AN INFORMAL ARITHMETICAL APPROACH TO COMPUTABILITY AND COMPUTATION, III

Z.A. Melzak[*]

(received April 14, 1966)

17. It is assumed that the reader is acquainted with the first two parts of the present paper, [1] and [2], in which there was developed informally the theory of a certain class of hypothetical computing devices, the Q-machines. In the present part of the paper we develop a way of describing Q-programs and Q-computations; then, following the theorem in [2], we obtain some further connections between the arithmetical nature of a number x and the computational schemas for generating sequences of rational numbers converging to x; and finally, by means of the usual device of Goedel numbering we begin to examine some properties of programs which operate on programs.

18. Consider first a simple program P of Q_2^0. Any such program can be prefaced by the sequence $(A_0 A_1)^{p_1}$ $(A_0 A_2)^{p_2} \dots (A_0 A_n)^{p_n}$ which results in placing p_i counters in the location A_i for $i = 1, \dots, n$. It may be assumed therefore that all locations are initially empty. Let P consist of c commands $I_1, \dots, I_c$, exclusive of the command 'stop' which will be designated by I_0. We suppose that $I_i = A_j A_k$; further, I_1 is the initial command, and the halting command I_0 is written as $A_0 A_0$. The location numbers j and k are functions of i so that $I_i = A_{F(i)} A_{S(i)}$, $i = 0, 1, \dots, c$, and $F(0) = S(0) = 0$. If

[*] Present address: Courant Institute of Mathematical Sciences, N.Y., U.S.A.

$$\mathrm{Max}\,(F(1),\ S(1),\ \ldots,\ F(c),\ S(c)) = L$$

then no location beyond A_L is used in P. However, by a suitable re-enumeration of locations it may be assumed that $L \leq 2c$. Once the number c and the functions $F(i)$ and $S(i)$ are given, the commands of P are known. To show how these commands are linked together into the program P we introduce two further functions $Y(i)$ and $N(i)$: $Y(i)$, resp. $N(i)$, is the number of the command at the end of the 'y' arrow, resp. the 'n' arrow, leading from the i-th command I_i. $Y(i)$ and $N(i)$ are defined for $1 \leq i \leq c$ and satisfy the inequalities $0 \leq Y(i) \leq c$, $0 \leq N(i) \leq c$.

The four functions $Y(i)$, $N(i)$, $F(i)$ and $S(i)$ describe P completely. Previously imposed conditions require only that $F(i) \neq S(i)$ except for the special convention $F(0) = S(0) = 0$, and that $Y(i)$ and $N(i)$ be not both equal to i. We require further that an arrow is to lead to every command, except possibly the first one, so that for every i, $i \neq 1$, there is j such that $Y(j) = i$ or $N(j) = i$.

To simplify further description we convert Q_2^0 into a synchronous machine. That is, the commands are supposed to be executed only at times $t = 1, 2, \ldots$; the first command I_1 is executed instantaneously at the time $t = 1$, the next one instantaneously at the time $t = 2$ and so on, and nothing happens in between those times. Let $\varphi(t)$ be the number of the command carried out at the time t, so that $\varphi(1) = 1$. Let also $f_k(t)$ be the contents of the k-th location at the time $t - 1/2$, or, what is the same thing, just before the execution of the t-th successive command $I_{\varphi(t)}$. We have now the initial and boundary conditions

$$f_k(1) = 0 \text{ for } k \geq 1,\ f_0(t) = \infty \text{ for all } t,\ \varphi(1) = 1. \tag{1}$$

Further, $0 \leq \varphi(t) \leq c$ for all t for which φ is defined. If T is the earliest time such that $\varphi(T) = 0$ the program terminates at that time and the computation stops, $\varphi(t)$ is then not defined for any $t > T$. If the program is not terminating then $\varphi(t)$ is defined for all t and we have $\varphi(t) \geq 1$. A simple example of such a program is

$$\begin{tikzcd} A_0\,A_1 \arrow[loop left, "y"] \arrow[dr, "n"] & \\ & A_0\,A_0 \end{tikzcd} ,$$

here all the requirements are satisfied and $c = 1$, $F(1) = 0$, $S(1) = 1$, $Y(1) = 1$, $N(1) = 0$, $f_1(t) = t - 1$, $f_k(t) = 0$ for $k \geq 2$, and $\varphi(t) = 1$ for all t.

The functions $\varphi(t)$, $f_1(t)$, ..., $f_L(t)$ are called the state functions of the program and our next object is to derive the equations satisfied by them; these will be called the state equations. At the time t the program executes the command $I_{\varphi(t)} = A_{F(\varphi(t))} A_{S(\varphi(t))}$. Therefore the next command $I_{\varphi(t+1)}$ to be executed will be either $I_{Y(\varphi(t))}$ or $I_{N(\varphi(t))}$ depending on whether the contents $f_{F(\varphi(t))}(t)$ of the location $A_{F(\varphi(t))}$, just prior to the time t, is positive or 0. This gives us the state equation for $\varphi(t)$:

(2) $$\varphi(t+1) = Y(\varphi(t)) \operatorname{sgn} f_{F(\varphi(t))}(t) + N(\varphi(t))[1 - \operatorname{sgn} f_{F(\varphi(t))}(t)].$$

Next, observe that $f_j(t+1)$ can be different from $f_j(t)$ only if the command $I_{\varphi(t)}$ carried out at the time t involves the location A_j, or equivalently, if either $F(\varphi(t)) = j$ or $S(\varphi(t)) = j$. In the first case $I_{\varphi(t)} = A_j A_i$ and so $f_j(t+1) = f_j(t) - 1$ if $f_j(t) > 0$ and $f_j(t+1) = f_j(t)$ if $f_j(t) = 0$. To sum up,

(3) $$f_j(t+1) = f_j(t) - \operatorname{sgn} f_j(t) \qquad \text{if } F(\varphi(t)) = j.$$

Similarly

(4) $$f_j(t+1) = f_j(t) + \operatorname{sgn} f_{F(\varphi(t))}(t) \qquad \text{if } S(\varphi(t)) = j.$$

Combininb (3) and (4) leads to the state equation for $f_j(t)$:

(5) $$f_j(t+1) = f_j(t) - [1 - \operatorname{sgn}|F(\varphi(t)) - j|] \operatorname{sgn} f_j(t)$$
$$+ [1 - \operatorname{sgn}|S(\varphi(t)) - j|] \operatorname{sgn} f_{F(\varphi(t))}(t).$$

State equations (2) and (5) together with the initial and boundary conditions (1) describe completely the whole course of the computation of P. The reader acquainted with such matters may recognize the similarity of the state equations (2) and (5) to the sequential Boolean equations of circuit-synthesis.

Consider next the case of a Q_2^0 program P which may be subscripted. There are again c commands $I_1, \ldots, I_c$ together with the stop I_0, and the same conventions will apply to I_1 and I_0 as before. The command I_i is now not necessarily simple and we have $I_i = A_{x(i)} A_{y(i)}$ where the symbols $x(i)$ and $y(i)$ are either numbers or numbered locations. To distinguish between those possibilities we introduce two further functions $a(i)$ and $b(i)$, $0 \leq i \leq c$, as follows: $a(i) = 0$ if $x(i)$ is a number and $a(i) = 1$ if $x(i)$ is a numbered location; $b(i)$ is similarly defined with respect to $y(i)$. The functions $F(i)$ and $S(i)$ are as before, but with the above proviso. There are now four possibilities for a command:

$$\text{(6)}\quad \begin{aligned} I_i &= A_{F(i)} A_{S(i)} && \text{if } a(i) = 0 \text{ and } b(i) = 0, \\ I_i &= A_{A_{F(i)}} A_{S(i)} && \text{if } a(i) = 1 \text{ and } b(i) = 0, \\ I_i &= A_{F(i)} A_{A_{S(i)}} && \text{if } a(i) = 0 \text{ and } b(i) = 1, \\ I_i &= A_{A_{F(i)}} A_{A_{S(i)}} && \text{if } a(i) = 1 \text{ and } b(i) = 1. \end{aligned}$$

The special convention regarding the stop I_0 requires that $a(0) = b(0) = 0$ and, as before, $F(0) = S(0) = 0$. The functions $Y(i)$, $N(i)$, $\varphi(t)$ and $f_j(t)$ are exactly as before. Observing (6) and proceeding as in the derivation of (2) we get the following state equation for $\varphi(t)$:

$$\text{(7)}\quad \begin{aligned} \varphi(t+1) = {} & Y(\varphi(t)) \{[1 - a(\varphi(t))] \operatorname{sgn} f_{F(\varphi(t))}(t) \\ & + a(\varphi(t)) \operatorname{sgn} f_{f_{F(\varphi(t))}(t)}(t) \} \\ & + N(\varphi(t)) \{[1 - a(\varphi(t))][1 - \operatorname{sgn} f_{F(\varphi(t))}(t)] \\ & + a(\varphi(t)) [1 - \operatorname{sgn} f_{f_{F(\varphi(t))}(t)}(t)]\} . \end{aligned}$$

The state equation for f_j is now rather lengthy because of the number of ways in which the location A_j may be involved in the command $I_{\varphi(t)}$. We have

$$\begin{aligned}(8)\qquad f_j(t+1) = f_j(t) &+ [1 - a(\varphi(t))] \{ \operatorname{sgn} f_{F(\varphi(t))}(t) \\ &- \operatorname{sgn} f_j(t) [1 - \operatorname{sgn} |F(\varphi(t)) - j|]\} \\ &+ a(\varphi(t)) \operatorname{sgn} f_{f_{F(\varphi(t))}(t)}(t) \operatorname{sgn} |f_{F(\varphi(t))}(t) - j| \\ &- \{[1 - b(\varphi(t))] \operatorname{sgn} |S(\varphi(t)) - j| + b(\varphi(t)) \operatorname{sgn} |f_{S(\varphi(t))}(t) - j|\} \cdot \\ &\cdot \{[1 - a(\varphi(t))] \operatorname{sgn} f_{F(\varphi(t))}(t) + a(\varphi(t)) \operatorname{sgn} f_{f_{F(\varphi(t))}(t)}(t)\} .\end{aligned}$$

As a check we verify that (7) and (8) reduce to (2) and (5) when a and b are set equal to 0. State equations similar to (2) and (5) for the case of simple programs, and to (7) and (8) for the case of subscripted programs, can be written down in the same way as above for any Q-machine in terms of its transfer functions. With such machines as Q_3^0 or Q_5^0 which cannot start with empty locations it is necessary to replace the first equation in (1) by a suitable set of initial conditions describing the initial contents.

19. In this section we prove some results connecting the arithmetical nature of a number x and the type of program which computes it. We limit ourselves to real numbers x which are $\bar{Q}_5^0$ computable in the following way: x is the limit of a sequence p_n/q_n, $n = 0, 1, \ldots,$ of rational numbers, p_n and q_n $(q_n \neq 0)$ are integers which can be computed by a non-terminating cyclic simple $\bar{Q}_5^0$ program without internal loops: initial contents: $p_0, q_0, r_0, \ldots, s_0, 0, 0, \ldots$

P: $$\underline{\rightarrow I_1 \; I_2 \; \ldots \; I_c \rceil}$$

in which each command is either a pure addition or a pure multiplication. The above program is improper since there is no 'stop'. Negative integers are handled in the manner described in section

16 of [2]. One cannot speak here of final contents, but it is assumed that the contents after n cycles are p_n, q_n, r_n, ..., s_n, 0, 0, ...

To put it briefly, we are interested in numbers computable iteratively by means of a fixed number of additions and multiplications per iteration stage. Let the c commands consist of a additions and $m = c - a$ multiplications; we call such a program one of the type (a, m).

THEOREM 1. Let x_1 and x_2 be two irrational numbers computable by the programs of the type (a_1, m_1) and (a_2, m_2) respectively, and let m_1 and m_2 have their lowest possible values. If $m_1 \neq m_2$ then $x_1 x_2$ and x_1 / x_2 are irrational.

For if x_1 / x_2, say, is a rational number p/q and $x_2 = \lim p_n/q_n$, then $x_1 = \lim (p\, p_n)/(q\, q_n)$. Therefore, if x_2 is computable by a program of the type (A, m_2), x_1 is computable by a program of the type (B, m_2), where $B \leq A + p + q$. The theorem in section 16 of [2] implies at once

THEOREM 2. An irrational number computable by a program of the type $(a, 0)$ is algebraic. If x is a real irrational algebraic number which exceeds in absolute value all of its conjugates, and if A, B, C, D are any rational integers, then the number $(Ax + B) / (Cx + D)$ is computable by a program of the type $(a, 0)$.

This raises the question whether there exists a fixed bound M, such that every algebraic number is computable by a program of the type (a, m), with $m \leq M$.

A program of the type we are considering here will be said to be of span k if the number of sequences p_n, q_n, ... appearing in it is k. Since we are clearly interested only in irrational numbers the span of any program is at least 2: there are at least two sequences present, one for the numerators and one for the denominators of the approximating sequence.

THEOREM 3. An irrational number b is algebraic if and only if it is computable by a program of span 2.

To prove it in one direction it suffices to show that the Newton method can be programmed as a program of span 2. Let $F(x) = 0$ be an irreducible equation of which b is a root. Then an open interval J can be found such that if p_0 / q_0 is in J and

$$(9) \qquad p_{n+1} / q_{n+1} = p_n/q_n - F(p_n/q_n)/F'(p_n/q_n)$$

then p_n / q_n converges to b. Define recursively

$$(10) \qquad \begin{aligned} p_{n+1} &= q_n^d [p_n F'(p_n / q_n) - q_n F(p_n/q_n)] \\ q_{n+1} &= q_n^{d+1} F'(p_n / q_n), \end{aligned}$$

where p_0 and q_0 are any given integers subject to the condition $p_0 / q_0 \in J$, and d is the degree of the polynomial F. It follows from (10) that (9) holds; further,

$$p_{n+1} = P(p_n, q_n), \qquad q_{n+1} = Q(p_n, q_n)$$

where P and Q are fixed polynomials with integer coefficients. Therefore p_{n+1} and q_{n+1} are obtainable from p_n and q_n by one cycle of a program of span 2.

Suppose next that the real irrational number b can be computed by a program of span 2. One has then $b = \lim p_n / q_n$ where

$$p_{n+1} = G(p_n, q_n), \qquad q_{n+1} = H(p_n, q_n).$$

Since each command executes an addition or a multiplication, G and H are polynomials with integer coefficients. Therefore

$$G(p_n, q_n) = \sum_{j=0}^{N} P_j(p_n, q_n), \qquad H(p_n, q_n) = \sum_{j=0}^{M} Q_j(p_n, q_n);$$

here P_j and Q_j are homogeneous polynomials, with integer coefficients, of degree j. Hence

$$p_{n+1} / q_{n+1} = q_n^{N-M}$$

(11)
$$\left[\sum_{j=0}^{N} q_n^{-(N-j)} P_j(p_n/q_n, 1) \Big/ \sum_{j=0}^{M} q_n^{-(M-j)} Q_j(p_n/q_n, 1) \right].$$

It may be assumed that $\epsilon > 0$ and n_0 can be found, such that

$$|P_N(p_n / q_n, 1)| \geq \epsilon, \qquad |Q_M(p_n / q_n, 1)| \geq \epsilon \qquad \text{for } n \geq n_0,$$

for otherwise the limit $b = \lim p_n / q_n$ satisfies the polynomial equation $P_N(x, 1) = 0$ or $Q_M(x, 1) = 0$ and hence must be algebraic. Since b is irrational q_n tends to infinity with n; it follows now that $N = M$ and by (11) b is a root of the equation

$$x = P_N(x, 1) / Q_N(x, 1).$$

Therefore b is algebraic and the proof is complete.

It may be added that if the algebraic number b is of degree n and height H (= maximum of the absolute values of the coefficients in the minimal polynomial) and if b is computed by a program of the type (a, m) then it is possible to give upper bounds on the minimal values of a and m in terms of n and H. Theorem 3 is best possible of its kind since a program of span 3 can already compute a transcendental number. It is known [3, pp. 34-35] that the number

$$\xi = \sum_{0}^{\infty} 2^{-2^n}$$

is transcendental. Let $p_0 = 1$, $q_0 = 2$, $r_0 = 2$,

$$p_{n+1} / q_{n+1} = p_n/q_n + 2^{-2^{n+1}},$$

and consider the following program:

initial contents: p_0, q_0, r_0, 0, 0, ...

```
┌→03304
│ 33600
│ 44630
│ 02304
│ 01305
│ 11600
│ 22610
│ 55610
│ 44620
└─────┘
```

contents after n cycles: p_n, q_n, r_n, 0, 0, ... where $r_n = 2^{2^n}$.

Here we use the abbreviation 03304 for $A_0\, A_3\, A_3\, A_0\, A_4$. The above program computes ξ and is of span 3. In similar fashion it is easy to supply programs of span 4 for π and e.

We show next the existence of a large class of $\bar{Q}_5^0$ computable numbers. Let $P(x, y, \ldots)$ be a polynomial with integer coefficients in any finite number of variables; a function of the type

$$f(n) = P(n,\ 2^n,\ 3^n,\ \ldots,\ 2^{2^n},\ 2^{3^n},\ \ldots,\ 3^{2^n},\ \ldots)$$

will be called a hyperexponential polynomial, in analogy to polynomials and exponential polynomials.

THEOREM 4. Let $f_1(n)$ and $f_2(n)$ be two hyperexponential polynomials. Suppose that $f_2(n) \neq 0$ for $n = 0, 1, \ldots$ and that the series

$$\xi = \sum_0^\infty f_1(n) / f_2(n)$$

converges. Then ξ is $\bar{Q}_5^0$ computable.

Instead of sequences p_n, q_n, ... we shall now use sequences p_{in}, $i = 1, \ldots, s$, $n = 0, 1, \ldots$. Let $p_{30} = p_{3n} = 1$, $p_{40} = 0$, $p_{4\,n+1} = p_{4n} + p_{3n}$ so that $p_{4n} = n$. Since $p_{4n} = n$, by means of a finite number of additions and multiplications we can generate all the necessary powers of n entering into $f_1(n)$

https://doi.org/10.4153/CMB-1966-073-6 Published online by Cambridge University Press

and $f_2(n)$. Powers like 2^n can be generated by letting $p_{50} = p_{5n} = 2$, $p_{60} = 1$, $p_{6\ n+1} = p_{6n}\, p_{5n}$, so that $p_{6n} = 2^n$. Iterated powers like 4^{3^n} can be generated by letting $p_{90} = 4$, $p_{9\ n+1} = p_{9n}^3$ so that $p_{9n} = 4^{3^n}$. It is now a mere matter of detail to show that for some finite s a suitable $\bar{Q}_5^0$ program computes ξ.

A natural question arises here of exhibiting numbers which are computable but not $\bar{Q}_5^0$ computable. In this connection we notice that only double exponentials occur in the hyperexponential polynomials of Theorem 4, and we are led to conjecture that in general the occurence of triple exponentials will prevent a number from being $\bar{Q}_5^0$ computable. In particular, we conjecture that the number

$$\eta = \sum_0^\infty 2^{-2^{2^n}}$$

is not $\bar{Q}_5^0$ computable.

Define the span of a real irrational $\bar{Q}_5^0$ computable number to be the smallest span of a program which computes it. From Theorem 3 it follows that all algebraic numbers are of span 2 and all $\bar{Q}_5^0$ computable transcendental numbers are of span ≥ 3. This raises several further questions. Do there exist transcendental numbers of arbitrarily high span ? Of any given span s for $s \geq 3$? Are the spans of π and e 3 or 4 ? What is the span of Euler's constant γ ? If the span of b is s and $f(x)$ is some such function as e^x or $\log x$, what is the span of $f(b)$? Are two numbers of different spans necessarily algebraically independent? The answers to these, and other similar questions, do not appear to be easy. It is possible that some of them might throw a certain amount of light on the nature and properties of the transcendental numbers. A partial information on the last question is provided by the following.

https://doi.org/10.4153/CMB-1966-073-6 Published online by Cambridge University Press

THEOREM 5. Let a_1 and a_2 be two real irrational $\bar{Q}_5^0$ computable numbers of spans s_1 and s_2 respectively. If $|s_1 - s_2| > 2$ then a_1 and a_2 are algebraically independent.

Suppose that $s_2 > s_1 + 2$ and that a_1 and a_2 are algebraically dependent. Therefore a_2 is a root of a polynomial equation $P(x) = 0$ where

(12) $$P(x) = \sum_{j=0}^{N} P_j(a_1)\, x^j ;$$

here P_j are polynomials with integer coefficients. We can now use the Newton method to find a_2: if u_0 and v_0 are suitable integers and

$$u_{n+1} / v_{n+1} = u_n / v_n - P(u_n / v_n) / P'(u_n / v_n)$$

then u_n / v_n tends to the limit a_2. However, the number a_1 and consequently the coefficients of the polynomial P have also to be determined approximatively. By the hypothesis, there is a program of span s_1 with the sequences p_n, q_n, ..., r_n such that $p_n / q_n \to a_1$. We therefore let

(13) $$u_{n+1} / v_{n+1} = u_n / v_n - \sum_{j=0}^{N} P_j(p_n/q_n)(u_n/v_n)^j \Big/ \sum_{j=1}^{N} j\, P_j(p_n/q_n)(u_n/v_n)^{j-1} .$$

A simple continuity argument shows that if u_0 and v_0 are appropriate and if p_0 / q_0 is close enough to a_1, then the iterative Newton scheme (13) leads to $u_n / v_n \to a_2$. Let d_j = degree P_j, $d = \max_j d_j$, then (13) may be written as

$$u_{n+1} / v_{n+1} = u_n / v_n -$$

$$\sum_{j=0}^{N} q_n^d P_j(p_n/q_n) u_n^j v_n^{N-j} \Big/ \sum_{j=1}^{N} j q_n^d P_j(p_n/q_n) u_n^{j-1} v_n^{N-j+1} .$$

Therefore we define

$$u_{n+1} = u_n \sum_{j=1}^{N} j q_n^d P_j(p_n/q_n) u_n^{j-1} v_n^{N-j+1}$$

$$- v_n \sum_{j=0}^{N} q_n^d P_j(p_n/q_n) u_n^j v_n^{N-j} ,$$

$$v_{n+1} = v_n \sum_{j=1}^{N} j q_n^d P_j(p_n/q_n) u_n^{j-1} v_n^{N-j+1}$$

which is of the form

$$u_{n+1} = P(p_n, q_n, u_n, v_n), \quad v_{n+1} = Q(p_n, q_n, u_n, v_n)$$

where P and Q are fixed polynomials with integer coefficients. We now adjoin the two sequences u_n and v_n to the s_1 sequences $p_n, q_n, \ldots, r_n$ and we find that the limit $a_2 = \lim u_n / v_n$ is $\bar{Q}_5^0$ computable by a program of span $s_1 + 2$. Therefore $s_2 \leq s_1 + 2$ which is a contradiction. This completes the proof.

COROLLARY. If a_1 is $\bar{Q}_5^0$ computable and is of span s_1, and if a_1 and a_2 are algebraically dependent, then a_2 is also $\bar{Q}_5^0$ computable (and of span $\leq s_1 + 2$).

In the preceding we have associated the type (a, m) and the span s with $\bar{Q}_5^0$ programs; another such number which can be so associated is the total number w of locations used throughout the program; this will be called the weight of the program. The weight of a $\bar{Q}_5^0$ computable number is the smallest weight of a program which computes it. We have now

THEOREM 6. Every algebraic number is of weight ≤ 7.

To prove it we merely check that the Newton procedure can be programmed as a $\bar{Q}_5^0$ program in such a way that only the first seven locations need ever be occupied.

With respect to the weight of a number we can ask now some questions similar to those for its span. In addition, we have the following analogue of Theorem 5:

THEOREM 7. Let a_1 and a_2 be two real irrational $\bar{Q}_5^0$ computable numbers of weights w_1 and w_2 respectively. If $|w_1 - w_2| > 7$ then a_1 and a_2 are algebraically independent.

The proof is similar to that of Theorem 5 and will be omitted.

20. Up to now the Q-machines were operating on numbers, computing thereby new numbers. In this section we shall be interested in some questions where a Q-machine operates reflexly: here a program is to operate on a class of programs. Examples of such questions are a) the halting problem: is there a program which decides whether any given program terminates? b) the boundedness problem: is there a program which computes the total number of locations used in any given program? c) the occupation problem: is there a program which decides whether any given program will use some arbitrary prescribed location? d) the simplification problem: is there a program which decides whether the computation of any given program can be carried out by a simple program? Unlike the first problem, the last three are trivial for the special case of simple programs.

We limit ourselves to Q_2^0 programs in which all locations are initially empty. The programs which are to act on such programs are necessarily subscripted and must have some input: the initial contents will be a single number describing the program which is to be acted upon. Let P be any Q_2^0 program of size c, that is, with c commands, exclusive of the stop I_0. P is completely known once we are given the six functions F(i), S(i), Y(i), N(i), a(i), b(i), $1 \leq i \leq c$. By re-enumerating the locations if needed, it may be assumed that $0 \leq F(i) \leq 2c$ and $0 \leq S(i) \leq 2c$. We have also $0 \leq Y(i) \leq c$ and $0 \leq N(i) \leq c$; a(i) and b(i) assume only the values 0 or 1. It follows that aside from re-

https://doi.org/10.4153/CMB-1966-073-6 Published online by Cambridge University Press

enumerations the number $N(c)$ of programs of size c satisfies

$$N(c) \leq [2(2c+1)(c+1)]^{2c} .$$

Further, let p_j be the j-th consecutive odd prime starting with 3, and put

$$(14) \qquad G(P) = 2^c \prod_{i=1}^{c} p_{6i}^{F(i)} p_{6i+1}^{S(i)} p_{6i+2}^{Y(i)} p_{6i+3}^{N(i)} p_{6i+4}^{a(i)} p_{6i+5}^{b(i)} .$$

$G(P)$ is called the (Goedel) number of P and it determines P uniquely. By means of this device we shall be able to transform questions about programs into questions about numbers. Recalling the bounds on the various exponents in (14) we find that except for the re-enumeration of locations the number of a program of size c does not exceed

$$(15) \qquad M(c) = 2^c \, p_{6c+5}^{(6c+2)c} .$$

Let $g(n)$ be a function defined for $n \geq 0$ and assuming non-negative integer values. We shall say that g is computable if there is a terminating program P with the initial contents $n, 0, 0, \ldots$ and the final contents $g(n), 0, 0, \ldots$. Let $g_1(n) = 0$ if n is not a number of a program and $g_1(n) = 1$ if n is such a number. The question 'is $g_1(n)$ computable?' is now the precise form of the question 'can Q_2^0 recognize those integers which are numbers of programs?' . The answer is yes:

THEOREM 8. $g_1(n)$ is computable.

We suppose that there are eight rows of locations A-H. The initial contents of A_1 is n and all other locations are empty. The row B serves as the arithmetical space; a suitable subroutine using prime numbers 'decodes' n according to (14) (with $G(P)$ replaced by n) and stores the values of $F(1), \ldots, F(c)$ in $C_1, \ldots, C_c$. Similarly, the values of the function $S(i)$ are stored in $D_1, \ldots, D_c$, and so on. Once the values of the six functions $F(i)$ - $b(i)$ are stored in the tabular form as location contents of the rows C-H it is a simple matter to program the 'check' whether the conventions referring to these functions are satisfied, and to arrange for the corresponding output 0 or 1 in A_1.

We shall call a program $\hat{P}$ universal if it does the following: the initial contents of A_1 is n, the final contents of A_1 is $g_1(n)$ and if $g_1(n) = 1$, so that n is the number of a program P, then $\hat{P}$ duplicates the computation of P and results in the same final contents (of $A_2, A_3, \ldots$) as P would have done (for the contents of $A_1, A_2, \ldots$). We require $\hat{P}$ to be terminating if P is terminating. The existence of $\hat{P}$ bears some similarity to the existence of a universal Turing machine.

THEOREM 9. There exists a universal program $\hat{P}$.

We start as before, except that we have now nine rows of locations A-I. We compute first $g_1(n)$. If $g_1(n) = 0$ the program stops (except for possible 'erasing'). If $g_1(n) = 1$ we suppose that the six functions $F(i)$ - $b(i)$ are stored in the rows C-H as before. The row I serves for storing the contents $f_i(t)$ (these are the state functions defined in section 18). In other words, the row I simulates with its locations $I_1, I_2, \ldots$ the usual row $A_1, A_2, \ldots$. Using the state equations (7) and (8) we now proceed to duplicate the computation of the program whose number is n. To assure that the state functions $\varphi(t)$ and $f_j(t)$ can be computed without complications, we arrange for an index which keeps at all times track of the location I_i with the largest i, which is non-empty. The details are left to the reader.

Define $g_2(n)$ as follows: $g_2(n) = 0$ if n is the number of a terminating program, and $g_2(n) = 1$ otherwise. The halting problem is now reduced to the question whether $g_2(n)$ is computable. From the theory of Turing machines it is known that the answer is no; here we shall produce an independent proof. Let $f(c)$ be the largest number which can be computed by a terminating program of size c. That is, there is a terminating program of size c which starts with all locations empty and leads to the final contents $f(c)$ in some location; moreover, $f(c)$ is the largest number with that property. Since aside from re-enumerations there is only a finite number of programs of size c, as shown before, there is also a finite number of terminating programs of size c, and therefore $f(c)$ is uniquely defined and finite for each c. Since clearly $f(c+1) \geq 1 + f(c)$, $f(c)$ is strictly increasing.

LEMMA 1. $f(c)$ is not computable.

Suppose that this is false. Then there is a terminating program P with the initial contents $c, 0, 0, \ldots$ and final contents $f(c), 0, 0, \ldots$. Let the size of P be g. Define $f^{-1}(c) = \min \{ k: f(k) \geq c \}$; it follows that there is a program P_0 of size $f^{-1}(c)$ which starts with all locations empty and leads to the final contents $\geq c$ in, say, A_1. Now juxtapose P_0 and P; we obtain then a program of size $g + f^{-1}(c)$ which leads to the final contents $\geq f(c)$, therefore

$$f(c) \leq f(g + f^{-1}(c)). \tag{16}$$

However, it is easy to show that $f(c)$ grows faster than c (or indeed faster than any computable function of c) so that $f^{-1}(c) = o(c)$ for large c, and this contradicts (16).

THEOREM 10. $g_2(n)$ is not computable.

We prove this by showing that the computability of $g_2(n)$ implies the computability of $f(c)$, thus contradicting Lemma 1. Suppose then that there is a terminating program P_{halt} with the initial contents $n, 0, 0, \ldots$ and final contents $g_2(n), 0, 0, \ldots$. We can now compute $f(c)$ as follows. As remarked before, the number of a program of size c is $\leq M(c)$ except for possible re-enumeration of locations; here $M(c)$ is given by (15). Let n vary over the set C of integers from 2^c to $M(c)$ which are divisible by 2^c and by no higher power of 2. For each n we decide first whether it is the number of a program (i.e., whether $g_1(n) = 1$), by Theorem 8 this can be done. If n is not the number of a program we pass on to the next value $n+1$, if n is the number of a program P, we apply P_{halt} to find out whether P is terminating. If it is, we use the universal program $\hat{P}$ of Theorem 9 to simulate P, and we check at each stage of the simulation what is the largest number appearing in the locations. In this way we get the largest number appearing

https://doi.org/10.4153/CMB-1966-073-6 Published online by Cambridge University Press

as the result of the computation of P. After n has ranged over the finite set C, we compare these largest numbers and we choose the largest of them. The latter is precisely f(c); thus we have computed f(c), and the proof is complete.

REFERENCES

1. Z.A. Melzak, Canad. Math. Bull., vol. 4, no. 3, Sept. 1961, pages 279-293.

2. Z.A. Melzak, Canad. Math. Bull., vol. 7, no. 2, April 1964, pages 183-200.

3. T. Schneider, Einfuehrung in die transzendenten Zahlen, Springer, 1957.

University of British Columbia

https://doi.org/10.4153/CMB-1966-073-6 Published online by Cambridge University Press