1, pq, 0, ...
To simplify the writing of programs we shall write the above program as



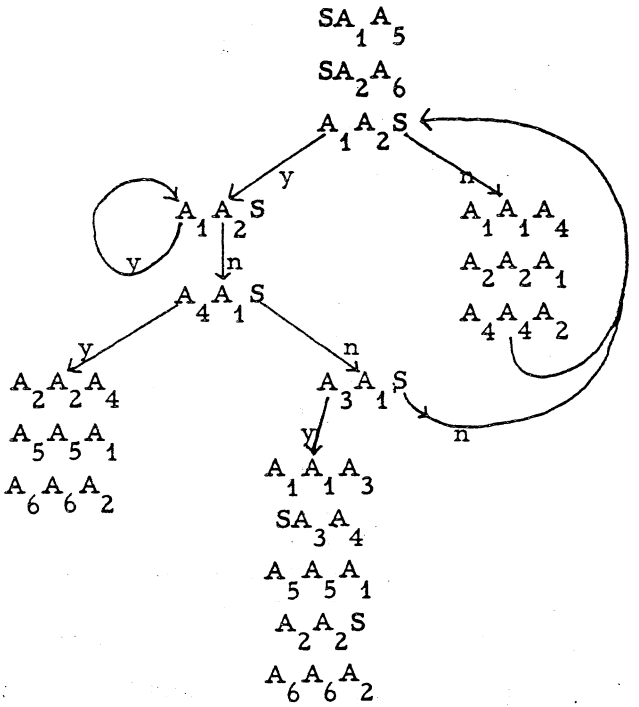
That is to say, we omit the terminal arrow (or arrows) leading to 'stop' and we replace two arrows marked by 'y' and 'n' and leading from one command to another, by a single unmarked arrow. We shall also omit arrows descending straight down so that, for instance, the following are equivalent:



B) The Euclidean Algorithm.
Initial contents: p, q, 1, 0, 0, ... , $p > 0$, $q > 0$.

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

Program:



Final contents: p, q, 1, g.c.d.(p, q), 0, 0, ...

C) Calculating the n-th. prime π_n.

To simplify our work we introduce first an analogue of subroutines. Consider the following scheme:

E = E(p, q, 1, 0, 0, 0; p, q, 1, g.c.d.(p, q), 0, 0).

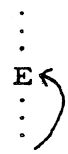
This is an abbreviation for the program of B) with the initial contents shown in front of the semicolon and the final contents after it. The two zeros after the semicolon and the three in front indicate that the locations A_1 - A_6 must be kept free for the program. E may now be incorporated into other programs. For instance, given two integers p and q, $p > q > 0$, g.c.d.(p+q, p-q) may be calculated as follows:

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

Initial contents: p, q, 1, 0, 0, ...
Program:

$$SA_1A_7$$
$$SA_2A_8$$
$$A_2A_2A_1$$
$$SA_7A_4$$
$$A_4A_8S$$
$$A_4A_4A_2$$
$$E$$
$$A_1A_1S$$
$$A_2A_2S$$
$$A_7A_7A_1$$
$$A_8A_8A_2$$

Final contents: p, q, 1, g.c.d.(p+q, p-q), 0, 0, ...
Here it is to be understood that the arrow from E to the next command A_1A_1S (not shown in the above program) represents all the terminal arrows issuing out of E and leading (before) to the order 'stop'. Also, if a part of the program looked like this

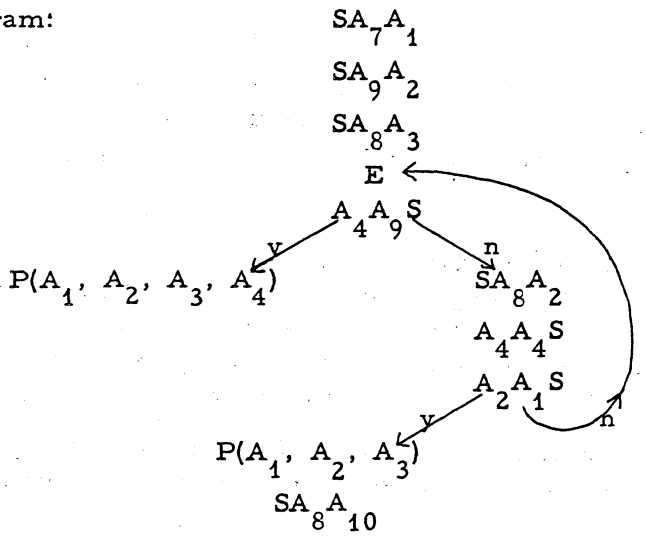


then the arrow would be understood to lead to the first command of E. The same conventions will apply throughout. We shall occasionally use the abbreviation $P(A_{i_1}, \ldots, A_{i_k})$ for the program which empties the indicated locations into S.

It will be convenient to set up first an intermediate program for checking whether a given integer $m \geq 3$ is a prime or not.
Initial contents: 0, 0, 0, 0, 0, 0, m, 1, 2, 0, ...
(The first six zeros show the locations reserved for executing the Euclidean Algorithm program B))

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

Program:



Final contents: 0, 0, 0, 0, 0, 0, m, 1, 2, a_m, 0, ...

where $a_m = 0$ if m composite and $a_m = 1$ if m prime.

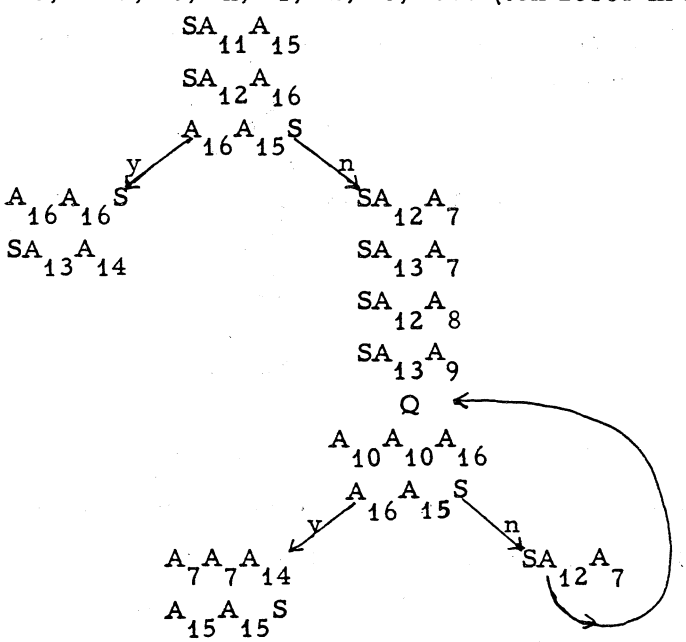
We shall denote this program by

$Q = Q(0, 0, 0, 0, 0, 0, m, 1, 2, 0; 0, 0, 0, 0, 0, 0, m, 1, 2, a_m)$

Now the n-th prime π_n can be calculated by the following program.

Initial contents: 0, ..., 0, n, 1, 2, 0, ... (ten zeros in front).

Program:



https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

Final contents: 0, ..., 0, n, 1, 2, π_n, 0, ... (ten zeros in front)

It must be added that the above programs are very wasteful of space and time, for instance, a number n is tested for primality by being divided successively by 2, 3, 4, ..., n-1.

7. We shall introduce one final modification; this will allow the machine to change its own program in the process of carrying it out. Our modification consists essentially of introducing what is known in computer terminology as 'a computed 'go to' instruction'. Consider a command $A_i A_j A_k$, its effect is to operate on certain locations which are fixed in advance, namely i, j, k. We shall allow commands with iterated locations, e.g.

$$A_{A_1} A_{A_2} A_{A_3};$$

this command is carried out by taking from the location whose number is the contents of the location A_1 as many counters as there are in the location whose number is the contents of the location A_2, and transferring them to the location whose number is the contents of the location A_3 (if this can be done, etc.).

A simple analysis shows that single iteration of subscripts suffices, for instance

$$A_{A_{A_1}} A_{A_{A_2}} A_{A_{A_3}}$$

can be replaced by the sequence

$$SA_{A_1} A_i$$

$$SA_{A_2} A_j$$

$$SA_{A_3} A_k$$

$$A_{A_i} A_{A_j} A_{A_k},$$

where A_i, A_j and A_k are any three available locations.

Similarly, any command of the type

$$A_{A \cdots A_i} A_{A \cdots A_j} A_{A \cdots A_k}$$

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

can be replaced by a sequence of commands of the type

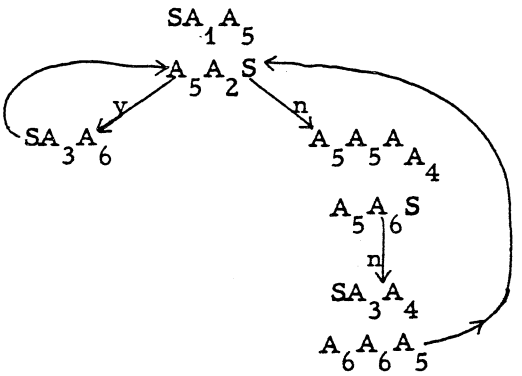
$$A A_i A A_j A A_k .$$

As a simple application, we consider two programs.

D) Decimal representation in the scale of d.
Let p and d be positive integers, $d \geq 2$. We want to find the unique coefficients in the representation

$$p = \Sigma_{n=0}^{N} p_n d^n, \quad 0 \leq p_n < d.$$

Initial contents: p, d, 1, 7, 0, 0, 0, ...
Program:



Final contents: p, d, 1, N+7, 0, 0, p_0, p_1, ..., p_N, 0, ...

E) Sorting by magnitude.
Let $p_1, \ldots, p_n$ be n non-negative integers, $n \geq 2$, and let

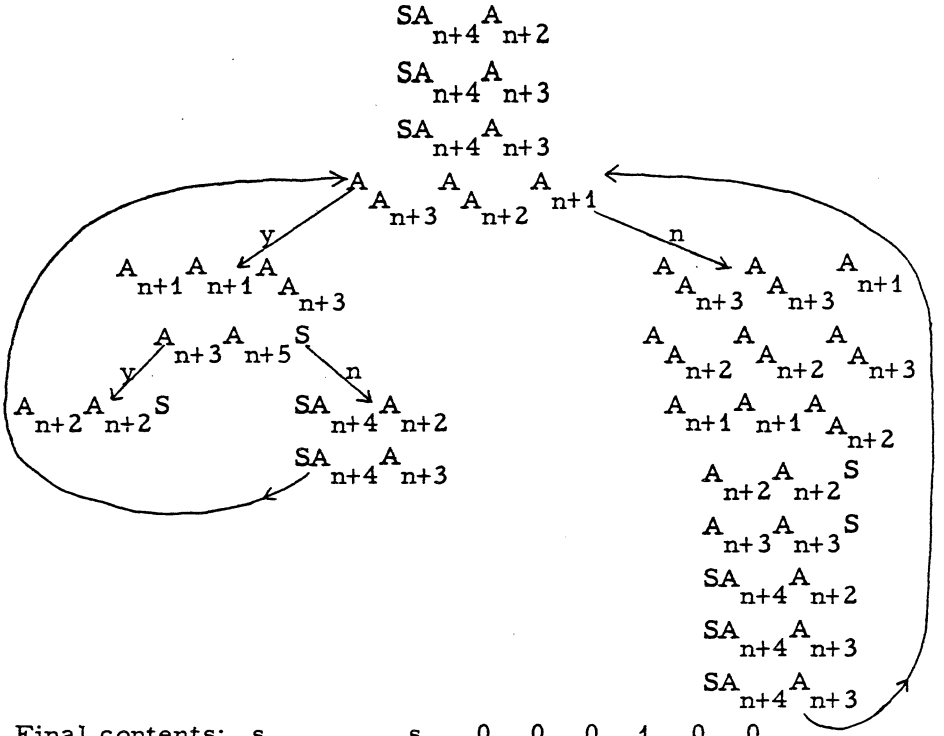
$$\{p_1, \ldots, p_n\} = \{s_1, \ldots, s_n\}$$

where $s_1 \leq s_2 \leq \ldots \leq s_n$.

Initial contents: $p_1, \ldots, p_n$, 0, 0, 0, 1, n, 0, ...

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

Program:



Final contents: $s_1, \ldots, s_n, 0, 0, 0, 1, 0, 0, \ldots$

8. We show now that the Q-machine, with a suitable program, can calculate any number which is computable in the sense of Turing. While this could be done in several direct and indirect ways, we shall show instead that an arbitrary Turing machine can be simulated on the Q-machine. We remark first that instead of a single row of locations $A_1, A_2, \ldots$ one may consider any finite number of rows of locations A_{ij}, $i = 1, 2, \ldots k$, $j = 1, 2, \ldots$ This can be arranged by letting A_n correspond to A_{pq}, where $n = qk + (p-1)$, $1 \leq p \leq k$. Since the Euclidean Algorithm can be carried out on the Q-machine, the above arrangement is likewise executable. It is also possible to arrange countably many rows of locations A_{ij}, $i, j = 1, 2, \ldots$ by letting A_n correspond to A_{ij}, where i and j are uniquely given by the equation $n = j + \{(i + j - 1)(i + j - 2)\}/2$. Conversely, a Q-machine with a finite or a countably infinite

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

column of location rows is equivalent to the Q-machine with a single row of locations. This corresponds to the well-known proposition that a Turing machine with a finite number of tapes or with a plane tape (an infinite square lattice) is equivalent to the ordinary one.

We suppose that the Turing machine to be simulated has the tape symbols $s_1, \ldots, s_N$ and the states $q_1, \ldots, q_M$. We shall indicate these by their subscripts alone. Consider now the Q-machine with nine rows:

		function
1	$A_1, A_2, \ldots$	the tape
2	$B_1, B_2, \ldots$	
3	$C_1, C_2, \ldots$	state and tape indicator
4	$D_1, D_2, \ldots$	list of symbol-state combinations
5	$E_1, E_2, \ldots$	
6	$F_1, F_2, \ldots$	list of symbol-state-motion combinations
7	$G_1, G_2, \ldots$	
8	$H_1, H_2, \ldots$	
9	$I_1, I_2, \ldots$	program execution

Here the first two rows simulate the squares of the tape in this order: $\ldots, B_2, B_1, A_1, A_2, \ldots$ and A_1 represents the square inside the box. In our simulation the 'tape' will move and the 'box' will remain stationary. In the third row C_1 indicates the current state of the box while C_2 and C_3 give the last occupied squares in the first and second row respectively. That is to say, C_2 and C_3 indicate precisely how much of the 'tape' is occupied at each instant. The fourth and fifth rows contain the list of all distinct symbol-state pairs in the form $(D_1, E_1), \ldots, (D_P, E_P)$, and the sixth, seventh and eighth rows contain the corresponding list of symbol-state-motion triples in the form $(F_1, G_1, H_1), \ldots, (F_P, G_P, H_P)$. Each location H_i contains 0, 1 or 2 counters, corresponding to the motion to the right, rest and the motion to the left. The

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

rows 4 - 8 contain the complete transition list for the Turing machine. If the current symbol (= the contents of A_1) is i and if the current state of the machine (= the contents of C_1) is j, then there is exactly one index k, such that $D_k = i$ and $E_k = j$. The current symbol is then changed to F_k, the state is changed to G_k and the tape is moved one square to the right if $H_k = 0$, it remains stationary if $H_k = 1$ and it is moved one square to the left if $H_k = 2$. The motion 'one square to the right' means that each A_i is transferred to A_{i+1}, B_1 is transferred to A_1 and each B_i, $i \geq 2$, is transferred to B_{i-1}. Since the locations C_2 and C_3 show the extent of the occupied part of the rows simulating the tape, the above operation simulating the motion of the tape can be carried out. The motion 'one square to the left' is similarly arranged. When this (i.e., the motion) has been done, the locations C_1, C_2 and C_3 are changed accordingly, and the whole operation is carried out again. The initial contents of the locations of the first two rows and of C_1, C_2 and C_3 give the description of the initial situation. The details of constructing the simulation program are clear now and the program itself is left to the reader as an exercise.

By using the elements of the theory of recursive functions it can also be easily shown that every number calculable on the Q-machine is computable.

9. The chief advantage claimed for the Q-machine is its simplicity. Indeed, its operation could probably be explained without any words by using merely a few universally understandable ideograms. On the other hand, the Q-machine is fairly versatile so that the size and complexity of its programs compare not too badly with those for the actual computers. Further, programming for the Q-machine has many features in common with the standard programming. For these reasons the Q-machine might be useful as a teaching tool, especially at the more

https://doi.org/10.4153/CMB-1961-031-9 Published online by Cambridge University Press

elementary level.

However, the Ω-machine has some serious drawbacks. Its program, which is what is known as a labelled directed graph, is a much more complex entity than its Turing counterpart. Then, instead of a single unbounded tape as in Turing machines, one deals here with an unbounded number of locations, each one of which may contain an unbounded number of counters. Also, there appears to be no easy way to arrange for a stored program. Further, though this is not necessarily bad, there is no separation between the storage and the arithmetic parts of the machine. Still, these shortcomings would be serious only if the Ω-machine were used as an actual computer, and they need not rule it out as a teaching aid and a source of amusement.

10. The author acknowledges gratefully the benefit of conversations with Drs. R. Hamming, D. McIlroy and V. Vyssotsky of the Bell Telephone Laboratories and with Dr. H. Wang of Oxford University, and also the help of the Canadian Mathematical Congress in the form of a Fellowship at the 1961 Mathematics Summer Research Institute.

University of British Columbia