CHAPTER 2

Recursion Theory

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 (1963). We base our development on the latter since it affords the quickest route to the results we want to establish.

2.1. Register machines

2.1.1. 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: $[\mathbf{if}\, x = y\ \mathbf{then}\ I_n\ \mathbf{else}\ I_m]$.

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_n or I_m 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 $n, m \leq k$ and I_k means "halt". Notice of course that some programs do not terminate, for example the following one-liner:

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

2.1.2. Program constructs. We develop some shorthand for building up standard sorts of programs.

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

$$\begin{array}{l} x := 0 \\ [\textbf{if } x = y \textbf{ then } I_4 \textbf{ else } I_2] \\ x := x + 1 \\ [\textbf{if } x = x \textbf{ then } I_1 \textbf{ else } I_1], \end{array}$$

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:

$$\begin{array}{l} x := 0 \\ z := 0 \\ [\textbf{if } x = y \textbf{ then } I_8 \textbf{ else } I_3] \\ z := z + 1 \\ [\textbf{if } z = y \textbf{ then } I_8 \textbf{ else } I_5] \\ z := z + 1 \\ x := x + 1 \\ [\textbf{if } z = y \textbf{ then } I_8 \textbf{ else } I_5]. \end{array}$$

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_n **else** I_m]" still operate properly within Q they need to be re-numbered by changing the addresses n, m to $k+n, k+m$ 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

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

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 bypasses Q and halts. If $x \neq y$ then program Q is performed.

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

$$\begin{array}{l} i := 0 \\ [\textbf{if } x = i \textbf{ then } I_{k+4} \textbf{ else } I_2] \\ i := i + 1 \\ \vdots P \\ [\textbf{if } x = i \textbf{ then } I_{k+4} \textbf{ else } I_2] \end{array}$$

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

$$\begin{array}{l} y := 0 \\ [\textbf{if } x = y \textbf{ then } I_{k+3} \textbf{ else } I_2] \\ \vdots P \\ [\textbf{if } x = y \textbf{ then } I_{k+3} \textbf{ else } I_2] \end{array}$$

where again, k is the length of program P and the jump instructions in P must be re-addressed by adding 2. This program keeps on doing P until (if ever) the register x becomes 0; it requires the restriction that the auxiliary register y is not re-assigned new values inside P.

2.1.3. Register machine 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 $P(x_1, \dots, x_k; y)$ to signify that program P has input registers $x_1, \dots, x_k$ and one output register y, which are distinct.

DEFINITION. The program $P(x_1, \dots, x_k; y)$ is said to *compute* the k-ary partial function $\varphi\colon \mathbb{N}^k \to \mathbb{N}$ if, starting with any numerical values $n_1, \dots, n_k$ in the input registers, the program terminates with the number m in the output register if and only if $\varphi(n_1, \dots, n_k)$ 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. "$\mathrm{Add}(x, y; z)$" is the program

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

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

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

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

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 $Q(x_1, \ldots, x_k, z; x)$:

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

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 $\mathrm{Mult}(z, y; x)$ 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 $\mathrm{Add}(v, y; x)$ is replaced by $\mathrm{Mult}(v, y; x)$, 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 $P_j(x_1, \ldots, x_k; y_j)$ computes φ_j for each $j = 1, \ldots, n$ and if $P_0(y_1, \ldots, y_n; y_0)$ computes φ_0, then the program $Q(x_1, \ldots, x_k; y_0)$:

$$P_1(x_1, \ldots, x_k; y_1) \ \textbf{;} \ \ldots \ \textbf{;} \ P_n(x_1, \ldots, x_k; y_n) \ \textbf{;} \ P_0(y_1, \ldots, y_n; y_0)$$

computes the function

$$\psi(x_1, \ldots, x_k) = \varphi_0(\varphi_1(x_1, \ldots, x_k), \ldots, \varphi_n(x_1, \ldots, x_k))$$

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

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

$$\begin{aligned}&y := 0 \ \textbf{;} \ z := 0 \ \textbf{;} \ z := z + 1 \ \textbf{;}\\ &\textbf{while} \ z \neq 0 \ \textbf{do} \ P(x_1, \ldots, x_k, y; z) \ \textbf{;} \ y := y + 1 \ \textbf{od} \ \textbf{;}\\ &z := y \dot{-} 1\end{aligned}$$

computes the function

$$\psi(x_1, \ldots, x_k) \ = \ \mu_y(\varphi(x_1, \ldots, x_k, y) = 0)$$

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

2.2. Elementary functions

2.2.1. 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

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

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

In addition the elementary functions are closed under

Definition 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 $\mu_{k<m+1}(g(\vec{n}, k) = 0)$.

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) *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

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

using $n^n < 2^{2^n}$ (since $n^n < (2^{n-1})^n \le 2^{2^n}$ for $n \ge 3$).

If f is defined from $g_0, g_1, \ldots, g_l$ by composition:

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

where for each $j \le l$ 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. □

2.2.2. 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:

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

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. For instance, it is an easy exercise to show that $\lfloor \frac{n}{m} \rfloor$ is elementary, and then that $n \bmod m$ is elementary. Using this fact we can conclude that the following are elementary as well:

$$\begin{aligned} \mathrm{Prime}(n) &\leftrightarrow 1 < n \wedge \neg \exists_{m<n}(1 < m \wedge n \bmod m = 0), \\ p_n &= \mu_{m<2^{2^n}}(\mathrm{Prime}(m) \wedge n = \sum_{i<m} c_{\mathrm{Prime}}(i)), \end{aligned}$$

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

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

2.2.3. 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) *$\pi(a, b) + b + 2 \le (a + b + 1)^2$ for $a + b \ge 1$, hence $\pi(a, b) < (a + b + 1)^2$,*
(c) *$\pi_1(c), \pi_2(c) \le c$,*
(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:

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

At position $(0,b)$ we clearly have the sum of the lengths of the preceding 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

$$\pi(a,b) = \Bigl(\sum_{i \le a+b} i\Bigr) + a = \frac{1}{2}(a+b)(a+b+1) + a.$$

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

$$\pi_1(c) := \mu_{x \le c} \exists_{y \le c} (\pi(x,y) = c),$$
$$\pi_2(c) := \mu_{y \le c} \exists_{x \le c} (\pi(x,y) = c).$$

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

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

π, π_1 and π_2 are in $\mathcal{E}$ by definition. For $\pi(a,b)$ we have the estimate

$$\pi(a,b) + b + 2 \le (a+b+1)^2 \quad \text{for } a+b \ge 1.$$

This follows with $n := a+b$ from

$$\frac{1}{2}n(n+1) + n + 2 \le (n+1)^2 \quad \text{for } n \ge 1,$$

which is equivalent to $n(n+1) + 2(n+1) \le 2((n+1)^2 - 1)$ and hence to $(n+2)(n+1) \le 2n(n+2)$, which holds for $n \ge 1$. $\square$

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

THEOREM (Gödel's β-function). *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 $c \le 4 \cdot 4^{n(b+n+1)^4}$ such that $\beta(c,i) = a_i$ for all $i < n$.*

PROOF. Let

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

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 $x = a_i$. – Therefore

$$a_i = \mu_{x<b} \exists_{z<d} ((1 + \pi(x, i)a!)z = d).$$

We can now define Gödel's β-function as

$$\beta(c, i) := \mu_{x<\pi_1(c)} \exists_{z<\pi_2(c)} ((1 + \pi(x, i) \cdot \pi_1(c)) \cdot z = \pi_2(c)).$$

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

The above definition of β shows that it is subelementary.

2.2.4. 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 schema*

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

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 theorem 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 μ_c can be bounded by an $\mathcal{E}$ function. However, the theorem on Gödel's β-function 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 μ_c can be bounded by an $\mathcal{E}$ function. □

Remark. Note 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 theorem. If instead, f is defined by bounded product, then proceed similarly. □

2.2.5. 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, $\mathrm{hd}(a) := \pi_2(a \dot{-} 1)$ is the "head" (i.e., rightmost element) and $\mathrm{tl}(a) := \pi_1(a \dot{-} 1)$ is the "tail" of the list. The kth iterate of tl is denoted $\mathrm{tl}^{(k)}$ and since $\mathrm{tl}(a)$ is less than or equal to a, $\mathrm{tl}^{(k)}(a)$ is elementarily definable (by limited recursion).

Thus we can define elementarily the "length" and "decoding" functions:

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

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

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

Furthermore $(a)_i = 0$ when $i \ge \mathrm{lh}(a)$. 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 $\mathrm{lh}(\cdot)$ and $(a)_i$ are subelementary, and so is $\langle n_0, n_1, \dots, n_{k-1}\rangle$ for each fixed k.

LEMMA (Estimate for sequence numbers).

$$(n+1)k \le \langle\underbrace{n,\dots,n}_{k}\rangle < (n+1)^{2^k}.$$

PROOF. We prove a slightly strengthened form of the second estimate:

$$\langle\underbrace{n,\dots,n}_{k}\rangle + n + 1 \le (n+1)^{2^k},$$

by induction on k. For $k=0$ the claim is clear. In the step $k \mapsto k+1$ we have

$$\begin{aligned}\langle\underbrace{n,\dots,n}_{k+1}\rangle + n + 1 &= \pi(\langle\underbrace{n,\dots,n}_{k}\rangle, n) + n + 2\\ &\le (\langle\underbrace{n,\dots,n}_{k}\rangle + n + 1)^2 \quad \text{by the lemma in Section 2.2.3}\\ &\le (n+1)^{2^{k+1}} \quad \text{by induction hypothesis.}\end{aligned}$$

For the first estimate the base case $k=0$ is clear, and in the step we have

$$\begin{aligned}\langle\underbrace{n,\dots,n}_{k+1}\rangle &= \pi(\langle\underbrace{n,\dots,n}_{k}\rangle, n) + 1\\ &\ge \langle\underbrace{n,\dots,n}_{k}\rangle + n + 1\\ &\ge (n+1)(k+1) \quad \text{by induction hypothesis.}\end{aligned}$$ □

Concatenation of sequence numbers $b * a$ is defined thus:

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

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

$$h(b, a, i) = \langle (b)_0, \ldots, (b)_{\mathrm{lh}(b) \dot{-} 1}, (a)_0, \ldots, (a)_{i \dot{-} 1} \rangle \quad \text{for } i \leq \mathrm{lh}(a)$$

it follows from the estimate above that $h(a, b, i) \leq (b + a)^{2^{\mathrm{lh}(b)+i}}$. Thus h is definable by limited recursion from elementary functions and hence is itself elementary. Finally

$$b * a = h(b, a, \mathrm{lh}(a)).$$

2.3. Kleene's normal form theorem

2.3.1. 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, ... 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 $P = I_0, I_1, \ldots, I_{k-1}$ can then be coded by a "program number" or "index" $\sharp P$ thus:

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

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 2.1 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, $\mathrm{comp}(e_0, e_1)$ 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:

$$\mathrm{comp}(e_0, e_1) = e_0 * \langle r(e_0, e_1, 0), r(e_0, e_1, 1), \ldots, r(e_0, e_1, \mathrm{lh}(e_1) \dot{-} 1) \rangle$$

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

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

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 $n_1, \dots, n_r$. Alternative notation for $\varphi_e^{(r)}(n_1, \dots, n_r)$ is $\{e\}(n_1, \dots, n_r)$.

2.3.2. 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 $n_1, \dots, n_r$,*

(a) $\varphi_e^{(r)}(n_1, \dots, n_r)$ *is defined if and only if* $\exists_s T(e, n_1, \dots, n_r, s)$,

(b) $\varphi_e^{(r)}(n_1, \dots, n_r) = U(e, n_1, \dots, n_r, \mu_s T(e, n_1, \dots, n_r, s))$.

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

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

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 i-th one, thus $(e)_i$ is its instruction number.

The "state transition function" $\mathrm{tr}\colon \mathbb{N} \to \mathbb{N}$ 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 $\mathrm{tr}(x)$ is therefore as follows:

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

where

(i) If $(e)_i = \langle 0, j \rangle$ where $j \leq l$ then $i' = i + 1$, $m'_j = 0$, and all other registers remain unchanged, i.e., $m'_k = m_k$ for $k \neq j$.
(ii) 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.
(iii) If $(e)_i = \langle 2, j_0, j_1, i_0, i_1 \rangle$ where $j_0, j_1 \leq l$ and $i_0, i_1 \leq \mathrm{lh}(e)$ 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 $j \leq l$.
(iv) Otherwise, if e is not a program number, or if it refers to a register v_k with $l < k$, or if $\mathrm{lh}(e) \leq i$, then $\mathrm{tr}(x)$ 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" $\mathrm{state}(e, \vec{n}, s)$ is also *elementary* because it can be defined by iterating the transition function by limited recursion on s as follows:

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

where for the bounding function h we can take

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

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 $(m+1)^{2^i}$, by the estimate Lemma in Section 2.2.5.

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

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

Clearly T is elementary and

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

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

$$U(e, \vec{n}, s) := (\mathrm{state}(e, \vec{n}, s))_2$$

then U is also elementary and

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

□

2.3.3. 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 $\vec{n} = n_1, \dots, n_r$,

$$R(\vec{n}\,) \leftrightarrow \exists_{k_1,\dots k_l} E(\vec{n}, k_1, \dots, k_l).$$

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

$$R(\vec{n}\,) \leftrightarrow \exists_m (f(m) = \langle \vec{n} \rangle)$$

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 i-th 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 Section 2.2.3 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 the theorem in Section 2.2.4 one sees that the μ-recursive functions are closed under (unlimited) primitive recursive definitions: $f(\vec{m}, 0) = g(\vec{m})$, $f(\vec{m}, n+1) = h(n, f(\vec{m}, n), \vec{m})$. Thus one can μ-recursively define bounded sums and bounded products, and hence all elementary functions. $\square$

2.3.4. 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 Section 2.1 (on program constructs).

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

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

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

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

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 2.1 show how the definition of any μ-recursive function translates automatically into a while program. Finally, Section 2.1 shows how to implement any while program on a register machine. $\square$

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

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

PROOF. Immediate from the normal form. $\square$

LEMMA. *Let R and $\bar{R}$ be disjoint inhabited relations with $\forall_{\vec{n}}(R\vec{n} \vee \bar{R}\vec{n})$. Then R is computable if and only if both R and $\bar{R}$ are Σ^0_1-definable.*

PROOF. We assume (for simplicity) that R and $\bar{R}$ are unary.
“$\to$”. By the theorem above every computable relation is Σ^0_1-definable, and with R clearly $\bar{R}$ is computable.
“$\leftarrow$”. Let $f, g \in \mathcal{E}$ enumerate R and $\bar{R}$, respectively. Then

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

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

2.3.5. 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

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

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.