CHAPTER 3

Computability

In this chapter we develop the basics of recursive function theory, or as it is more generally known, computability theory. Its history goes back to the seminal works of Turing, Kleene and others in the 1930's.

A computable function is one defined by a program whose operational semantics tell an idealized computer what to do to its storage locations as it proceeds deterministically from input to output, without any prior restrictions on storage space or computation time. We shall be concerned with various program-styles and the relationships between them, but the emphasis throughout will be on one underlying data-type, namely the natural numbers, since it is there that the most basic foundational connections between proof theory and computation are to be seen in their clearest light.

The two best-known models of machine computation are the Turing Machine and the (Unlimited) Register Machine of Shepherdson and Sturgis [**22**]. We base our development on the latter since it affords the quickest route to the results we want to establish.

1. Register Machines

1.1. Programs. A *register machine* stores natural numbers in registers denoted u, v, w, x, y, z possibly with subscripts, and it responds step by step to a *program* consisting of an ordered list of basic instructions:

$$\begin{array}{l} I_0 \\ I_1 \\ \vdots \\ I_{k-1} \end{array}$$

Each instruction has one of the following three forms whose meanings are obvious:

Zero: $x := 0$
Succ: $x := x + 1$
Jump: **if** $x = y$ **then** I_m **else** I_n .

The instructions are obeyed in order starting with I_0 except when a conditional jump instruction is encountered, in which case the next instruction will be either I_m or I_n according as the numerical contents of registers x and y are equal or not at that stage. The computation *terminates* when it runs out of instructions, that is when the next instruction called for is I_k. Thus if a program of length k contains a jump instruction as above then it must satisfy the condition $m, n \leq k$ and I_k means "halt". Notice of course that some programs do not terminate, for example the following one-liner:

$$\textbf{if } x = x \textbf{ then } I_0 \textbf{ else } I_1$$

1.2. Program Constructs. We develop some shorthand for building up standard sorts of programs.

Transfer. "$x := y$" is the program

$$\begin{aligned}&x := 0\\&\textbf{if } x = y \textbf{ then } I_4 \textbf{ else } I_2\\&x := x + 1\\&\textbf{if } x = x \textbf{ then } I_1 \textbf{ else } I_1\end{aligned}$$

which copies the contents of register y into register x.

Predecessor. The program "$x := y \dot{-} 1$" copies the modified predecessor of y into x, and simultaneously copies y into z:

$$\begin{aligned}&x := 0\\&z := 0\\&\textbf{if } x = y \textbf{ then } I_8 \textbf{ else } I_3\\&z := z + 1\\&\textbf{if } z = y \textbf{ then } I_8 \textbf{ else } I_5\\&z := z + 1\\&x := x + 1\\&\textbf{if } z = y \textbf{ then } I_8 \textbf{ else } I_5\,.\end{aligned}$$

Composition. "P ; Q" is the program obtained by concatenating program P with program Q. However in order to ensure that jump instructions in Q of the form "**if** $x = y$ **then** I_m **else** I_n" still operate properly within Q they need to be re-numbered by changing the addresses m, n to $k+m, k+n$ respectively where k is the length of program P. Thus the effect of this program is to do P until it halts (if ever) and then do Q.

Conditional. "**if** $x = y$ **then** P **else** Q **fi**" is the program

$$\begin{aligned}&\textbf{if } x = y \textbf{ then } I_1 \textbf{ else } I_{k+2}\\&\vdots P\\&\textbf{if } x = x \textbf{ then } I_{k+2+l} \textbf{ else } I_2\\&\vdots Q\end{aligned}$$

where k, l are the lengths of the programs P, Q respectively, and again their jump instructions must be appropriately renumbered by adding 1 to the addresses in P and $k + 2$ to the addresses in Q. Clearly if $x = y$ then program P is obeyed and the next jump instruction automatically bipasses Q and halts. If $x \neq y$ then program Q is performed.

For Loop. "**for** $i = 1 \ldots x$ **do** P **od**" is the program

$$\begin{aligned}&i := 0\\&\textbf{if } x = i \textbf{ then } I_{k+4} \textbf{ else } I_2\\&i := i + 1\\&\vdots P\\&\textbf{if } x = i \textbf{ then } I_{k+4} \textbf{ else } I_2\end{aligned}$$

where again, k is the length of program P and the jump instructions in P must be appropriately re-addressed by adding 3. The intention of this new program is that it should iterate the program P x times (do nothing if $x = 0$). This requires the restriction that the register x and the "local" counting-register i are not re-assigned new values inside P.

While Loop. "**while** $x \neq 0$ **do** P **od**" is the program

$$\begin{aligned}&\textbf{if } x = 0 \textbf{ then } I_{k+2} \textbf{ else } I_1\\ &\vdots P\\ &\textbf{if } x = 0 \textbf{ then } I_{k+2} \textbf{ else } I_1\end{aligned}$$

where again, k is the length of program P and the jump instructions in P must be re-addressed by adding 1. This program keeps on doing P until (if ever) the register x becomes 0.

1.3. Computable Functions. A register machine program P may have certain distinguished "input registers" and "output registers". It may also use other "working registers" for scratchwork and these will initially be set to zero. We write $P(x_1, \ldots, x_k; y)$ to signify that program P has input registers $x_1, \ldots, x_k$ and one output register y, which are distinct.

DEFINITION. The program $P(x_1, \ldots, x_k; y)$ is said to *compute* the k-ary partial function $\varphi\colon \mathbb{N}^k \to \mathbb{N}$ if, starting with any numerical values $n_1, \ldots, n_k$ in the input registers, the program terminates with the number m in the output register if and only if $\varphi(n_1, \ldots, n_k)$ is defined with value m. In this case, the input registers hold their original values.

A function is *register machine computable* if there is some program which computes it.

Here are some examples.

Addition. "$\mathsf{Add}(x, y; z)$" is the program

$$z := x \ ; \ \textbf{for } i = 1, \ldots, y \textbf{ do } z := z + 1 \textbf{ od}$$

which adds the contents of registers x and y into register z.

Subtraction. "$\mathsf{Subt}(x, y; z)$" is the program

$$z := x \ ; \ \textbf{for } i = 1, \ldots, y \textbf{ do } w := z \dot{-} 1 \ ; \ z := w \textbf{ od}$$

which computes the modified subtraction function $x \dot{-} y$.

Bounded Sum. If $P(x_1, \ldots, x_k, w; y)$ computes the $k+1$-ary function φ then the program $Q(x_1, \ldots, x_k, z; x)$:

$$\begin{aligned}&x := 0 \ ;\\ &\textbf{for } i = 1, \ldots, z \textbf{ do } w := i \dot{-} 1 \ ; \ P(\vec{x}, w; y) \ ; \ v := x \ ; \ \mathsf{Add}(v, y; x) \textbf{ od}\end{aligned}$$

computes the function

$$\psi(x_1, \ldots, x_k, z) = \sum_{w<z} \varphi(x_1, \ldots, x_k, w)$$

which will be undefined if for some $w < z$, $\varphi(x_1, \ldots, x_k, w)$ is undefined.

Multiplication. Deleting "$w := i \dot{-} 1 \ ; P$" from the last example gives a program $\mathsf{Mult}(z, y; x)$ which places the product of y and z into x.

Bounded Product. If in the bounded sum example, the instruction $x := x + 1$ is inserted immediately after $x := 0$, and if $\mathsf{Add}(v, y; x)$ is replaced by $\mathsf{Mult}(v, y; x)$, then the resulting program computes the function

$$\psi(x_1, \ldots, x_k, z) = \prod_{w<z} \varphi(x_1, \ldots, x_k, w) .$$

Composition. If $P_j(x_1, \dots, x_k; y_j)$ computes φ_j for each $j = i, \dots, m$ and if $P_0(y_1, \dots, y_m; y_0)$ computes φ_0, then the program $Q(x_1, \dots, x_k; y_0)$:

$$P_1(x_1, \dots, x_k; y_1) \ ; \ \dots \ ; \ P_m(x_1, \dots, x_k; y_m) \ ; \ P_0(y_1, \dots, y_m; y_0)$$

computes the function

$$\psi(x_1, \dots, x_k) = \varphi_0(\varphi_1(x_1, \dots, x_k), \dots, \varphi_m(x_1, \dots, x_k))$$

which will be undefined if any of the φ-subterms on the right hand side is undefined.

Unbounded Minimization. If $P(x_1, \dots, x_k, y; z)$ computes φ then the program $Q(x_1, \dots, x_k; z)$:

$y := 0$; $z := 0$; $z := z + 1$;
while $z \neq 0$ **do** $P(x_1, \dots, x_k, y; z)$; $y := y + 1$ **od** ;
$z := y \dot{-} 1$

computes the function

$$\psi(x_1, \dots, x_k) \ = \ \mu y \, (\varphi(x_1, \dots, x_k, y) = 0)$$

that is, the *least number* y such that $\varphi(x_1, \dots, x_k, y')$ is defined for every $y' \leq y$ and $\varphi(x_1, \dots, x_k, y) = 0$.

2. Elementary Functions

2.1. Definition and Simple Properties. The *elementary functions* of Kalmár (1943) are those number-theoretic functions which can be defined explicitly by compositional terms built up from variables and the constants $0, 1$ by repeated applications of addition $+$, modified subtraction $\dot{-}$, bounded sums and bounded products.

By omitting bounded products, one obtains the *subelementary* functions.

The examples in the previous section show that all elementary functions are computable and totally defined. Multiplication and exponentiation are elementary since

$$m \cdot n = \sum_{i<n} m \ \text{ and } \ m^n = \prod_{i<n} m$$

and hence by repeated composition, all exponential polynomials are elementary.

In addition the elementary functions are closed under

Definitions by Cases.

$$f(\vec{n}) = \begin{cases} g_0(\vec{n}) & \text{if } h(\vec{n}) = 0 \\ g_1(\vec{n}) & \text{otherwise} \end{cases}$$

since f can be defined from g_0, g_1 and h by

$$f(\vec{n}) = g_0(\vec{n}) \cdot (1 \dot{-} h(\vec{n})) + g_1(\vec{n}) \cdot (1 \dot{-} (1 \dot{-} h(\vec{n}))).$$

Bounded Minimization.

$$f(\vec{n}, m) = \mu\, k{<}m\, (g(\vec{n}, k) = 0)$$

since f can be defined from g by

$$f(\vec{n}, m) = \sum_{i<m} \big(1 \dot{-} \sum_{k\leq i} (1 \dot{-} g(\vec{n}, k))\big).$$

Note: this definition gives value m if there is no $k < m$ such that $g(\vec{n}, k) = 0$. It shows that not only the elementary, but in fact the subelementary functions are closed under bounded minimization. Furthermore, we define $\mu\, k{\leq}m\, (g(\vec{n}, k) = 0)$ as $\mu\, k{<}m{+}1\, (g(\vec{n}, k) = 0)$. Another notational convention will be that we shall often replace the brackets in $\mu\, k{<}m\, (g(\vec{n}, k) = 0)$ by a dot, thus: $\mu\, k{<}m.\, g(\vec{n}, k) = 0$.

LEMMA.

(a) *For every elementary function $f \colon \mathbb{N}^r \to \mathbb{N}$ there is a number k such that for all $\vec{n} = n_1, \dots, n_r$,*

$$f(\vec{n}) \;<\; 2_k \max(\vec{n})$$

where $2_0(m) = m$ and $2_{k+1}(m) = 2^{2_k(m)}$.

(b) *Hence the function $n \mapsto 2_n(1)$ is not elementary.*

PROOF. (a). By induction on the build-up of the compositional term defining f. The result clearly holds if f is any one of the base functions:

$$f(\vec{n}) = 0 \text{ or } 1 \text{ or } n_i \text{ or } n_i + n_j \text{ or } n_i \dot{-} n_j.$$

If f is defined from g by application of bounded sum or product:

$$f(\vec{n}, m) = \sum_{i<m} g(\vec{n}, i) \text{ or } \prod_{i<m} g(\vec{n}, i)$$

where $g(\vec{n}, i) < 2_k \max(\vec{n}, i)$ then we have

$$f(\vec{n}, m) \leq 2_k \max(\vec{n}, m)^m < 2_{k+2} \max(\vec{n}, m)$$

(using $m^m < 2^{2^m}$). If f is defined from $g_0, g_1, \dots, g_l$ by composition:

$$f(\vec{n}) = g_0(g_1(\vec{n}), \dots, g_l(\vec{n}))$$

where for each $j \leq l$ we have $g_j(-) < 2_{k_j}(\max(-))$, then with $k = \max_j k_j$,

$$f(\vec{n}) < 2_k(2_k \max(\vec{n})) = 2_{2k} \max(\vec{n})$$

and this completes the first part.

(b). If $2_n(1)$ were an elementary function of n then by (a) there would be a positive k such that for all n,

$$2_n(1) < 2_k(n)$$

but then putting $n = 2_k(1)$ yields $2_{2_k(1)}(1) < 2_{2k}(1)$, a contradiction. $\square$

2.2. Elementary Relations. A relation R on $\mathbb{N}^k$ is said to be *elementary* if its characteristic function

$$c_R(\vec{n}) = \begin{cases} 1 & \text{if } R(\vec{n}) \\ 0 & \text{otherwise} \end{cases}$$

is elementary. In particular, the "equality" and "less than" relations are elementary since their characteristic functions can be defined as follows:

$$c_<(m,n) = 1 \dot{-} (1 \dot{-} (n \dot{-} m)) \ ; \ c_=(m,n) = 1 \dot{-} (c_<(m,n) + c_<(n,m)) .$$

Furthermore if R is elementary then so is the function

$$f(\vec{n}, m) = \mu\, k{<}m\, R(\vec{n}, k)$$

since $R(\vec{n}, k)$ is equivalent to $1 \dot{-} c_R(\vec{n}, k) = 0$.

LEMMA. *The elementary relations are closed under applications of propositional connectives and bounded quantifiers.*

PROOF. For example, the characteristic function of $\neg R$ is

$$1 \dot{-} c_R(\vec{n}) .$$

The characteristic function of $R_0 \wedge R_1$ is

$$c_{R_0}(\vec{n}) \cdot c_{R_1}(\vec{n}).$$

The characteristic function of $\forall i{<}m\, R(\vec{n}, i)$ is

$$c_=(m, \mu i{<}m.\ c_R(\vec{n}, i) = 0).$$

□

EXAMPLES. The above closure properties enable us to show that many "natural" functions and relations of number theory are elementary; thus

$$\begin{aligned}
\lfloor \tfrac{m}{n} \rfloor &= \mu\, k{<}m\, (m < (k+1)n) \\
m \bmod n &= m \dot{-} \lfloor \tfrac{m}{n} \rfloor n \\
\mathsf{Prime}(m) &\leftrightarrow 1 < m \wedge \neg\exists n{<}m (1 < n \wedge m \bmod n = 0) \\
p_n &= \mu\, m{<}2^{2^n} \big(\mathsf{Prime}(m) \wedge n = \sum_{i<m} c_{\mathsf{Prime}}(i)\big)
\end{aligned}$$

so $p_0, p_1, p_2, \dots$ gives the enumeration of primes in increasing order. The estimate $p_n \le 2^{2^n}$ for the nth prime p_n can be proved by induction on n: For $n = 0$ this is clear, and for $n \ge 1$ we obtain

$$p_n \le p_0 p_1 \cdots p_{n-1} + 1 \le 2^{2^0} 2^{2^1} \cdots 2^{2^{n-1}} + 1 = 2^{2^n - 1} + 1 < 2^{2^n}.$$

2.3. The Class $\mathcal{E}$.

DEFINITION. The class $\mathcal{E}$ consists of those number theoretic functions which can be defined from the initial functions: constant 0, successor S, projections (onto the ith coordinate), addition $+$, modified subtraction $\dot{-}$, multiplication $\cdot$ and exponentiation 2^x, by applications of composition and bounded minimization.

The remarks above show immediately that the characteristic functions of the equality and less than relations lie in $\mathcal{E}$, and that (by the proof of the lemma) the relations in $\mathcal{E}$ are closed under propositional connectives and bounded quantifiers.

Furthermore the above examples show that all the functions in the class $\mathcal{E}$ are elementary. We now prove the converse, which will be useful later.

LEMMA. *There are "pairing functions" π, π_1, π_2 in $\mathcal{E}$ with the following properties:*

(a) π *maps* $\mathbb{N} \times \mathbb{N}$ *bijectively onto* $\mathbb{N}$,
(b) $\pi(a, b) < (a + b + 1)^2$,
(c) $\pi_1(c), \pi_2(c) \leq c$,
(d) $\pi(\pi_1(c), \pi_2(c)) = c$,
(e) $\pi_1(\pi(a, b)) = a$,
(f) $\pi_2(\pi(a, b)) = b$.

PROOF. Enumerate the pairs of natural numbers as follows:

$$\begin{array}{lllll}
\vdots & & & & \\
10 & & & & \\
6 & \dots & & & \\
3 & 7 & \dots & & \\
1 & 4 & 8 & \dots & \\
0 & 2 & 5 & 9 & \dots
\end{array}$$

At position $(0, b)$ we clearly have the sum of the lengths of the preceeding diagonals, and on the next diagonal $a + b$ remains constant. Let $\pi(a, b)$ be the number written at position (a, b). Then we have

$$\pi(a, b) = \Big(\sum_{i \leq a+b} i\Big) + a = \frac{1}{2}(a + b)(a + b + 1) + a.$$

Clearly $\pi \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is bijective. Moreover, $a, b \leq \pi(a, b)$ and in case $\pi(a, b) \neq 0$ also $a < \pi(a, b)$. Let

$$\pi_1(c) := \mu x{\leq}c \exists y{\leq}c\, (\pi(x, y) = c),$$
$$\pi_2(c) := \mu y{\leq}c \exists x{\leq}c\, (\pi(x, y) = c).$$

Then clearly $\pi_i(c) \leq c$ for $i \in \{1, 2\}$ and

$$\pi_1(\pi(a, b)) = a,$$
$$\pi_2(\pi(a, b)) = b,$$
$$\pi(\pi_1(c), \pi_2(c)) = c.$$

π, π_1 and π_2 are elementary by definiton. $\square$

REMARK. The above proof shows that π, π_1 and π_2 are in fact subelementary.

LEMMA (Gödel). *There is in $\mathcal{E}$ a function β with the following property: For every sequence $a_0, \dots, a_{n-1} < b$ of numbers less than b we can find a number $c \leq 4 \cdot 4^{n(b+n+1)^4}$ such that $\beta(c, i) = a_i$ for all $i < n$.*

PROOF. Let

$$a := \pi(b,n) \quad \text{and} \quad d := \prod_{i<n} \big(1 + \pi(a_i, i)a!\big).$$

From $a!$ and d we can, for each given $i < n$, reconstruct the number a_i as the unique $x < b$ such that

$$1 + \pi(x,i)a! \mid d.$$

For clearly a_i is such an x, and if some $x < b$ were to satisfy the same condition, then because $\pi(x,i) < a$ and the numbers $1 + ka!$ are relatively prime for $k \leq a$, we would have $\pi(x,i) = \pi(a_j, j)$ for some $j < n$. Hence $x = a_j$ and $i = j$, thus $x = a_i$.

We can now define the *Gödel β-function* as

$$\beta(c,i) := \pi_1\big(\mu y{<}c.\ (1 + \pi(\pi_1(y), i) \cdot \pi_1(c)) \cdot \pi_2(y) = \pi_2(c)\big).$$

Clearly β is in $\mathcal{E}$. Furthermore with $c := \pi(a!, d)$ we see that $\pi(a_i, \lceil d/1 + \pi(a_i, i)a! \rceil)$ is the unique such y, and therefore $\beta(c,i) = a_i$. It is then not difficult to estimate the given bound on c, using $\pi(b,n) < (b+n+1)^2$. $\square$

REMARK. The above definition of β shows that it is subelementary.

2.4. Closure Properties of $\mathcal{E}$.

THEOREM. *The class $\mathcal{E}$ is closed under limited recursion. Thus if g, h, k are given functions in $\mathcal{E}$ and f is defined from them according to the scheme*

$$\begin{aligned} f(\vec{m}, 0) &= g(\vec{m}) \\ f(\vec{m}, n+1) &= h(n, f(\vec{m}, n), \vec{m}) \\ f(\vec{m}, n) &\leq k(\vec{m}, n) \end{aligned}$$

then f is in $\mathcal{E}$ also.

PROOF. Let f be defined from g, h and k in $\mathcal{E}$, by limited recursion as above. Using Gödel's β-function as in the last lemma we can find for any given $\vec{m}, n$ a number c such that $\beta(c,i) = f(\vec{m}, i)$ for all $i \leq n$. Let $R(\vec{m}, n, c)$ be the relation

$$\beta(c, 0) = g(\vec{m}) \wedge \forall i{<}n.\ \beta(c, i+1) = h(i, \beta(c,i), \vec{m})$$

and note by the remarks above that its characteristic function is in $\mathcal{E}$. It is clear, by induction, that if $R(\vec{m}, n, c)$ holds then $\beta(c,i) = f(\vec{m}, i)$, for all $i \leq n$. Therefore we can define f explicitly by the equation

$$f(\vec{m}, n) = \beta(\mu c\, R(\vec{m}, n, c), n).$$

f will lie in $\mathcal{E}$ if μc can be bounded by an $\mathcal{E}$ function. However, Lemma 2.3 gives a bound $4 \cdot 4^{(n+1)(b+n+2)^4}$, where in this case b can be taken as the maximum of $k(\vec{m}, i)$ for $i \leq n$. But this can be defined in $\mathcal{E}$ as $k(\vec{m}, i_0)$, where $i_0 = \mu i{\leq}n.\ \forall j{\leq}n.\ k(\vec{m}, j) \leq k(\vec{m}, i)$. Hence μc can be bounded by an $\mathcal{E}$ function. $\square$

REMARK. Notice that it is in this proof only that the exponential function is required, in providing a bound for μ.

COROLLARY. *$\mathcal{E}$ is the class of all elementary functions.*

PROOF. It is sufficient merely to show that $\mathcal{E}$ is closed under bounded sums and bounded products. Suppose for instance, that f is defined from g in $\mathcal{E}$ by bounded summation: $f(\vec{m}, n) = \sum_{i<n} g(\vec{m}, i)$. Then f can be defined by limited recursion, as follows

$$\begin{aligned} f(\vec{m}, 0) &= 0 \\ f(\vec{m}, n+1) &= f(\vec{m}, n) + g(\vec{m}, n) \\ f(\vec{m}, n) &\leq n \cdot \max_{i<n} g(\vec{m}, i) \end{aligned}$$

and the functions (including the bound) from which it is defined are in $\mathcal{E}$. Thus f is in $\mathcal{E}$ by the last lemma. If instead, f is defined by bounded product, then proceed similarly. $\square$

2.5. Coding Finite Lists. Computation on lists is a practical necessity, so because we are basing everything here on the single data type $\mathbb{N}$ we must develop some means of "coding" finite lists or sequences of natural numbers into $\mathbb{N}$ itself. There are various ways to do this and we shall adopt one of the most traditional, based on the pairing functions π, π_1, π_2.

The empty sequence is coded by the number 0 and a sequence n_0, n_1, $\dots$, n_{k-1} is coded by the "sequence number"

$$\langle n_0, n_1, \dots, n_{k-1} \rangle = \pi'(\dots \pi'(\pi'(0, n_0), n_1), \dots, n_{k-1})$$

with $\pi'(a, b) := \pi(a, b) + 1$, thus recursively,

$$\begin{aligned} &\langle\rangle := 0, \\ &\langle n_0, n_1, \dots, n_k \rangle := \pi'(\langle n_0, n_1, \dots, n_{k-1} \rangle, n_k). \end{aligned}$$

Because of the surjectivity of π, every number a can be decoded uniquely as a sequence number $a = \langle n_0, n_1, \dots, n_{k-1} \rangle$. If a is greater than zero, $\mathsf{hd}(a) := \pi_2(a \dot{-} 1)$ is the "head" (i.e. rightmost element) and $\mathsf{tl}(a) := \pi_1(a \dot{-} 1)$ is the "tail" of the list. The kth iterate of tl is denoted $\mathsf{tl}^{(k)}$ and since $\mathsf{tl}(a)$ is less than or equal to a, $\mathsf{tl}^{(k)}(a)$ is elementarily definable (by limited recursion). Thus we can define elementarily the "length" and "decoding" functions:

$$\begin{aligned} \mathsf{lh}(a) &:= \mu k{\leq}a.\ \mathsf{tl}^{(k)}(a) = 0, \\ (a)_i &:= \mathsf{hd}(\mathsf{tl}^{(\mathsf{lh}(a) \dot{-} (i+1))}(a)). \end{aligned}$$

Then if $a = \langle n_0, n_1, \dots, n_{k-1} \rangle$ it is easy to check that

$$\mathsf{lh}(a) = k \text{ and } (a)_i = n_i \text{ for each } i < k.$$

Furthermore $(a)_i = 0$ when $i \geq \mathsf{lh}(a)$. We shall write $(a)_{i,j}$ for $((a)_i)_j$ and $(a)_{i,j,k}$ for $(((a)_i)_j)_k$. This elementary coding machinery will be used at various crucial points in the following.

Note that our previous remarks show that the functions lh and $(a)_i$ are subelementary, and so is $\langle n_0, n_1, \dots, n_{k-1} \rangle$ for each fixed k .

Concatenation of sequence numbers $b \star a$ is defined thus:

$$\begin{aligned} &b \star \langle\rangle := b, \\ &b \star \langle n_0, n_1, \dots, n_k \rangle := \pi(b \star \langle n_0, n_1, \dots, n_{k-1} \rangle, n_k) + 1. \end{aligned}$$

To check that this operation is also elementary, define $h(b,a,i)$ by recursion on i as follows.

$$\begin{aligned} h(b,a,0) &= b, \\ h(b,a,i+1) &= \pi(h(b,a,i),(a)_i)+1 \end{aligned}$$

and note that since $\pi(h(b,a,i),(a)_i) < (h(b,a,i)+a)^2$ it follows by induction on i that $h(b,a,i)$ is less than or equal to $(b+a+i)^{2^i}$. Thus h is definable by limited recursion from elementary functions and hence is itself elementary. Finally

$$b \star a = h(b,a,\mathsf{lh}(a)).$$

Lemma. *The class $\mathcal{E}$ is closed under limited course-of-values recursion. Thus if h, k are given functions in $\mathcal{E}$ and f is defined from them according to the scheme*

$$\begin{aligned} &f(\vec{m},n) = h(n,\langle f(\vec{m},0),\dots.f(\vec{m},n-1)\rangle,\vec{m}) \\ &f(\vec{m},n) \le k(\vec{m},n) \end{aligned}$$

then f is in $\mathcal{E}$ also.

Proof. $\bar{f}(\vec{m},n) := \langle f(\vec{m},0),\dots.f(\vec{m},n-1)\rangle$ is definable by

$$\begin{aligned} \bar{f}(\vec{m},0) &= 0, \\ \bar{f}(\vec{m},n+1) &= \bar{f}(\vec{m},n) \star \langle h(n,\bar{f}(\vec{m},n),\vec{m})\rangle \\ \bar{f}(\vec{m},n) &\le \Big(\sum_{i\le n} k(\vec{m},i)+1\Big)^{2^n}, \quad \text{using } \langle \underbrace{n,\dots,n}_{k}\rangle < (n+1)^{2^k} \end{aligned}$$

□

3. The Normal Form Theorem

3.1. Program Numbers. The three types of register machine instructions I can be coded by "instruction numbers" $\sharp I$ thus, where $v_0, v_1, v_2, \dots$ is a list of all variables used to denote registers:

If I is "$v_j := 0$" then $\sharp I = \langle 0, j\rangle$.
If I is "$v_j := v_j + 1$" then $\sharp I = \langle 1, j\rangle$.
If I is "**if** $v_j = v_l$ **then** I_m **else** I_n" then $\sharp I = \langle 2, j, l, m, n\rangle$.

Clearly, using the sequence coding and decoding apparatus above, we can check elementarily whether or not a given number is an instruction number.

Any register machine program $P = I_0, I_1, \dots, I_{k-1}$ can then be coded by a "program number" or "index" $\sharp P$ thus:

$$\sharp P \;=\; \langle\, \sharp I_0, \sharp I_1, \dots, \sharp I_{k-1}\,\rangle$$

and again (although it is tedious) we can elementarily check whether or not a given number is indeed of the form $\sharp P$ for some program P. Tradition has it that e is normally reserved as a variable over putative program numbers.

Standard program constructs such as those in Section 1 have associated "index-constructors", i.e. functions which, given indices of the subprograms, produce an index for the constructed program. The point is that for standard program constructs the associated index-constructor functions are elementary. For example there is an elementary index-constructor comp such that, given programs P_0, P_1 with indices e_0, e_1, $\mathsf{comp}(e_0, e_1)$ is an index of the program P_0 ; P_1. A moment's thought should convince the reader that the appropriate definition of comp is as follows:

$$\mathsf{comp}(e_0, e_1) \;=\; e_0 \star \langle r(e_0, e_1, 0), r(e_0, e_1, 1), \dots, r(e_0, e_1, \mathsf{lh}(e_1) \dot{-} 1)\rangle$$

where $r(e_0, e_1, i) =$

$$\begin{cases} \langle 2, (e_1)_{i,1}, (e_1)_{i,2}, (e_1)_{i,3} + \mathsf{lh}(e_0), (e_1)_{i,4} + \mathsf{lh}(e_0)\rangle & \text{if } (e_1)_{i,0} = 2 \\ (e_1)_i & \text{otherwise} \end{cases}$$

re-addresses the jump instructions in P_1. Clearly r and hence comp are elementary functions.

Definition. Henceforth, $\varphi_e^{(r)}$ denotes the partial function computed by the register machine program with program number e, operating on the input registers $v_1, \dots, v_r$ and with output register v_0. There is no loss of generality here, since the variables in any program can always be renamed so that $v_1, \dots, v_r$ become the input registers and v_0 the output. If e is not a program number, or it is but does not operate on the right variables, then we adopt the convention that $\varphi_e^{(r)}(n_1, \dots, n_r)$ is undefined for all inputs $n_1, \dots, n_r$.

3.2. Normal Form.

Theorem (Kleene's Normal Form). *For each arity r there is an elementary function U and an elementary relation T such that, for all e and all inputs $n_1, \dots, n_r$,*

- $\varphi_e^{(r)}(n_1, \dots, n_r)$ *is defined* $\iff \exists s\, T(e, n_1, \dots, n_r, s)$
- $\varphi_e^{(r)}(n_1, \dots, n_r) \;=\; U(\, e, n_1, \dots, n_r, \mu s\, T(e, n_1, \dots, n_r, s)\,).$

Proof. A computation of a register machine program $P(v_1, \dots, v_r; v_0)$ on numerical inputs $\vec{n} = n_1, \dots, n_r$ proceeds deterministically, step by step, each step corresponding to the execution of one instruction. Let e be its program number, and let $v_0, \dots, v_l$ be all the registers used by P, including the "working registers" so $r \le l$.

The "state" of the computation at step s is defined to be the sequence number

$$\mathsf{state}(e, \vec{n}, s) = \langle e, i, m_0, m_1, \dots, m_l\rangle$$

where $m_0, m_1, \dots, m_l$ are the values stored in the registers $v_0, v_1, \dots, v_l$ after step s is completed, and the next instruction to be performed is the ith one, thus $(e)_i$ is its instruction number.

The "state transition function" $\mathsf{tr}\colon \mathbb{N} \to \mathbb{N}$ computes the "next state". So suppose that $x = \langle e, i, m_0, m_1, \dots, m_l\rangle$ is any putative state. Then in what follows, $e = (x)_0$, $i = (x)_1$, and $m_j = (x)_{j+2}$ for each $j \le l$. The definition of $\mathsf{tr}(x)$ is therefore as follows:

$$\mathsf{tr}(x) \;=\; \langle e, i', m_0', m_1', \dots, m_l'\rangle$$

where

- If $(e)_i = \langle 0, j\rangle$ where $j \le l$ then $i' = i + 1$, $m_j' = 0$, and all other registers remain unchanged, i.e. $m_k' = m_k$ for $k \ne j$.

- If $(e)_i = \langle 1, j\rangle$ where $j \leq l$ then $i' = i+1$, $m'_j = m_j + 1$, and all other registers remain unchanged.
- If $(e)_i = \langle 2, j_0, j_1, i_0, i_1\rangle$ where $j_0, j_1 \leq l$ and $i_0, i_1 \leq \mathsf{lh}(e)$ then $i' = i_0$ or $i' = i_1$ according as $m_{j_0} = m_{j_1}$ or not, and all registers remain unchanged, i.e. $m'_j = m_j$ for all $j \leq l$.
- Otherwise, if x is not a sequence number, or if e is not a program number, or if it refers to a register v_k with $l < k$, or if $\mathsf{lh}(e) \leq i$, then $\mathsf{tr}(x)$ simply repeats the same state x so $i' = i$, and $m'_j = m_j$ for every $j \leq l$.

Clearly tr is an *elementary* function, since it is defined by elementarily decidable cases, with (a great deal of) elementary decoding and re-coding involved in each case.

Consequently, the "state function" $\mathsf{state}(e, \vec{n}, s)$ is also *elementary* because it can be defined by iterating the transition function by limited recursion on s as follows:

$$\begin{aligned} \mathsf{state}(e, \vec{n}, 0) &= \langle e, 0, n_1, \dots, n_r, 0, \dots, 0\rangle \\ \mathsf{state}(e, \vec{n}, s+1) &= \mathsf{tr}(\, \mathsf{state}(e, \vec{n}, s)\,) \\ \mathsf{state}(e, \vec{n}, s) &\leq h(e, \vec{n}, s) \end{aligned}$$

where for the bounding function h we can take

$$h(e, \vec{n}, s) = \langle e, e\rangle \star \langle \max(\vec{n}) + s, \dots, \max(\vec{n}) + s\rangle,$$

This is because the maximum value of any register at step s cannot be greater than $\max(\vec{n}) + s$. Now this expression clearly is elementary, since $\langle m, \dots, m\rangle$ with i occurrences of m is definable by a limited recursion with bound $(m+i)^{2^i}$, as is easily seen by induction on i.

Now recall that if program P has program number e then computation terminates when instruction $I_{\mathsf{lh}(e)}$ is encountered. Thus we can define the "termination relation" $T(e, \vec{n}, s)$ meaning "computation terminates at step s", by

$$T(e, \vec{n}, s) \iff (\, \mathsf{state}(e, \vec{n}, s)\,)_1 = \mathsf{lh}(e).$$

Clearly T is elementary and

$$\varphi_e^{(r)}(\vec{n}) \text{ is defined} \iff \exists s\, T(e, \vec{n}, s).$$

The output on termination is the value of register v_0, so if we define the "output function" $U(e, \vec{n}, s)$ by

$$U(e, \vec{n}, s) \;=\; (\, \mathsf{state}(e, \vec{n}, s)\,)_2$$

then U is also elementary and

$$\varphi_e^{(r)}(\vec{n}) \;=\; U(e, \vec{n}, \mu s\, T(e, \vec{n}, s)).$$

This completes the proof. □

3.3. Σ^0_1-Definable Relations and μ-Recursive Functions. A relation R of arity r is said to be Σ^0_1-*definable* if there is an elementary relation E, say of arity $r + l$, such that for all $\vec{n} = n_1, \dots, n_r$,

$$R(\vec{n}) \iff \exists k_1 \dots \exists k_l\, E(\vec{n}, k_1, \dots, k_l).$$

A partial function φ is said to be Σ^0_1-*definable* if its graph

$$\{\, (\vec{n}, m) \mid \varphi(\vec{n}) \text{ is defined and } = m \,\}$$

is Σ^0_1-definable.

To say that a non-empty relation R is Σ^0_1-definable is equivalent to saying that the set of all sequences $\langle \vec{n} \rangle$ satisfying R can be enumerated (possibly with repetitions) by some elementary function $f \colon \mathbb{N} \to \mathbb{N}$. Such relations are called *elementarily enumerable*. For choose any fixed sequence $\langle a_1, \dots, a_r \rangle$ satisfying R and define

$$f(m) \;=\; \begin{cases} \langle (m)_1, \dots, (m)_r \rangle & \text{if } E((m)_1, \dots, (m)_{r+l}) \\ \langle a_1, \dots, a_r \rangle & \text{otherwise.} \end{cases}$$

Conversely if R is elementarily enumerated by f then

$$R(\vec{n}) \iff \exists m\, (f(m) = \langle \vec{n} \rangle)$$

is a Σ^0_1-definition of R.

The μ-*recursive functions* are those (partial) functions which can be defined from the initial functions: constant 0, successor S, projections (onto the ith coordinate), addition $+$, modified subtraction $\dot{-}$ and multiplication $\cdot$, by applications of composition and unbounded minimization. Note that it is through unbounded minimization that partial functions may arise.

LEMMA. *Every elementary function is μ-recursive.*

PROOF. By simply removing the bounds on μ in the lemmas in 2.3 one obtains μ-recursive definitions of the pairing functions π, π_1, π_2 and of Gödel's β-function. Then by removing all mention of bounds from Theorem in 2.4 one sees that the μ-recursive functions are closed under (unlimited) primitive recursive definitions: $f(\vec{m}, 0) = g(\vec{m})$, $f(\vec{m}, n+1) = h(n, f(\vec{m}, n))$. Thus one can μ-recursively define bounded sums and bounded products, and hence all elementary functions. $\square$

3.4. Computable Functions.

DEFINITION. The *while-programs* are those programs which can be built up from assignment statements $x := 0$, $x := y$, $x := y + 1$, $x := y \dot{-} 1$, by Conditionals, Composition, For-Loops and While-Loops as in the subsection on program constructs in Section 1.

THEOREM. *The following are equivalent:*

(a) *φ is register machine computable,*
(b) *φ is Σ^0_1-definable,*
(c) *φ is μ-recursive,*
(d) *φ is computable by a while program.*

PROOF. The Normal Form Theorem shows immediately that every register machine computable function $\varphi^{(r)}_e$ is Σ^0_1-definable since

$$\varphi^{(r)}_e(\vec{n}) = m \iff \exists s . T(e, \vec{n}, s) \wedge U(e, \vec{n}, s) = m$$

and the relation $T(e, \vec{n}, s) \wedge U(e, \vec{n}, s) = m$ is clearly elementary. If φ is Σ^0_1-definable, say

$$\varphi(\vec{n}) = m \iff \exists k_1 \dots \exists k_l\, E(\vec{n}, m, k_1, \dots, k_l)$$

then φ can be defined μ-recursively by

$$\varphi(\vec{n}) \;=\; (\,\mu m\, E(\vec{n},(m)_0,(m)_1,\dots,(m)_l)\,)_0\;,$$

using the fact (above) that elementary functions are μ-recursive. The examples of computable functionals in Section 1 show how the definition of any μ-recursive function translates automatically into a while program. Finally, the subsection on program constructs in Section 1 shows how to implement any while program on a register machine. □

Henceforth *computable* means "register machine computable" or any of its equivalents.

COROLLARY. *The function $\varphi_e^{(r)}(n_1,\dots,n_r)$ is a computable partial function of the $r+1$ variables $e,n_1,\dots,n_r$.*

PROOF. Immediate from the Normal Form. □

LEMMA. *A relation R is computable if and only if both R and its complement $\mathbb{N}^n \setminus R$ are Σ_1^0-definable.*

PROOF. We can assume that both R and $\mathbb{N}^n\setminus R$ are not empty, and (for simplicity) also $n=1$.

$\Rightarrow$. By the theorem above every computable relation is Σ_1^0-definable, and with R clearly its complement is computable.

$\Leftarrow$. Let $f,g\in\mathcal{E}$ enumerate R and $\mathbb{N}\setminus R$, respectively. Then

$$h(n) := \mu i. f(i)=n \vee g(i)=n$$

is a total μ-recursive function, and $R(n)\leftrightarrow f(h(n))=n$. □

3.5. Undecidability of the Halting Problem. The above corollary says that there is a single "universal" program which, given numbers e and $\vec{n}$, computes $\varphi_e^{(r)}(\vec{n})$ if it is defined. However we cannot decide in advance whether or not it will be defined. There is no program which, given e and $\vec{n}$, computes the total function

$$h(e,\vec{n}) \;=\; \begin{cases} 1 & \text{if } \varphi_e^{(r)}(\vec{n}) \text{ is defined,}\\ 0 & \text{if } \varphi_e^{(r)}(\vec{n}) \text{ is undefined.}\end{cases}$$

For suppose there were such a program. Then the function

$$\psi(\vec{n}) \;=\; \mu m\,(h(n_1,\vec{n})=0)$$

would be computable, say with fixed program number e_0, and therefore

$$\varphi_{e_0}^{(r)}(\vec{n}) \;=\; \begin{cases} 0 & \text{if } h(n_1,\vec{n})=0\\ \text{undefined} & \text{if } h(n_1,\vec{n})=1\end{cases}$$

But then fixing $n_1=e_0$ gives:

$$\varphi_{e_0}^{(r)}(\vec{n}) \text{ defined} \iff h(e_0,\vec{n})=0 \iff \varphi_{e_0}^{(r)}(\vec{n}) \text{ undefined}$$

a contradiction. Hence the relation $R(e,\vec{n})$ which holds if and only if $\varphi_e^{(r)}(\vec{n})$ is defined, is not recursive. It is however Σ_1^0-definable.

There are numerous attempts to classify total computable functions according to the complexity of their termination proofs.

4. Recursive Definitions

4.1. Least Fixed Points of Recursive Definitions. By a *recursive definition* of a partial function φ of arity r from given partial functions $\psi_1, \ldots, \psi_m$ of fixed but unspecified arities, we mean a defining equation of the form

$$\varphi(n_1, \ldots, n_r) = t(\psi_1, \ldots, \psi_m, \varphi; n_1, \ldots, n_r)$$

where t is any compositional term built up from the numerical variables $\vec{n} = n_1, \ldots, n_r$ and the constant 0 by repeated applications of the successor and predecessor functions, the given functions $\psi_1, \ldots, \psi_m$, the function φ itself, and the "definition by cases" function :

$$\mathsf{dc}(x, y, u, v) = \begin{cases} u & \text{if } x, y \text{ are both defined and equal} \\ v & \text{if } x, y \text{ are both defined and unequal} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Our notion of recursive definition is essentially a reformulation of the Herbrand-Gödel-Kleene equation calculus; see Kleene [**15**].

There may be many partial functions φ satisfying such a recursive definition, but the one we wish to single out is the least defined one, i.e. the one whose defined values arise inevitably by *lazy evaluation* of the term t "from the outside in", making only those function calls which are absolutely necessary. This presupposes that each of the functions from which t is constructed already comes equipped with an evaluation strategy. In particular if a subterm $\mathsf{dc}(t_1, t_2, t_3, t_4)$ is called then it is to be evaluated according to the program construct:

$$x := t_1 \ ; \ y := t_2 \ ; \ \textbf{if } x := y \textbf{ then } t_3 \textbf{ else } t_4.$$

Some of the function calls demanded by the term t may be for further values of φ itself, and these must be evaluated by repeated unravellings of t (in other words by recursion).

This "least solution" φ will be referred to as *the function defined by that recursive definition* or its *least fixed point*. Its existence and its computability are guaranteed by Kleene's Recursion Theorem below.

4.2. The Principles of Finite Support and Monotonicity, and the Effective Index Property. Suppose we are given any fixed partial functions $\psi_1, \ldots, \psi_m$ and ψ, of the appropriate arities, and fixed inputs $\vec{n}$. If the term $t = t(\psi_1, \ldots, \psi_m, \psi; \vec{n})$ evaluates to a defined value k then the following principles are required to hold:

Finite Support Principle. Only finitely many values of $\psi_1, \ldots, \psi_m$ and ψ are used in that evaluation of t.

Monotonicity Principle. The same value k will be obtained no matter how the partial functions $\psi_1, \ldots, \psi_m$ and ψ are extended.

Note also that any such term t satisfies the

Effective Index Property. There is an elementary function f such that if $\psi_1, \ldots, \psi_m$ and ψ are computable partial functions with program numbers $e_1, \ldots, e_m$ and e respectively, then according to the lazy evaluation strategy just described,

$$t(\psi_1, \ldots, \psi_m, \psi; \vec{n})$$

defines a computable function of $\vec{n}$ with program number $f(e_1, \ldots, e_m, e)$.

The proof of the Effective Index Property is by induction over the build-up of the term t. The base case is where t is just one of the constants $0, 1$ or a variable n_j, in which case it defines either a constant function $\vec{n} \mapsto 0$ or $\vec{n} \mapsto 1$, or a projection function $\vec{n} \mapsto n_j$. Each of these is trivially computable with a fixed program number, and it is this program number we take as the value of $f(e_1, \ldots, e_m, e)$. Since in this case f is a constant function, it is clearly elementary. The induction step is where t is built up by applying one of the given functions: successor, predecessor, definition by cases or ψ (with or without a subscript) to previously constructed subterms $t_i(\psi_1, \ldots, \psi_m, \psi; \vec{n})$, $i = 1 \ldots l$, thus:

$$t \;=\; \psi(t_1, \ldots, t_l).$$

Inductively we can assume that for each $i = 1 \ldots l$, t_i defines a partial function of $\vec{n} = n_1, \ldots, n_r$ which is register machine computable by some program P_i with program number given by an already-constructed elementary function $f_i = f_i(e_1, \ldots, e_m, e)$. Therefore if ψ is computed by a program Q with program number e, we can put $P_1, \ldots, P_l$ and Q together to construct a new program obeying the evaluation strategy for t. Furthermore, by the remark on index-constructions near the beginning of Section 3, we will be able to compute its program number $f(e_1, \ldots, e_m, e)$ from the given numbers $f_1, \ldots, f_l$ and e, by some elementary function.

4.3. Recursion Theorem.

THEOREM (Kleene's Recursion Theorem). *For given partial functions $\psi_1, \ldots, \psi_m$, every recursive definition*

$$\varphi(\vec{n}) \;=\; t(\psi_1, \ldots, \psi_m, \varphi; \vec{n})$$

has a least fixed point, i.e. a least defined solution, φ. Moreover if $\psi_1, \ldots, \psi_m$ are computable, so is the least fixed point φ.

PROOF. Let $\psi_1, \ldots, \psi_m$ be fixed partial functions of the appropriate arities. Let Φ be the functional from partial functions of arity r to partial functions of arity r defined by lazy evaluation of the term t as described above:

$$\Phi(\psi)(\vec{n}) \;=\; t(\psi_1, \ldots, \psi_m, \psi; \vec{n}).$$

Let $\varphi_0, \varphi_1, \varphi_2, \ldots$ be the sequence of partial functions of arity r generated by Φ thus: φ_0 is the completely undefined function, and $\varphi_{i+1} = \Phi(\varphi_i)$ for each i. Then by induction on i, using the Monotonicity Principle above, we see that each φ_i is a subfunction of φ_{i+1}. That is, whenever $\varphi_i(\vec{n})$ is defined with a value k then $\varphi_{i+1}(\vec{n})$ is defined with that same value. Since their defined values are consistent with one another we can therefore construct the "union" φ of the φ_i's as follows:

$$\varphi(\vec{n}) = k \iff \exists i\,(\varphi_i(\vec{n}) = k).$$

(i) This φ is then the required least fixed point of the recursive definition.

To see that it is a fixed point, i.e. $\varphi = \Phi(\varphi)$, first suppose $\varphi(\vec{n})$ is defined with value k. Then by the definition of φ just given, there is an $i > 0$ such that $\varphi_i(\vec{n})$ is defined with value k. But $\varphi_i = \Phi(\varphi_{i-1})$ so $\Phi(\varphi_{i-1})(\vec{n})$ is defined with value k. Therefore by the Monotonicity Principle for Φ, since

φ_{i-1} is a subfunction of φ, $\Phi(\varphi)(\vec{n})$ is defined with value k. Hence φ is a subfunction of $\Phi(\varphi)$.

It remains to show the converse, that $\Phi(\varphi)$ is a subfunction of φ. So suppose $\Phi(\varphi)(\vec{n})$ is defined with value k. Then by the Finite Support Principle, only finitely many defined values of φ are called for in this evaluation. By the definition of φ there must be some i such that φ_i already supplies all of these required values, and so already at stage i we have $\Phi(\varphi_i)(\vec{n}) = \varphi_{i+1}(\vec{n})$ defined with value k. Since φ_{i+1} is a subfunction of φ it follows that $\varphi(\vec{n})$ is defined with value k. Hence $\Phi(\varphi)$ is a subfunction of φ.

To see that φ is the least such fixed point, suppose φ' is any fixed point of Φ. Then $\Phi(\varphi') = \varphi'$ so by the Monotonicity Principle, since φ_0 is a subfunction of φ' it follows that $\Phi(\varphi_0) = \varphi_1$ is a subfunction of $\Phi(\varphi') = \varphi'$. Then again by Monotonicity, $\Phi(\varphi_1) = \varphi_2$ is a subfunction of $\Phi(\varphi') = \varphi'$ etcetera so that for each i, φ_i is a subfunction of φ'. Since φ is the union of the φ_i's it follows that φ itself is a subfunction of φ'. Hence φ is the least fixed point of Φ.

(ii) Finally we have to show that φ is computable if the given functions $\psi_1, \ldots, \psi_m$ are. For this we need the Effective Index Property of the term t, which supplies an elementary function f such that if ψ is computable with program number e then $\Phi(\psi)$ is computable with program number $f(e) = f(e_1, \ldots, e_m, e)$. Thus if u is any fixed program number for the completely undefined function of arity r, $f(u)$ is a program number for $\varphi_1 = \Phi(\varphi_0)$, $f^2(u) = f(f(u))$ is a program number for $\varphi_2 = \Phi(\varphi_1)$, and in general $f^i(u)$ is a program number for φ_i. Therefore in the notation of the Normal Form Theorem,

$$\varphi_i(\vec{n}) \;=\; \varphi^{(r)}_{f^i(u)}(\vec{n})$$

and by the second corollary to the Normal Form Theorem, this is a computable function of i and $\vec{n}$, since $f^i(u)$ is a computable function of i definable (informally) say by a for-loop of the form "**for** $j = 1 \ldots i$ **do** f **od**". Therefore by the earlier equivalences, $\varphi_i(\vec{n})$ is a Σ^0_1-definable function of i and $\vec{n}$, and hence so is φ itself because

$$\varphi(\vec{n}) = m \iff \exists i \, (\, \varphi_i(\vec{n}) = m \,) \, .$$

So φ is computable and this completes the proof. □

NOTE. The above proof works equally well if φ is a vector-valued function. In other words if, instead of defining a single partial function φ, the recursive definition in fact defines a finite list $\vec{\varphi}$ of such functions *simultaneously*. For example, the individual components of the machine state of any register machine at step s are clearly defined by a simultaneous recursive definition, from zero and successor.

4.4. Recursive Programs and Partial Recursive Functions. A *recursive program* is a finite sequence of possibly simultaneous recursive definitions:

$$\begin{aligned}
\vec{\varphi}_0(n_1, \ldots, n_{r_0}) &= t_0(\vec{\varphi}_0; n_1, \ldots, n_{r_0}) \\
\vec{\varphi}_1(n_1, \ldots, n_{r_1}) &= t_1(\vec{\varphi}_0, \vec{\varphi}_1; n_1, \ldots, n_{r_1}) \\
\vec{\varphi}_2(n_1, \ldots, n_{r_2}) &= t_2(\vec{\varphi}_0, \vec{\varphi}_1, \vec{\varphi}_2; n_1, \ldots, n_{r_2})
\end{aligned}$$

$$\vdots$$

$$\vec{\varphi}_k(n_1, \ldots, n_{r_k}) = t_k(\vec{\varphi}_0, \ldots, \vec{\varphi}_{k-1}, \vec{\varphi}_k; n_1, \ldots, n_{r_k}).$$

A partial function is said to be *partial recursive* if it is one of the functions defined by some recursive program as above. A partial recursive function which happens to be totally defined is called simply a *recursive function.*

THEOREM. *A function is partial recursive if and only if it is computable.*

PROOF. The Recursion Theorem tells us immediately that every partial recursive function is computable. For the converse we use the equivalence of computability with μ-recursiveness already established in Section 3. Thus we need only show how to translate any μ-recursive definition into a recursive program:

The constant 0 function is defined by the recursive program

$$\varphi(\vec{n}) \ = \ 0$$

and similarly for the constant 1 function.

The addition function $\varphi(m, n) = m + n$ is defined by the recursive program

$$\varphi(m, n) \ = \ \mathsf{dc}(n, 0, m, \varphi(m, n \dot{-} 1) + 1)$$

and the subtraction function $\varphi(m, n) = m \dot{-} n$ is defined similarly but with the successor function $+1$ replaced by the predecessor $\dot{-}1$. Multiplication is defined recursively from addition in much the same way. Note that in each case the right hand side of the recursive definition is an allowed term.

The composition scheme is a recursive definition as it stands.

Finally, given a recursive program defining ψ, if we add to it the recursive definition:

$$\varphi(\vec{n}, m) \ = \ \mathsf{dc}(\,\psi(\vec{n}, m), 0, m, \varphi(\vec{n}, m + 1)\,)$$

followed by

$$\varphi'(\vec{n}) \ = \ \varphi(\vec{n}, 0)$$

then the computation of $\varphi'(\vec{n})$ proceeds as follows:

$$\begin{aligned} \varphi'(\vec{n}) &= \varphi(\vec{n}, 0) && \\ &= \varphi(\vec{n}, 1) && \text{if } \psi(\vec{n}, 0) \neq 0 \\ &= \varphi(\vec{n}, 2) && \text{if } \psi(\vec{n}, 1) \neq 0 \\ &\ \vdots && \\ &= \varphi(\vec{n}, m) && \text{if } \psi(\vec{n}, m - 1) \neq 0 \\ &= m && \text{if } \psi(\vec{n}, m) = 0 \end{aligned}$$

Thus the recursive program for φ' defines unbounded minimization:

$$\varphi'(\vec{n}) \ = \ \mu m\, (\psi(\vec{n}, m) = 0).$$

This completes the proof. $\square$