CHAPTER 5

EFFECTIVE COMPUTABILITY

1. Markov Algorithms

A function $f(x_1, \ldots, x_n)$ is thought of as being effectively computable if there is a mechanical procedure for determining the value $f(k_1, \ldots, k_n)$ when the arguments $k_1, \ldots, k_n$ are given. The phrase "mechanical procedure" is not at all precise; what we mean is a process which requires no ingenuity for its performance. An obvious example is the addition of two integers expressed in decimal notation. Another well-known case is the Euclidean algorithm for obtaining the greatest common divisor of two integers. In these two examples, it seems intuitively clear that the given functions are effectively computable. This is generally the case when an effective procedure has already been discovered. However, more and more in mathematics, we are faced with the task of showing that there is no effectively computable function of a certain kind or that there is no effective procedure for solving a large class of problems. To illustrate, we can cite on the one hand the well-known effective way of determining whether or not any given polynomial f(x) in one variable with integral coefficients has an integral root.† On the other hand, the famous Tenth Problem of Hilbert asked whether there is an effective procedure for determining whether or not any given polynomial $f(x_1, \ldots, x_n)$ with integral coefficients, in any finite number of variables, has integral roots. This problem recently has been solved by Matiyasevich [1970], whose proof was the culmination of previous work by M. Davis, J. Robinson, and H. Putnam. (Cf. Davis [1973] for a complete exposition.) If we attempt to prove that there is no effective procedure or operation of a certain kind, it is apparent that we have to give a precise, mathematical definition of the notion of effective computability. The situation is analogous to that which prevailed in mathematics before notions like continuity, curve, surface, and area were explicated.

Any particular problem of a general class of problems can be formulated as an expression of some language. Any expression of a language can be considered as a sequence of symbols of that language, provided that the blank which is

†If $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, then any integral solution of $f(x) = 0$ must be a divisor of a_0. Each of the finite number of divisors of a_0 can be tested to see whether it satisfies $f(x) = 0$.

usually used to separate words is assumed to be a symbol in its own right. By an *alphabet* we mean a non-empty finite set of symbols. Most natural languages use only a finite number of symbols, and, for our purposes, it also suffices to treat only such alphabets. (Indeed, anything that can be done with an infinite alphabet $a_1, a_2, \ldots$ can be accomplished with a two-symbol alphabet $\{b, c\}$, if we let $\underbrace{bcc \ldots ccb}_{n \text{ times}}$ play the role of a_n.) For uniformity, we assume that the symbols of all alphabets are taken from the denumerable sequence $S_0, S_1, S_2, \ldots$, though, sometimes, for convenience, we shall use other letters.

A *word* in an alphabet A is any finite sequence of symbols of A. The empty sequence of symbols is called the *empty word*, and is denoted by Λ. If P denotes a word $S_{j_1} \ldots S_{j_k}$ and Q denotes a word $S_{r_1} \ldots S_{r_m}$, then we use PQ to denote the juxtaposition $S_{j_1} \ldots S_{j_k} S_{r_1} \ldots S_{r_m}$ of the two words. In particular, $P\Lambda = \Lambda P = P$; also, $(P_1P_2)P_3 = P_1(P_2P_3)$.

An alphabet A is an *extension* of an alphabet B if and only if $B \subseteq A$. If A is an extension of B, any word of B is a word of A.

By an *algorithm in* an alphabet A, we mean an effectively computable function $\mathfrak{A}$ whose domain is a subset of the set of words of A and the values of which are also words in A. If P is a word in A, $\mathfrak{A}$ is said to be *applicable* to P if P is in the domain of $\mathfrak{A}$; if $\mathfrak{A}$ is applicable to P, we denote its value by $\mathfrak{A}(P)$. By an *algorithm over* an alphabet A we mean an algorithm $\mathfrak{A}$ in an extension B of A. Of course, the notion of algorithm is as hazy as that of effectively computable function.

Most familiar algorithms can be broken down into a few simple steps. Starting from this observation, and following Markov [1954], we select a particularly simple operation, substitution of one word for another, as the basic unit from which algorithms are to be constructed. To this end, if P and Q are words of an alphabet A, then we call the expressions $P \to Q$ and $P \to \cdot Q$ *productions* in the alphabet A. We assume here that "$\to$" and the dot "$\cdot$" are not symbols of A. Notice that P or Q can be the empty word. $P \to Q$ is called a *simple* production, while $P \to \cdot Q$ is a *terminal* production. Let us use $P \to (\cdot)Q$ to denote either $P \to Q$ or $P \to \cdot Q$. A finite list of productions in A

$$\begin{aligned} P_1 &\to (\cdot)Q_1 \\ P_2 &\to (\cdot)Q_2 \\ &\vdots \\ P_r &\to (\cdot)Q_r \end{aligned}$$

is called an *algorithm schema* and determines the following algorithm $\mathfrak{A}$ in A. As a preliminary definition, we say that a word T *occurs* in a word Q if there are words U, V (possibly empty) such that $Q = UTV$. Now, given a word P in A: (1) We write $\mathfrak{A}$: $P \sqsupset$ if none of the words $P_1, \ldots, P_r$ occurs in P. (2) Otherwise, if m is the least integer, with $1 \leqslant m \leqslant r$, such that P_m occurs in P, and if R is the word which results from replacing the left-most occurrence of P_m in P by Q_m,

then we write

(a) $$\mathfrak{A} : P \vdash R$$

if $P_m \to (\cdot)Q_m$ is simple (and we say that $\mathfrak{A}$ simply transforms P into R);

(b) $$\mathfrak{A} : P \vdash \cdot R$$

if $P_m \to (\cdot)Q_m$ is terminal (and we say that $\mathfrak{A}$ terminally transforms P into R).

We then define $\mathfrak{A} : P \models R$ to mean that there is a sequence $R_0, R_1, \ldots, R_k$ such that $P = R_0$; $R = R_k$; if $0 \leqslant j \leqslant k - 2$, $\mathfrak{A} : R_j \vdash R_{j+1}$; and either $\mathfrak{A} : R_{k-1} \vdash R_k$ or $\mathfrak{A} : R_{k-1} \vdash \cdot R_k$. (In the second case, we write $\mathfrak{A} : P \models \cdot R$.) We set $\mathfrak{A}(P) = R$ if and only if either $\mathfrak{A} : P \models \cdot R$, or $\mathfrak{A} : P \models R$ and $\mathfrak{A} : R \urcorner$. The algorithm thus defined is called a *normal* algorithm (or Markov algorithm) in the alphabet A.

The action of $\mathfrak{A}$ can be described as follows: given a word P, we find the first production $P_m \to (\cdot)Q_m$ in the schema such that P_m occurs in P. We then substitute Q_m for the left-most occurrence of P_m in P. Let R_1 be the new word obtained in this way. If $P_m \to (\cdot)Q_m$ is a terminal production, the process stops and the value of the algorithm is R_1. If $P_m \to (\cdot)Q_m$ is simple, then we apply the same process to R_1 as was just applied to P, and so on. If we ever obtain a word R_i such that $\mathfrak{A} : R_i \urcorner$, i.e., no P_m occurs in R_i for $1 \leqslant m \leqslant r$, then the process stops and the value of $\mathfrak{A}$ is R_i. It is possible that the process just described never stops. In that case, $\mathfrak{A}$ is not applicable to the given word P.

Our exposition of the theory of normal algorithms will closely follow that of Markov [1954].

Examples.

1. Let A be the alphabet $\{b, c\}$. Consider the schema

$$b \to \cdot\Lambda$$
$$c \to c$$

The normal algorithm $\mathfrak{A}$ defined by this schema transforms any word containing at least one occurrence of b into the word obtained by erasing the left-most occurrence of b. $\mathfrak{A}$ transforms the empty word Λ into itself. $\mathfrak{A}$ is not applicable to any non-empty word not containing b.

2. Let A be the alphabet $\{a_0, a_1, \ldots, a_n\}$. Consider the schema

$$a_0 \to \Lambda$$
$$a_1 \to \Lambda$$
$$\vdots$$
$$a_n \to \Lambda$$

We can abbreviate this schema as follows:

$$\xi \to \Lambda \quad (\xi \text{ in A})$$

(Whenever we use such abbreviations, the productions intended may be listed in any order.) The corresponding normal algorithm transforms every word into the

empty word. For example, $\mathfrak{A} : a_1a_2a_1a_3a_0 \vdash a_1a_2a_1a_3 \vdash a_2a_1a_3 \vdash a_2a_3 \vdash a_3 \vdash \Lambda$ and $\mathfrak{A} : \Lambda\urcorner$. Hence $\mathfrak{A}(a_1a_2a_1a_3a_0) = \Lambda$.

3. Let A be an alphabet containing the symbol S_1, which we shall abbreviate 1. For natural numbers n, we define $\bar{n}$ inductively as follows: $\bar{0} = 1$ and $\overline{n+1} = \bar{n}1$. Thus, $\bar{1} = 11$, $\bar{2} = 111$, etc. The words $\bar{n}$ are called numerals. Now consider the schema $\Lambda \to \cdot\, 1$, defining a normal algorithm $\mathfrak{A}$. For any word P in A, $\mathfrak{A}(P) = 1P$ · † In particular, for every natural number n, $\mathfrak{A}(\bar{n}) = \overline{n+1}$.

4. Let A be an arbitrary alphabet $\{a_0, a_1, \ldots, a_n\}$. Given a word $P = a_{j_0}a_{j_1} \ldots a_{j_k}$, let $\check{P} = a_{j_k} \ldots a_{j_1}a_{j_0}$ be the *inverse* of P. We seek a normal algorithm $\mathfrak{A}$ such that $\mathfrak{A}(P) = \check{P}$. Consider the following (abbreviated) algorithm schema in the alphabet $B = A \cup \{\alpha, \beta\}$.

(a)	$\alpha\alpha \to \beta$	
(b)	$\beta\xi \to \xi\beta$	(ξ in A)
(c)	$\beta\alpha \to \beta$	
(d)	$\beta \to \cdot\Lambda$	
(e)	$\alpha\eta\xi \to \xi\alpha\eta$	(ξ, η in A)
(f)	$\Lambda \to \alpha$	

This determines a normal algorithm $\mathfrak{A}$ in B. Let $P = a_{j_0}a_{j_1} \ldots a_{j_k}$ be any word in A. Then, $\mathfrak{A} : P \vdash \alpha P$ by production (f); $\alpha P \vdash a_{j_1}\alpha a_{j_0}a_{j_2} \ldots a_{j_k} \vdash a_{j_1}a_{j_2}\alpha a_{j_0}a_{j_3} \ldots a_{j_k} \ldots \vdash a_{j_1}a_{j_2} \ldots a_{j_k}\alpha a_{j_0}$ by production (e). Thus, $\mathfrak{A} : P \vdash a_{j_1}a_{j_2} \ldots a_{j_k}\alpha a_{j_0}$. Then, by production (f), $\mathfrak{A} : P \vdash \alpha a_{j_1}a_{j_2} \ldots a_{j_k}\alpha a_{j_0}$. Applying, as before, production (e), $\mathfrak{A} : P \vdash a_{j_2}a_{j_3} \ldots a_{j_k}\alpha a_{j_1}\alpha a_{j_0}$. Iterating this process, we obtain $\mathfrak{A} : P \vdash \alpha a_{j_k}\alpha a_{j_{k-1}}\alpha \ldots \alpha a_{j_1}\alpha a_{j_0}$. Then, by production (f), $\mathfrak{A} : P \vdash \alpha\alpha a_{j_k}\alpha a_{j_{k-1}}\alpha \ldots \alpha a_{j_1}\alpha a_{j_0}$, and, by production (a), $\mathfrak{A} : P \vdash \beta a_{j_k}\alpha a_{j_{k-1}}\alpha \ldots \alpha a_{j_1}\alpha a_{j_0}$; applying productions (b) and (c), and, finally, (d), we arrive at $\mathfrak{A} : P \vdash \cdot\,\check{P}$. Thus, $\mathfrak{A}$ is a normal algorithm over A which inverts every word of A.‡

EXERCISES

5.1. Let A be an alphabet. Describe the action of the normal algorithms given by the following schemas.

(a) Let Q be a fixed word in A, and let the algorithm schema be: $\Lambda \to \cdot\, Q$.

(b) Let Q be a fixed word in A, and let α be a symbol not in A. Let $B = A \cup \{\alpha\}$. Consider the schema

$$\begin{aligned} \alpha\xi &\to \xi\alpha \qquad (\xi \text{ in A}) \\ \alpha &\to \cdot\, Q \\ \Lambda &\to \alpha \end{aligned}$$

†To see this, observe that Λ occurs at the beginning of any word P, since $P = \Lambda P$.

‡The distinction between a normal algorithm in A and a normal algorithm over A is important. A normal algorithm in A uses only symbols of A, while a normal algorithm over A may employ additional symbols not in A. Every normal algorithm in A is a normal algorithm over A, but there are algorithms in A which are determined by normal algorithms over A but which are not normal algorithms in A (cf. Exercise 5.9(d), p. 240).

(c) Let Q be a fixed word in A. Take the schema

$$\xi \to \Lambda \qquad (\xi \text{ in A})$$
$$\Lambda \to \cdot Q$$

(d) Let $B = A \cup \{1\}$. Consider the schema

$$\xi \to 1 \qquad (\xi \text{ in } A - \{1\})$$
$$\Lambda \to \cdot 1$$

5.2. Let A be an alphabet not containing the symbols α, β, γ. Let $B = A \cup \{\alpha\}$ and $C = A \cup \{\alpha, \beta, \gamma\}$.

(a) Construct a normal algorithm $\mathfrak{A}$ in B such that $\mathfrak{A}(\Lambda) = \Lambda$ and $\mathfrak{A}(\xi P) = P$ for any symbol ξ in A and any word P in A. Thus, $\mathfrak{A}$ erases the first letter of any non-empty word in A.

(b) Construct a normal algorithm $\mathfrak{D}$ in B such that $\mathfrak{D}(\Lambda) = \Lambda$ and $\mathfrak{D}(P\xi) = P$ for any symbol ξ in A and word P in A. Thus, $\mathfrak{D}$ erases the last letter of any non-empty word in A.

(c) Construct a normal algorithm $\mathfrak{C}$ in B such that $\mathfrak{C}(P) = \Lambda$ if P contains exactly two occurrences of α, and $\mathfrak{C}(P)$ is defined and $\neq \Lambda$ in all other cases.

(d) Construct a normal algorithm $\mathfrak{B}$ in C such that, for any word P of A, $\mathfrak{B}(P) = PP$.

5.3. Let A and B be alphabets, and let α be a symbol in neither A nor B. For certain symbols $a_1, \ldots, a_k$ in A, let $Q_1, \ldots, Q_k$ be corresponding words in B. Consider the algorithm which associates with each word of A the word $\mathrm{Sub}_{Q_1, \ldots, Q_k}^{a_1, \ldots, a_k}(P)$ obtained by simultaneous substitution of each Q_i for a_i ($i = 1, \ldots, k$). Show that this is given by a normal algorithm in $A \cup B \cup \{\alpha\}$.

5.4. Let $H = \{1\}$ and $M = \{1, *\}$. Every natural number n is represented by its numeral $\bar{n}$, which is a word in H. We represent every k-tuple $(n_1, n_2, \ldots, n_k)$ of natural numbers by the word $\bar{n}_1 * \bar{n}_2 * \ldots * \bar{n}_k$ in M. We shall denote this word by $\overline{(n_1, \ldots, n_k)}$. For example, $\overline{(3, 1, 2)}$ is $1111 * 11 * 111$.

(a) Show that the schema

$$* \to *$$
$$\alpha 11 \to \alpha 1$$
$$\alpha 1 \to \cdot 1$$
$$\Lambda \to \alpha$$

defines a normal algorithm $\mathfrak{A}_Z$ over M such that $\mathfrak{A}_Z(\bar{n}) = \bar{0}$ for any n, and $\mathfrak{A}_Z$ is applicable only to numerals in M.

(b) Show that the schema

$$* \to *$$
$$\alpha 1 \to \cdot 11$$
$$\Lambda \to \alpha$$

defines a normal algorithm $\mathfrak{A}_Z$ over M such that $\mathfrak{A}_N(\bar{n}) = \overline{n+1}$ for all n, and $\mathfrak{A}_N$ is applicable only to numerals in M.

(c) Let $\alpha_1, \ldots, \alpha_{2k}$ be symbols not in M. Let $1 \leq j \leq k$. Let $\mathfrak{S}_i$ be the list

$$\alpha_{2i-1} * \to \alpha_{2i-1} *$$
$$\alpha_{2i-1} 1 \to \alpha_{2i} 1$$
$$\alpha_{2i} 1 \to \alpha_{2i}$$
$$\alpha_{2i} * \to \alpha_{2i+1}$$

If $1 < j < k$, consider the algorithm schema

$\mathfrak{S}_1$
$\vdots$
$\mathfrak{S}_{j-1}$
$\alpha_{2j-1} * \to \alpha_{2j-1} *$
$\alpha_{2j-1} 1 \to \alpha_{2j} 1$
$\alpha_{2j} 1 \to 1 \alpha_{2j}$
$\alpha_{2j} * \to \alpha_{2j+1}$
$\mathfrak{S}_{j+1}$
$\vdots$
$\mathfrak{S}_{k-1}$
$\alpha_{2k-1} * \to \alpha_{2k-1} *$
$\alpha_{2k-1} 1 \to \alpha_{2k} 1$
$\alpha_{2k} 1 \to \alpha_{2k}$
$\alpha_{2k} * \to \alpha_{2k} *$
$\alpha_{2k} \to \cdot \Lambda$
$\Lambda \to \alpha_1$

If $j = 1$, consider the schema

$\alpha_1 * \to \alpha_1 *$
$\alpha_1 1 \to \alpha_2 1$
$\alpha_2 1 \to 1 \alpha_2$
$\alpha_2 * \to \alpha_3$
$\mathfrak{S}_2$
$\vdots$
$\mathfrak{S}_{k-1}$
$\alpha_{2k-1} * \to \alpha_{2k-1} *$
$\alpha_{2k-1} 1 \to \alpha_{2k} 1$
$\alpha_{2k} 1 \to \alpha_{2k}$
$\alpha_{2k} * \to \alpha_{2k} *$
$\alpha_{2k} \to \cdot \Lambda$
$\Lambda \to \alpha_1$

If $j = k$, consider the schema

$\mathfrak{S}_1$
$\vdots$
$\mathfrak{S}_{k-1}$
$\alpha_{2k-1} * \to \alpha_{2k-1} *$
$\alpha_{2k-1} 1 \to \alpha_{2k} 1$
$\alpha_{2k} 1 \to 1 \alpha_{2k}$
$\alpha_{2k} * \to \alpha_{2k} *$
$\alpha_{2k} * \to \alpha_{2k} *$
$\alpha_{2k} \to \cdot \Lambda$
$\Lambda \to \alpha_1$

Show that the corresponding normal algorithm $\mathfrak{A}_j^k$ is such that $\mathfrak{A}_j^k(\overline{(n_1, \ldots, n_k)}) = \overline{n_j}$; and $\mathfrak{A}_j^k$ is applicable only to words of the form $\overline{(n_1, \ldots, n_k)}$,

(d) Construct a schema for a normal algorithm in M transforming $\overline{(n_1, n_2)}$ into $\overline{|n_1 - n_2|}$.

(e) Construct a normal algorithm in M for addition.

(f) Construct a normal algorithm over M for multiplication.

Given algorithms $\mathfrak{A}$ and $\mathfrak{B}$ and a word P, we write $\mathfrak{A}(P) \approx \mathfrak{B}(P)$ if and only if either $\mathfrak{A}$ and $\mathfrak{B}$ are both applicable to P and $\mathfrak{A}(P) = \mathfrak{B}(P)$ or neither $\mathfrak{A}$ nor $\mathfrak{B}$ is applicable to P. More generally, if C and D are expressions, then $C \approx D$ is to hold if and only if neither C nor D is defined or both C and D are defined and denote the same object. If $\mathfrak{A}$ and $\mathfrak{B}$ are algorithms over an alphabet A, then we say that $\mathfrak{A}$ and $\mathfrak{B}$ are *fully equivalent* relative to A if and only if $\mathfrak{A}(P) \approx \mathfrak{B}(P)$ for every word P in A; we say that $\mathfrak{A}$ and $\mathfrak{B}$ are *equivalent* relative to A if and only if, for any word P in A, whenever $\mathfrak{A}(P)$ or $\mathfrak{B}(P)$ exists and is in A, then $\mathfrak{A}(P) \approx \mathfrak{B}(P)$.

Let M be the alphabet $\{1, *\}$, as in Exercise 5.4 above; let ω be the set of natural numbers. Given a partial effectively computable number-theoretic

function φ of k arguments, i.e., a function from a subset of ω^k into ω, we denote by $\mathfrak{B}_\varphi$ the corresponding algorithm in M; that is, $\mathfrak{B}_\varphi(\overline{(n_1, \ldots, n_k)}) = \overline{\varphi(n_1, \ldots, n_k)}$ whenever either of the two sides of the equation is defined; $\mathfrak{B}_\varphi$ is assumed to be inapplicable to words not of the form $\overline{(n_1, \ldots, n_k)}$. The function φ is said to be *partially Markov-computable* if and only if there is a normal algorithm $\mathfrak{A}$ over M which is fully equivalent to $\mathfrak{B}_\varphi$ relative to M.† If the function φ is total, i.e., if φ is defined for all k-tuples of natural numbers, and if φ is partially Markov-computable, then φ is said to be *Markov-computable*.

Let us generalize the notion of recursive function (cf. p 138). A partial function φ of k arguments is called *partial recursive* if and only if φ can be obtained from the initial functions Z (zero function), U_j^n (projection functions), and N (successor function) by means of substitution, recursion, and the unrestricted μ-operator. (We say that ψ comes from τ by means of the unrestricted μ-operator if and only if $\psi(x_1, \ldots, x_n) = \mu y(\tau(x_1, \ldots, x_n, y) = 0)$. More precisely, $\mu y(\tau(x_1, \ldots, x_n, y) = 0)$ is the least number k (if such exists) such that, if $0 \leqslant i < k$, $\tau(x_1, \ldots, x_n, i)$ exists and is not 0, and $\tau(x_1, \ldots, x_n, k) = 0$. Notice that ψ may not be defined for certain n-tuples; in particular, for those n-tuples $(x_1, \ldots, x_n)$ for which there is no y such that $\tau(x_1, \ldots, x_n, y) = 0$.) Clearly, every recursive function is partial recursive. The assertion that every total partial recursive function is recursive is true, but not at all obvious, and will be proved later. We shall show that the partial recursive functions coincide with the partially Markov-computable functions and that the recursive functions are identical with the Markov-computable functions.

A normal algorithm is said to be *closed* if and only if one of the productions in its schema has the form $\Lambda \to \cdot\, Q$. Such an algorithm can only end terminally, i.e., by an application of a terminal production. Given an arbitrary normal algorithm $\mathfrak{A}$, add on at the end of the schema for $\mathfrak{A}$ the new production $\Lambda \to \cdot\, \Lambda$, and denote by $\mathfrak{A} \cdot$ the normal algorithm determined by this enlarged schema. $\mathfrak{A} \cdot$ is closed, and $\mathfrak{A} \cdot$ is fully equivalent to $\mathfrak{A}$ relative to the alphabet of $\mathfrak{A}$.

Let us show now that the composition of two normal algorithms is again a normal algorithm. Let $\mathfrak{A}$ and $\mathfrak{B}$ be normal algorithms in an alphabet A. For each symbol b in A, form a new symbol $\bar{b}$, called the correlate of b. Let $\bar{A}$ be the alphabet consisting of the correlates of the symbols of A. Let α and β be two symbols not in $A \cup \bar{A}$. Let $\mathfrak{S}_\mathfrak{A}$ be the schema of $\mathfrak{A} \cdot$ except that the terminal dot in terminal productions is replaced by α. Let $\mathfrak{S}_\mathfrak{B}$ be the schema of $\mathfrak{B} \cdot$ except that every symbol is replaced by its correlate, every terminal dot by β, productions of the form $\Lambda \to Q$ are replaced by $\alpha \to \alpha Q$, and productions $\Lambda \to \cdot\, Q$ are

†In this and in all other definitions in this chapter, the existential quantifier "there is" is meant in the ordinary, "classical" sense. When we assert that there exists an object of a certain kind, we do not necessarily imply that any human being has found or ever will find such an object. Thus, a function φ may be partially Markov-computable without our ever knowing it to be so.

replaced by $\alpha \to \alpha\beta Q$. Consider the abbreviated schema

$$a\alpha \to \alpha a \qquad (a \text{ in A})$$
$$\alpha a \to \alpha \bar{a} \qquad (a \text{ in A})$$
$$\bar{a}b \to \bar{a}\bar{b} \qquad (a, b \text{ in A})$$
$$\bar{a}\beta \to \beta\bar{a} \qquad (a \text{ in A})$$
$$\beta\bar{a} \to \beta a \qquad (a \text{ in A})$$
$$a\bar{b} \to ab \qquad (a, b \text{ in A})$$
$$\alpha\beta \to \cdot\Lambda$$
$$\mathfrak{S}_{\mathfrak{B}}$$
$$\mathfrak{S}_{\mathfrak{A}}$$

This schema determines a normal algorithm $\mathfrak{C}$ over A such that $\mathfrak{C}(P) \approx \mathfrak{B}(\mathfrak{A}(P))$ for any word P in A. (Exercise.) $\mathfrak{C}$ is called the *composition* of $\mathfrak{A}$ and $\mathfrak{B}$, and is denoted $\mathfrak{B} \circ \mathfrak{A}$. In general, by $\mathfrak{A}_n \circ \ldots \circ \mathfrak{A}_1$, we mean $\mathfrak{A}_n \circ (\ldots \circ (\mathfrak{A}_3 \circ (\mathfrak{A}_2 \circ \mathfrak{A}_1)) \ldots)$.

Let $\mathfrak{D}$ be a normal algorithm in an alphabet A, and let B be an extension of A. If we take a schema for $\mathfrak{D}$ and prefix to it the productions $b \to b$ for each symbol b in B − A, then the new schema determines a normal algorithm $\mathfrak{D}_B$ in B such that $\mathfrak{D}_B(P) \approx \mathfrak{D}(P)$ for every word P in A, and $\mathfrak{D}_B$ is not applicable to any word in B containing any symbol of B − A. $\mathfrak{D}_B$ is fully equivalent to $\mathfrak{D}$ relative to A and is called the *propagation* of $\mathfrak{D}$ onto B.

Assume that $\mathfrak{A}$ is a normal algorithm in an alphabet A_1 and $\mathfrak{B}$ is a normal algorithm in an alphabet A_2. Let $A = A_1 \cup A_2$. Let $\mathfrak{A}_A$ and $\mathfrak{B}_A$ be the propagations of $\mathfrak{A}$ and $\mathfrak{B}$, respectively, onto A. Then the composition $\mathfrak{C}$ of $\mathfrak{A}_A$ and $\mathfrak{B}_A$ is called the *normal composition* of $\mathfrak{A}$ and $\mathfrak{B}$, and is denoted by $\mathfrak{B} \circ \mathfrak{A}$. (When $A_1 = A_2$, the normal composition of $\mathfrak{A}$ and $\mathfrak{B}$ is identical with the composition of $\mathfrak{A}$ and $\mathfrak{B}$; hence the notation $\mathfrak{B} \circ \mathfrak{A}$ is unambiguous.) $\mathfrak{C}$ is a normal algorithm over A such that $\mathfrak{C}(P) \approx \mathfrak{B}(\mathfrak{A}(P))$ for any word P in A_1, and $\mathfrak{C}$ is applicable only to those words P of A such that P is a word of A_1, $\mathfrak{A}$ is applicable to P, and $\mathfrak{B}$ is applicable to $\mathfrak{A}(P)$.

Let an alphabet B be an extension of an alphabet A. Given a word P in B, the *projection* P^A of P on A is the word obtained by erasing in P all symbols in B − A. The abbreviated schema $\xi \to \Lambda$ (ξ in B − A) determines the *projection algorithm* $\mathfrak{B}$ of B on A, i.e., $\mathfrak{B}(P) = P^A$ for all words P in B.

Let A and C be alphabets without any symbols in common, and let $B = A \cup C$. Then the abbreviated schema $ca \to ac$ (a in A; c in C) determines a normal algorithm $\mathfrak{L}_{A,C}$ in B such that, for any word P in B, $\mathfrak{L}_{A,C}(P) = P^A P^C$.

Given a normal algorithm $\mathfrak{A}$ in an alphabet A, and an extension B of A, then the *natural extension* $\mathfrak{B}$ of the algorithm $\mathfrak{A}$ to B is the normal algorithm in B

determined by the given schema for $\mathfrak{A}$. Clearly, $\mathfrak{B}(P) \approx \mathfrak{A}(P)$ for any word P in A, and, in addition, if P is a word in A and Q is a word in B − A, then $\mathfrak{B}(PQ) \approx \mathfrak{A}(P)Q$. Notice that the natural extension of $\mathfrak{A}$ to B generally is different from the propagation of $\mathfrak{A}$ onto B, since the latter is not applicable to any word containing symbols of B − A.

PROPOSITION 5.1. *Let* $\mathfrak{A}_1, \ldots, \mathfrak{A}_k$ *be normal algorithms, and let* A *be the union of their alphabets. Then there is a normal algorithm* $\mathfrak{B}$ *over* A, *called the* juxtaposition *of the algorithms* $\mathfrak{A}_1, \ldots, \mathfrak{A}_k$ *such that*

$$\mathfrak{B}(P) \approx \mathfrak{A}_1\#(P)\mathfrak{A}_2\#(P) \ldots \mathfrak{A}_k\#(P) \text{ for any word } P \text{ in A},$$

where $\mathfrak{A}_i\#$ *is the natural extension of* $\mathfrak{A}_i$ to A.

PROOF. By induction on k. Clearly, it suffices to prove the result for k = 2. For each symbol a in A, introduce a new symbol $\bar{a}$. Let $\overline{\mathrm{A}}$ be the alphabet made up of these new symbols, and let $\mathrm{B} = \mathrm{A} \cup \overline{\mathrm{A}}$. Let $\overline{\mathfrak{A}}_1$ be the normal algorithm in $\overline{\mathrm{A}}$ corresponding to the algorithm $\mathfrak{A}_1\#$ in A. (The schema for $\overline{\mathfrak{A}}_1$ is obtained from that for $\mathfrak{A}_1$ by replacing each symbol a by $\bar{a}$.) Let $\overline{\mathfrak{A}}_1\#$ and $\mathfrak{A}_2\#$ be the natural extensions, respectively, of $\overline{\mathfrak{A}}_1$ and $\mathfrak{A}_2$ to B. Let $\mathrm{A} = \{a_1, \ldots, a_n\}$. By Exercise 5.3 (p. 225), there exist normal algorithms $\mathfrak{W}_1 = \mathrm{Sub}^{a_1, a_2, \ldots, a_n}_{a_1\bar{a}_1, a_2\bar{a}_2, \ldots, a_n\bar{a}_n}$ and $\mathfrak{W}_2 = \mathrm{Sub}^{\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_n}_{a_1, a_2, \ldots, a_n}$ over B such that $\mathfrak{W}_1$ simultaneously substitutes $a_1\bar{a}_1$ for a_1, $a_2\bar{a}_2$ for $a_2, \ldots,$ and $\mathfrak{W}_2$ substitutes a_1 for $\bar{a}_1$, a_2 for $\bar{a}_2, \ldots$. We also have normal algorithms $\mathfrak{L}_{\mathrm{A}, \overline{\mathrm{A}}}$ and $\mathfrak{L}_{\overline{\mathrm{A}}, \mathrm{A}}$ such that $\mathfrak{L}_{\mathrm{A}, \overline{\mathrm{A}}}(P) = P^{\mathrm{A}}P^{\overline{\mathrm{A}}}$ and $\mathfrak{L}_{\overline{\mathrm{A}}, \mathrm{A}}(P) = P^{\overline{\mathrm{A}}}P^{\mathrm{A}}$. Then, take the normal composition $\mathfrak{B} = \mathfrak{W}_2 \circ \overline{\mathfrak{A}}_1\# \circ \mathfrak{L}_{\overline{\mathrm{A}}, \mathrm{A}} \circ \mathfrak{A}_2\# \circ \mathfrak{L}_{\mathrm{A}, \overline{\mathrm{A}}} \circ \mathfrak{W}_1$. It is easy to verify that $\mathfrak{B}(P) \approx \mathfrak{A}_1\#(P)\mathfrak{A}_2\#(P)$ for any word P in A.

COROLLARY 5.2. *Let* $\mathfrak{A}_1, \ldots, \mathfrak{A}_k$ *be normal algorithms; let* A *be the union of their alphabets. Then there is a normal algorithm* $\mathfrak{B}$ *over* $\mathrm{A} \cup \{ * \}$ *such that* $\mathfrak{B}(P) \approx \mathfrak{A}_1\#(P) * \mathfrak{A}_2\#(P) * \ldots * \mathfrak{A}_k\#(P)$ *for any word* P *in* A, *where* $\mathfrak{A}_i\#$ *is the natural extension of* $\mathfrak{A}_i$ *to* A. (*Hence,* $\mathfrak{B}(P) \approx \mathfrak{A}_1(P) * \mathfrak{A}_2(P) * \ldots * \mathfrak{A}_k(P)$ *for any word* P *common to the alphabets of all the* $\mathfrak{A}_i$'s.)

PROOF. There is a normal algorithm $\mathfrak{D}$ in $\mathrm{A} \cup \{ * \}$ such that $\mathfrak{D}(P) = *$ for any word P in $\mathrm{A} \cup \{ * \}$. Take as its defining schema

$$a \to \Lambda \qquad (a \text{ in A}) \cdot$$
$$\Lambda \to \cdot *$$

Let $\mathfrak{B}$ be the juxtaposition of $\mathfrak{A}_1, \mathfrak{D}, \mathfrak{A}_2, \mathfrak{D}, \ldots, \mathfrak{D}, \mathfrak{A}_k$, as in Proposition 5.1. Then $\mathfrak{B}(P) \approx \mathfrak{A}_1\#(P) * \mathfrak{A}_2\#(P) * \ldots * \mathfrak{A}_k\#(P)$ for any word P in $\mathrm{A} \cup \{ * \}$ (and $\mathfrak{B}(P) \approx \mathfrak{A}_1(P) * \mathfrak{A}_2(P) * \ldots * \mathfrak{A}_k(P)$ for any word P in the intersection of the alphabets of the $\mathfrak{A}_i$'s).

LEMMA 5.3

(1) *Let* $\mathfrak{C}$ *be a normal algorithm in an alphabet* A *and let* α *be any symbol. Then there is a normal algorithm* $\mathfrak{D}$ *over* $A \cup \{\alpha\}$ *such that*

$$\mathfrak{D}(P) = \begin{cases} \alpha P & \text{if } P \text{ is a word in A such that } \mathfrak{C}(P) = \Lambda \\ P & \text{if } P \text{ is a word in A such that } \mathfrak{C}(P) \neq \Lambda \end{cases}$$

and $\mathfrak{D}$ *applies only to those words to which* $\mathfrak{C}$ *applies.*

(2) *If* $\mathfrak{A}$ *and* $\mathfrak{B}$ *are normal algorithms in an alphabet* A *and* α *is a symbol not in* A, *then there is a normal algorithm* $\mathfrak{G}$ *over* $A \cup \{\alpha\}$ *such that*

$$\mathfrak{G}(P) \approx \mathfrak{A}(P) \quad \text{if P is a word in A, and}$$
$$\mathfrak{G}(\alpha P) \approx \mathfrak{B}(P) \quad \text{if P is a word in A.}$$

PROOF.

(1) There is a normal algorithm $\mathfrak{H}_1$ over $A \cup \{\alpha\}$ taking Λ into α and any other word of $A \cup \{\alpha\}$ into Λ. Let β be any symbol not in $A \cup \{\alpha\}$. Consider the abbreviated schema for $\mathfrak{H}_1$

$$\begin{aligned} a &\to \beta && (a \text{ in } A \cup \{\alpha\}) \\ \beta\beta &\to \beta \\ \beta &\to \cdot\Lambda \\ \Lambda &\to \cdot\alpha \end{aligned}$$

Let $\mathfrak{H}_2 = \mathfrak{H}_1 \circ \mathfrak{C}$. For any word P in A, if $\mathfrak{C}(P) = \Lambda$, then $\mathfrak{H}_2(P) = \alpha$ and if $\mathfrak{C}(P) \neq \Lambda$, then $\mathfrak{H}_2(P) = \Lambda$. Let $\mathfrak{I}$ be the identity algorithm in A (with the schema $\Lambda \to \cdot\,\Lambda$). Let $\mathfrak{D}$ be the juxtaposition of $\mathfrak{H}_2$ and $\mathfrak{I}$. If $\mathfrak{C}(P) = \Lambda$, then $\mathfrak{D}(P) = \alpha P$, and, if $\mathfrak{C}(P) \neq \Lambda$, then $\mathfrak{D}(P) = P$.

(2) For each symbol a of A, let $\bar{a}$ be a new symbol, and let $\bar{A}$ be the alphabet consisting of these $\bar{a}$'s. Let $B = A \cup \bar{A} \cup \{\alpha, \beta\}$, where β is not in $A \cup \bar{A} \cup \{\alpha\}$. If we replace in the schema of algorithm $\mathfrak{B}\cdot$ all symbols a by the corresponding symbols $\bar{a}$, all terminal dots by β, every production $\Lambda \to Q$ by $\alpha \to \alpha Q$, and every production $\Lambda \to \cdot\, Q$ by $\alpha \to \alpha\beta Q$, we obtain a new algorithm schema $\mathfrak{S}_{\mathfrak{B}}$. Let $\mathfrak{S}_{\mathfrak{A}.}$ be the schema for $\mathfrak{A}\cdot$. Form the schema

$$\begin{aligned} \alpha a &\to \alpha\bar{a} && (a \text{ in A}) \\ \bar{a}b &\to \bar{a}\bar{b} && (a, b \text{ in A}) \\ \bar{a}\beta &\to \beta\bar{a} && (a \text{ in A}) \\ \beta\bar{a} &\to \beta a && (a \text{ in A}) \\ a\bar{b} &\to ab && (a, b \text{ in A}) \\ \alpha\beta &\to \cdot\Lambda \\ &\mathfrak{S}_{\bar{\mathfrak{B}}} \\ &\mathfrak{S}_{\mathfrak{A}.} \end{aligned}$$

This determines a normal algorithm $\mathfrak{G}$ over $A \cup \{\alpha\}$ such that $\mathfrak{G}(P) \approx \mathfrak{A}(P)$ and $\mathfrak{G}(\alpha P) \approx \mathfrak{B}(P)$ if P is a word in A.

PROPOSITION 5.4. *Let* $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ *be normal algorithms and* A *the union of their alphabets. Then there is a normal algorithm* $\mathfrak{E}$ *over* A *such that*

$$\mathfrak{E}(P) \approx \begin{cases} \mathfrak{B}(P) & \text{if } P \text{ is a word in A and } \mathfrak{C}(P) = \Lambda \\ \mathfrak{A}(P) & \text{if } P \text{ is a word in A and } \mathfrak{C}(P) \neq \Lambda \end{cases}$$

and $\mathfrak{E}$ *applies only to those words in* A *to which* $\mathfrak{C}$ *is applicable. The algorithm* $\mathfrak{E}$ *is called the* ramification *of* $\mathfrak{A}$ *and* $\mathfrak{B}$ *governed by* $\mathfrak{C}$.

PROOF. Let $\mathfrak{A}_1$, $\mathfrak{B}_1$, $\mathfrak{C}_1$ be the propagations of $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ to A. Let α be a symbol not in A. By Lemma 5.3(1), there is a normal algorithm $\mathfrak{D}$ over $A \cup \{\alpha\}$ such that

$$\mathfrak{D}(P) = \begin{cases} \alpha P & \text{if } P \text{ is a word in A and } \mathfrak{C}(P) = \Lambda \\ P & \text{if } P \text{ is a word in A and } \mathfrak{C}(P) \neq \Lambda \end{cases}$$

By Lemma 5.3(2), there is a normal algorithm $\mathfrak{G}$ over $A \cup \{\alpha\}$ such that $\mathfrak{G}(P) \approx \mathfrak{A}_1(P)$, and $\mathfrak{G}(\alpha P) \approx \mathfrak{B}_1(P)$ if P is a word in A. Let $\mathfrak{E} = \mathfrak{G} \circ \mathfrak{D}$.

Suppose that $\mathfrak{A}$ and $\mathfrak{C}$ are algorithms in an alphabet A and that P_0 is a word in A. First, apply $\mathfrak{A}$ to P_0, and, if a word P_1 results, apply $\mathfrak{C}$ to P_1. If $\mathfrak{C}(P_1) = \Lambda$, stop; if $\mathfrak{C}(P_1) \neq \Lambda$, apply $\mathfrak{A}$ to P_1. If a word P_2 results, test P_2 by $\mathfrak{C}$: if $\mathfrak{C}(P_2) = \Lambda$, stop; if $\mathfrak{C}(P_2) \neq \Lambda$, apply $\mathfrak{A}$ to P_2, and so on. The algorithm $\mathfrak{B}$ defined in this way is called the *iteration* of $\mathfrak{A}$ governed by $\mathfrak{C}$. Clearly, $\mathfrak{B}(P_0) = Q$ when and only when there is a sequence of words $P_0, P_1, \ldots, P_n$ $(n > 0)$ such that $P_n = Q$, $\mathfrak{C}(P_n) = \Lambda$, $P_i = \mathfrak{A}(P_{i-1})$ if $0 < i \leq n$, and $\mathfrak{C}(P_i) \neq \Lambda$ if $0 < i < n$.

PROPOSITION 5.5. *Let* $\mathfrak{A}$ *and* $\mathfrak{C}$ *be normal algorithms, A the union of their alphabets, and* $\mathfrak{A}_1$ *and* $\mathfrak{C}_1$ *the propagations of* $\mathfrak{A}$ *and* $\mathfrak{C}$ *to A. Then the iteration of* $\mathfrak{A}_1$ *governed by* $\mathfrak{C}_1$ *is a normal algorithm over A.*

PROOF. It clearly suffices to prove the result when $\mathfrak{A}$ and $\mathfrak{C}$ have the same alphabet A, in which case $\mathfrak{A}_1 = \mathfrak{A}$ and $\mathfrak{C}_1 = \mathfrak{C}$. Let α be a symbol not in A. By Lemma 5.3(1), there is a normal algorithm $\mathfrak{D}$ over $B = A \cup \{\alpha\}$ such that

$$\mathfrak{D}(P) = \begin{cases} \alpha P & \text{if } P \text{ is a word in A such that } \mathfrak{C}(P) = \Lambda \\ P & \text{if } P \text{ is a word in A such that } \mathfrak{C}(P) \neq \Lambda \end{cases}$$

Let $\mathfrak{F} = \mathfrak{D} \circ \mathfrak{A}$. $\mathfrak{F}$ is a normal algorithm in an extension F of B. Let β be a symbol not in the alphabet F. Consider the following schema.

$$\begin{array}{ll} \xi\beta \to \beta\xi & (\xi \text{ in F}) \\ \beta\alpha \to \cdot\Lambda & \\ \beta \to \Lambda & \\ \mathfrak{S}_{\mathfrak{F}^\beta} & \end{array}$$

where $\mathfrak{S}_{\mathfrak{F}^\beta}$ is a schema for $\mathfrak{F} \cdot$ in which all terminal dots are replaced by β. The normal algorithm $\mathfrak{G}$ defined by this schema is the desired normal algorithm.

COROLLARY 5.6. *Let* $\mathfrak{A}$ *and* $\mathfrak{C}$ *be normal algorithms and* A *the union of their alphabets. Then there is a normal algorithm* $\mathfrak{H}$ *over* A *such that, for any word* P_0 *in* A, $\mathfrak{H}(P_0) = Q$ *if and only if there is a sequence* $P_0, \ldots, P_n$ $(n \geqslant 0)$ *such that* $P_n = Q$, $\mathfrak{C}(P_n) = \Lambda$, $P_{i+1} = \mathfrak{A}(P_i)$ *and* $\mathfrak{C}(P_i) \neq \Lambda$ *for* $0 \leqslant i < n$.

PROOF. Let $\mathfrak{I}$ be the identity algorithm and $\mathfrak{B}$ the iteration of $\mathfrak{A}$ governed by $\mathfrak{C}$. Take $\mathfrak{H}$ to be the ramification of $\mathfrak{B}$ and $\mathfrak{I}$ governed by $\mathfrak{C}$ (cf. Proposition 5.4). This algorithm $\mathfrak{H}$ is called the *full iteration* of $\mathfrak{A}$ governed by $\mathfrak{C}$.

PROPOSITION 5.7. *Let* $\mathfrak{A}$ *be a normal algorithm in an alphabet* A. *Then there is a normal algorithm* $\mathfrak{A}^{\mathrm{I}}$ *over the alphabet* $B = A \cup M$ *(where* $M = \{*, 1\}$*) such that, for any word* P_0 *in* A *and any natural number* n, $\mathfrak{A}^{\mathrm{I}}(\bar{n} * P_0) = Q$ *if and only if there is a sequence* $P_0, \ldots, P_n$ $(n \geqslant 0)$ *with* $P_n = Q$ *and* $P_i = \mathfrak{A}(P_{i-1})$ *for* $0 < i \leqslant n$.

PROOF. Let α be a symbol not in B, and let $C = B \cup \{\alpha\}$. Consider the normal algorithms in C given by the following schemas.

$$\mathfrak{H}_1 : \begin{cases} \alpha 1 1 \to \cdot 1 \\ \alpha 1 * \to \alpha * \\ \alpha * \xi \to \alpha * \quad (\xi \text{ in B}) \\ \alpha * \to \cdot \Lambda \\ \Lambda \to \alpha \end{cases}$$

Clearly, $\mathfrak{H}_1(\bar{0} * P) = \Lambda$ and $\mathfrak{H}_1(\bar{n} * P) \neq \Lambda$, for $n > 0$, where P is any word in B.

$$\mathfrak{H}_2 : \begin{cases} * \xi \to * \quad (\xi \text{ in B}) \\ * \to \Lambda \end{cases}$$

If P does not contain $*$, then $\mathfrak{H}_2(P * Q) = P$.

$$\mathfrak{H}_3 : \quad \begin{cases} \alpha 1 \to \alpha \\ \alpha * \to \cdot \Lambda \\ \Lambda \to \alpha \end{cases}$$

Then $\mathfrak{H}_3(\bar{n} * P) = P$.

$$\mathfrak{H}_4 : \quad 1 \to \cdot \Lambda$$
$$\mathfrak{H}_5 : \quad 1 * \to \cdot \Lambda$$

Clearly, $\mathfrak{H}_4(\bar{n} * P) = \overline{(n-1)} * P$ if $n > 0$, and $\mathfrak{H}_4(\bar{0} * P) = * P$. Also, $\mathfrak{H}_5(\bar{0} * P) = P$.

Let $\mathfrak{E}$ be the normal algorithm given by Corollary 5.2 such that $\mathfrak{E}(P) = (\mathfrak{H}_2 \circ \mathfrak{H}_4)(P) * (\mathfrak{A} \circ \mathfrak{H}_3)(P)$ for any word P in C. For any word P in A,

$$\mathfrak{E}(\bar{n} * P) = \begin{cases} \overline{n-1} * \mathfrak{A}(P) & \text{if n is a positive integer} \\ * \mathfrak{A}(P) \text{ if } n = 0 \end{cases}$$

Let E be the alphabet of $\mathfrak{E}$. By Corollary 5.6, let $\mathfrak{F}$ be a normal algorithm over E such that $\mathfrak{F}(P_0) = Q$ if and only if there is a sequence $P_0, \ldots, P_k$ ($k \geqslant 0$) with $P_k = Q$, $\mathfrak{H}_1(P_k) = \Lambda$, $P_i = \mathfrak{E}(P_{i-1})$ for $0 < i \leqslant k$, and $\mathfrak{H}_1(P_i) \neq \Lambda$ for $0 \leqslant i < k$. Now let $\mathfrak{A}^I = \mathfrak{H}_5 \circ \mathfrak{F}$. We leave as an exercise the verification that this is the required normal algorithm.

PROPOSITION 5.8. *Every partial recursive function is partially Markov-computable, and every recursive function is Markov-computable.*

PROOF.

(1) The initial functions Z, N, I_j^k ($1 \leqslant j \leqslant k$) are Markov-computable (cf. Exercise 5.4, pp. 225–226).

(2) Substitution. Assume that ψ arises from $\tau, \varphi_1, \ldots, \varphi_k$ by substitution: $\psi(x_1, \ldots, x_n) = \tau(\varphi_1(x_1, \ldots, x_n), \ldots, \varphi_k(x_1, \ldots, x_n))$, where $\tau, \varphi_1, \ldots, \varphi_k$ are partial recursive. Suppose that there are normal algorithms $\mathfrak{A}_\tau, \mathfrak{A}_{\varphi_1}, \ldots, \mathfrak{A}_{\varphi_k}$ over $M = \{1, *\}$ which partially compute the functions $\tau, \varphi_1, \ldots, \varphi_k$. By Corollary 5.2, there is an algorithm $\mathfrak{B}$ over M such that $\mathfrak{B}(P) \approx \mathfrak{A}_{\varphi_1}(P) * \mathfrak{A}_{\varphi_2}(P) * \cdots * \mathfrak{A}_{\varphi_k}(P)$ for any word P in M. In particular,

$$\mathfrak{B}(\overline{(x_1, \ldots, x_n)}) \approx \overline{\varphi_1(x_1, \ldots, x_n)} * \overline{\varphi_2(x_1, \ldots, x_n)} * \ldots * \overline{\varphi_k(x_1, \ldots, x_n)}$$

for any natural numbers $x_1, \ldots, x_n$. Now, let $\mathfrak{C} = \mathfrak{A}_\tau \circ \mathfrak{B}$. Then

$$\begin{aligned}\mathfrak{C}(\overline{(x_1, \ldots, x_n)}) &\approx \mathfrak{A}_\tau(\overline{\varphi_1(x_1, \ldots, x_n)} * \ldots * \overline{\varphi_k(x_1, \ldots, x_n)}) \\ &\approx \overline{\tau(\varphi_1(x_1, \ldots, x_n), \ldots, \varphi_k(x_1, \ldots, x_n))}\end{aligned}$$

for any natural numbers $x_1, \ldots, x_n$.

(3) Recursion. Assume that ψ arises from τ and φ by recursion:

$$\begin{aligned}\psi(x_1, \ldots, x_k, 0) &= \tau(x_1, \ldots, x_k) \\ \psi(x_1, \ldots, x_k, y + 1) &= \varphi(x_1, \ldots, x_k, y, \psi(x_1, \ldots, x_k, y))\end{aligned}$$

Suppose that τ and φ are partial recursive and that $\mathfrak{A}_\tau$ and $\mathfrak{A}_\varphi$ are normal algorithms over M which partially compute τ and φ. Let $\mathfrak{A}_Z$ be the normal algorithm computing the zero function, $\mathfrak{A}_N$ the normal algorithm computing the successor function, and let $\mathfrak{A}_j^k$ be the normal algorithm computing the projection function U_j^k. By Corollary 5.2, using the algorithms $\mathfrak{A}_i^{k+1}$, there is a normal algorithm $\mathfrak{B}_1$ over M such that $\mathfrak{B}_1(\bar{x}_1 * \ldots * \bar{x}_k * \bar{y}) = \bar{x}_1 * \ldots * \bar{x}_k$. Let $\mathfrak{R} = \mathfrak{A}_\tau \circ \mathfrak{B}_1$. Again by Corollary 5.2, applied to $\mathfrak{A}_{k+1}^{k+1}, \mathfrak{A}_1^{k+1}, \ldots, \mathfrak{A}_k^{k+1}, \mathfrak{A}_Z, \mathfrak{R}$, there is a normal algorithm $\mathfrak{B}_2$ over M such that $\mathfrak{B}_2(\bar{x}_1 * \ldots * \bar{x}_k * \bar{y}) \approx \bar{y} * \bar{x}_1 * \ldots * \bar{x}_k * \bar{0} * \overline{\tau(x_1, \ldots, x_k)}$. Let $\mathfrak{B}_3 = \mathfrak{A}_N \circ \mathfrak{A}_{k+1}^{k+2}$. Thus, $\mathfrak{B}_3(\bar{x}_1 * \ldots * \bar{x}_k * \bar{y} * \bar{x}) = y + 1$. By Corollary 5.2, applied to $\mathfrak{A}_1^{k+2}, \ldots, \mathfrak{A}_k^{k+2}, \mathfrak{B}_3, \mathfrak{A}_\varphi$, we obtain the juxtaposition algorithm $\mathfrak{B}_4$ over M such that

$$\mathfrak{B}_4(\bar{x}_1 * \ldots * \bar{x}_k * \bar{y} * \bar{z}) \approx \bar{x}_1 * \ldots * \bar{x}_k * \overline{y + 1} * \overline{\varphi(x_1, \ldots, x_k, y, z)}$$

By Proposition 5.7, there is a normal algorithm $\mathfrak{B}_4^{\mathrm{I}}$ such that, if $n \geqslant 0$, $\mathfrak{B}_4^{\mathrm{I}}(\bar{n} * P_0) = Q$ when and only when there is a sequence $P_0, \ldots, P_n$ such that $Q = P_n$ and $P_i = \mathfrak{B}_4(P_{i-1})$ for $0 < i \leqslant n$. Then $\mathfrak{B} = \mathfrak{A}_{k+2}^{k+2} \circ \mathfrak{B}_4^{\mathrm{I}} \circ \mathfrak{B}_2$ is a normal algorithm over M computing ψ. Notice that

$$\mathfrak{B}_2(\bar{x}_1 * \ldots * \bar{x}_k * \bar{y}) \approx \bar{y} * \bar{x}_1 * \ldots * \bar{x}_k * \bar{0} * \overline{\tau(x_1, \ldots, x_k)}$$

If we then apply $\mathfrak{B}_4^{\mathrm{I}}$, this produces a y-fold iteration of $\mathfrak{B}_4$ starting with $\bar{x}_1 * \ldots * \bar{x}_k * \bar{0} * \overline{\tau(x_1, \ldots, x_k)}$. It is easy to see that the result is then $\bar{x}_1 * \ldots * \bar{x}_k * \bar{y} * \overline{\psi(x_1, \ldots, x_k, y)}$. Then, applying $\mathfrak{A}_{k+2}^{k+2}$, we obtain $\overline{\psi(x_1, \ldots, x_k, y)}$.

(4) μ-operator. Suppose that $\psi(x_1, \ldots, x_n) = \mu y(\varphi(x_1, \ldots, x_n, y) = 0)$ and assume that φ is partially computable by a normal algorithm $\mathfrak{A}_\varphi$ over M. By Corollary 5.2, applied to the algorithms $\mathfrak{A}_1^{n+1}, \ldots, \mathfrak{A}_n^{n+1}, \mathfrak{A}_N \circ \mathfrak{A}_{n+1}^{n+1}$, there is a normal algorithm $\mathfrak{W}$ such that $\mathfrak{W}(\bar{x}_1 * \ldots * \bar{x}_n * \bar{y}) = \bar{x}_1 * \ldots * \bar{x}_n * \overline{y+1}$. Let $\mathfrak{D}$ be the normal algorithm over M given by the schema

$$11 \to \cdot 11$$
$$1 \to \Lambda$$

Then $\mathfrak{D}(\bar{n}) = \Lambda$ if $n = 0$ and $\mathfrak{D}(\bar{n}) \neq \Lambda$ if $n > 0$. Let $\mathfrak{C} = \mathfrak{D} \circ \mathfrak{U}_\varphi$. Then

$$\mathfrak{C}(\bar{x}_1 * \ldots * \bar{x}_n * \bar{y}) \begin{cases} = \Lambda & \text{if } \varphi(x_1, \ldots, x_n, y) = 0 \\ \neq \Lambda & \text{if } \varphi(x_1, \ldots, x_n, y) \neq 0 \end{cases}$$

Let $\mathfrak{R}$ be a normal algorithm over M such that

$$\mathfrak{R}(\bar{x}_1 * \ldots * \bar{x}_n) = \bar{x}_1 * \ldots * \bar{x}_n * \bar{0}$$

By Corollary 5.6, applied to $\mathfrak{W}$ and $\mathfrak{C}$, there is a normal algorithm $\mathfrak{H}$ over M such that $\mathfrak{H}(P_0) = Q$ if and only if there is a sequence $P_0, \ldots, P_n$ ($n \geqslant 0$) such that $P_n = Q$, $\mathfrak{C}(P_n) = \Lambda$, $P_{i+1} = \mathfrak{W}(P_i)$ and $\mathfrak{C}(P_i) \neq \Lambda$ for $0 \leqslant i < n$. Let $\mathfrak{B} = \mathfrak{A}_{n+1}^{n+1} \circ \mathfrak{H} \circ \mathfrak{R}$. Then

$$\mathfrak{B}(\bar{x}_1 * \ldots * \bar{x}_n) \approx \overline{\mu y(\varphi(x_1, \ldots, x_n, y) = 0)} \approx \overline{\psi(x_1, \ldots, x_n)}$$

From Parts (1)–(4), if ψ is a partial recursive function of k arguments, there is a normal algorithm $\mathfrak{A}_\psi$ over M such that

$$\mathfrak{A}_\psi(\bar{x}_1, * \ldots * \bar{x}_k) \approx \overline{\psi(x_1, \ldots, x_k)}$$

Let $\mathfrak{R}$ be a normal algorithm over M such that $\mathfrak{R}$ is defined only for words of M of the form $\bar{x}_1 * \ldots * \bar{x}_k$, where $x_1, \ldots, x_k$ are natural numbers, and $\mathfrak{R}(\bar{x}_1 * \ldots * \bar{x}_k) = \bar{x}_1 * \ldots * \bar{x}_k$. (We leave the construction of a schema for $\mathfrak{R}$ as an exercise.) Take $\mathfrak{E}_\psi = \mathfrak{A}_\psi \circ \mathfrak{R}$. Then $\mathfrak{E}_\psi(\bar{x}_1 * \ldots * \bar{x}_k) \approx \overline{\psi(x_1, \ldots, x_k)}$ and $\mathfrak{E}_\psi$ is defined only for those words of M of the form $\bar{x}_1 * \ldots * \bar{x}_k$ such that $\psi(x_1, \ldots, x_k)$ is defined. Hence, every partial recursive function is partially Markov-computable. Every recursive function is, a fortiori, partially Markov-computable, and, since it is total, it is Markov-computable.

We shall now assign Gödel numbers to the symbols S_0, S_1, S_2, ... out of which alphabets are constructed: $g(S_i) = 2i + 3$. Then, to any word $P = S_{j_0} \ldots S_{j_k}$ we assign the number

$$g(P) = 2^{g(S_{j_0})} 3^{g(S_{j_1})} \ldots p_k^{g(S_{j_k})} = 2^{2j_0+3} 3^{2j_1+3} \ldots p_k^{2j_k+3}$$

where p_k is the k^{th} prime number; we define $g(\Lambda) = 1$. To a sequence of words $P_0, \ldots, P_k$, we assign the number $2^{g(P_0)} 3^{g(P_1)} \ldots p_k^{g(P_k)}$.

We make the convention that S_1 is abbreviated by 1, and S_2 by $*$. Considering the numerals as words, we have $g(\bar{0}) = 2^5$; $g(\bar{1}) = 2^5 \cdot 3^5$, and, in general, $g(\bar{n}) = \prod_{i=0}^{n} p_i^5$.

There are normal algorithms $\mathfrak{T}_1$, $\mathfrak{T}_2$ over A $\cup$ M such that $\mathfrak{T}_1(P) = \overline{g(P)}$ for any word P in the alphabet A, and $\mathfrak{T}_2(\overline{g(P)}) = P$ for any word P in A. First, there is a normal algorithm $\mathfrak{B}_1$ over A $\cup$ M such that, for any non-empty word $P = a_{m_0} a_{m_1} \ldots a_{m_r}$ of A,

$$\mathfrak{B}_1(P) = g(a_{m_0}) * g(a_{m_1}) * \ldots * g(a_{m_r}) * \text{ and } \mathfrak{B}_1(a_{m_0}) = g(a_{m_0}) *$$

If A $= \{S_{j_0}, \ldots, S_{j_k}\}$, then the schema for $\mathfrak{B}_1$ is

$$\alpha S_{j_0} \to \overline{2j_0 + 3} * \alpha$$

$$\alpha S_{j_1} \to \overline{2j_1 + 3} * \alpha$$

$$\vdots$$

$$\alpha S_{j_k} \to \overline{2j_k + 3} * \alpha$$

$$\alpha \to \cdot \Lambda$$

$$\Lambda \to \alpha$$

Second, there is a normal algorithm $\mathfrak{B}_2$ such that $\mathfrak{B}_2(\bar{n} * Q) = \bar{0} * \overline{2^n} * Q$. (Exercise. Note that the function 2^x is recursive; so, by Proposition 5.8, there is a normal algorithm computing it.) Let $\mathfrak{B}_3 = \mathfrak{B}_2 \circ \mathfrak{B}_1$. Then, for any non-empty word $P = S_{m_0} \ldots S_{m_r}$,

$$\mathfrak{B}_3(P) = \bar{0} * \overline{2g(S_{m_0})} * \overline{g(S_{m_1})} * \ldots * \overline{g(S_{m_r})} *$$

Let $\mathfrak{A}$ be a normal algorithm such that

$$\mathfrak{A}(\bar{n} * \bar{u} * \bar{v} * Q) = \overline{n + 1} * \overline{u \cdot (p_{n+1})^v} * Q$$

(Exercise. Notice that the function $f(x, y, n) = x \cdot (p_{n+1})^y$ is recursive and hence computable by a normal algorithm.) Let $\mathfrak{C}$ be a normal algorithm such that $\mathfrak{C}(P) = \Lambda$ when and only when P contains exactly two occurrences of $*$. Using Corollary 5.6, let $\mathfrak{H}$ be the full iteration of $\mathfrak{A}$ governed by $\mathfrak{C}$; let $\mathfrak{E}$ be a normal algorithm such that $\mathfrak{E}(\bar{x} * \bar{y} *) = \bar{y}$, and let $\mathfrak{F} = \mathfrak{E} \circ \mathfrak{H} \circ \mathfrak{B}_3$. Then, for any non-empty word P of A, $\mathfrak{F}(P) = \overline{g(P)}$. Hence, if we use Proposition 5.4 to take care of the case $P = \Lambda$, there is a normal algorithm $\mathfrak{T}_1$ over A $\cup$ M such that $\mathfrak{T}_1(P) = \overline{g(P)}$ for any word P in A. (Remember that $g(\Lambda) = 1$.)

EXERCISES

5.5. Prove that there is a normal algorithm $\mathfrak{T}_2$ over A $\cup$ M such that $\mathfrak{T}_2(\overline{g(P)}) = P$ for any word P in A.

Hint: construct a normal algorithm $\mathfrak{D}$ such that $\mathfrak{D}(\overline{2i + 3}) = S_i$ for each symbol S_i of A, but $\mathfrak{D}$ is not defined for any other words. Construct a normal algorithm $\mathfrak{R}$ such that $\mathfrak{R}(\bar{u}) = \bar{0} * \bar{u} *$, for any positive integer u but $\mathfrak{R}$ is not defined for any other words. Construct a normal algorithm $\mathfrak{F}$ such that

$$\mathfrak{F}(\bar{n} * \bar{u} * P) = \overline{n + 1} * \overline{Qt(p_n^{(u)_n}, u)} * P\mathfrak{D}(\overline{(u)_n})$$

for any non-negative integers n, u and any word P. Let $\mathfrak{C}$ be a normal algorithm such that $\mathfrak{C}(\bar{n} * \bar{1} * P) = \Lambda$ for any non-negative integer n and word P, and $\mathfrak{C}$ is defined but not equal to Λ for words not of the form $\bar{n} * \bar{1} * P$. By Proposition 5.5, let $\mathfrak{R}$ be the normal algorithm which is the iteration of $\mathfrak{F}$ governed by $\mathfrak{C}$. Let $\mathfrak{G}$ be a normal algorithm such that $\mathfrak{G}(\bar{n} * \bar{1} * P) = P$ for any non-negative integer n and any word P of A. Let $\mathfrak{L} = \mathfrak{G} \circ \mathfrak{R} \circ \mathfrak{R}$. Then $\mathfrak{L}(\overline{g(Q)}) = Q$ for any non-empty word Q of A. Use Proposition 5.4 to take care of the empty word.

Let $\mathfrak{A}$ be any algorithm (not necessarily normal) over an alphabet A. We can associate with $\mathfrak{A}$ a partial function $\psi_{\mathfrak{A}}$ such that $\psi_{\mathfrak{A}}(n) = m$ if and only if either n is not the Gödel number of a word of A and m = 0 or n and m are Gödel numbers of words P and Q of A such that $\mathfrak{A}(P) = Q$. Suppose that $\psi_{\mathfrak{A}}$ is partial recursive. (We then call $\mathfrak{A}$ a *recursive algorithm.*) By Proposition 5.8, there is a normal algorithm $\mathfrak{B}$ over M such that $\mathfrak{B}(\bar{n}) \approx \overline{\psi_{\mathfrak{A}}(n)}$ for any natural number n and $\mathfrak{B}$ is defined only for those $\bar{n}$ for which $\psi_{\mathfrak{A}}(n)$ is defined. Let $\mathfrak{A}'$ be the normal algorithm $\mathfrak{T}_2 \circ \mathfrak{B} \circ \mathfrak{T}_1$. Then $\mathfrak{A}'$ is a normal algorithm over A which is fully equivalent to $\mathfrak{A}$ relative to A. Thus:

PROPOSITION 5.9. *If $\mathfrak{A}$ is any algorithm over* A, *and $\psi_{\mathfrak{A}}$ is partial recursive, then $\mathfrak{A}$ is fully equivalent relative to* A *to some normal algorithm over* A.

PROPOSITION 5.10. *If $\mathfrak{A}$ is a normal algorithm over* A, *then $\psi_{\mathfrak{A}}$ is partial recursive, and, if $\mathfrak{A}$ is applicable to all words in* A, *$\psi_{\mathfrak{A}}$ is recursive.*

PROOF. Given a simple production $P \to Q$, we call $2^1 3^{g(P)} 5^{g(Q)}$ its index; given a terminal production $P \to \cdot Q$, we let $2^2 3^{g(P)} 5^{g(Q)}$ be its index. If $P_0 \to (\cdot)Q_0, \ldots, P_r \to (\cdot)Q_r$ is an algorithm schema, we let its *index* be the number $2^{k_0} 3^{k_1} \ldots p_r^{k_r}$, where k_i is the index of $P_i \to (\cdot)Q_i$. Let Word (u) be the recursive predicate which holds if and only if u is the Gödel number of a word: $u = 1 \vee (z)(z < lh(u) \supset (Ey)(y < u \wedge (u)_z = 2y + 3))$. Let SI(u) be the recursive predicate which holds if and only if u is the index of a simple production: $lh(u) = 3 \wedge (u)_0 = 1 \wedge$ Word$((u)_1) \wedge$ Word$((u)_2)$. Similarly, let TI(u) be the recursive predicate which holds if and only if u is the index of a terminal production: $lh(u) = 3 \wedge (u)_0 = 2 \wedge$ Word$((u)_1) \wedge$ Word$((u)_2)$. Let Ind(u) be the recursive predicate which holds if and only if u is the index of an algorithm schema: $u > 1 \wedge (z)(z < lh(u) \supset SI((u)_z) \vee TI((u)_z))$. Let $x \square y$ stand for the recursive function which we denoted $x * y$ on p. 144(4). Then, if $x = \prod_{i=0}^{n} p_i^{\alpha_i}$

and each $\alpha_i > 0$, and $y = \Pi_{i=0}^{m} p_i^{\beta_i}$, $x \square y = \Pi_{i=0}^{n} p_i^{\alpha_i} \cdot \Pi_{i=0}^{m} p_{i+n+1}^{\beta_i}$. In addition, $x \square 1 = 1 \square x = x$.

$\square$ corresponds to the juxtaposition operation on words. Let Lsub(x, y, e) be the recursive predicate which holds if and only if e is the index of a production $P \to (\cdot)Q$ and x and y are Gödel numbers of words U and V such that P occurs in U, and V is the result of substituting Q for the left-most occurrence of P in U : Word(x) $\wedge$ Word(y) $\wedge$ (SI(e) $\vee$ TI(e)) $\wedge$ $(Eu)_{u \leqslant x}(Ev)_{v \leqslant x}(x = u \square (e)_1 \square v \wedge y = u \square (e)_2 \square v \wedge \sim (Ew)_{w \leqslant x} (Ez)_{z \leqslant x} (x = w \square (e)_1 \square z \wedge w < u))$. Let Occ(x, y) be the recursive predicate which holds if and only if x and y are Gödel numbers of words U and V, and V occurs in U: Word(x) $\wedge$ Word(y) $\wedge$ $(Ev)_{v \leqslant x}(Ez)_{z \leqslant x}(x = v \square y \square z)$. Let End(e, z) be the recursive predicate holding if and only if z is the Gödel number of a word P, e is the index of an algorithm schema, and any algorithm $\mathfrak{A}$ defined by this schema cannot be applied to P (i.e., $\mathfrak{A}$: $P\square$): Ind(e) $\wedge$ Word(z) $\wedge$ $(w)_{w < lh(e)}(\sim$ Occ(z, $((e)_w)_1))$. Let SCons (e, y, x) be the recursive predicate which holds if and only if e is the index of an algorithm schema and y and x are Gödel numbers of words V and U such that V arises from U by a simple production of the schema:

$$\text{Ind(e)} \wedge \text{Word(x)} \wedge \text{Word(y)} \wedge (\text{Ev})_{v < lh(e)}(\text{SI}((\text{e})_v)$$

$$\wedge \text{Lsub(x, y, (e)}_v) \wedge (\text{z})_{z < v}(\sim \text{Occ(x, ((e)}_z)_1)))$$

Similarly, one defines the recursive predicate TCons(e, y, x) which differs from SCons(e, y, x) only in that the production in question is terminal. Let Der(e, x, y) be the recursive predicate which is true when and only when e is the index of an algorithm schema, x is the Gödel number of a word U_0, y is the Gödel number of a sequence of words $U_0, \ldots, U_k$ ($k \geqslant 0$) such that, for $0 \leqslant i < k \dot{-} 1$, U_{i+1} arises from U_i according to an algorithm $\mathfrak{A}$ determined by the schema, and, either $\mathfrak{A}$: $U_{k \dot{-} 1} \vdash \cdot\, U_k$, or $\mathfrak{A}$: $U_{k \dot{-} 1} \vdash U_k$ and $\mathfrak{A}$: $U_k \square$ (or, if $k = 0$, just $\mathfrak{A}$: $U_k \square$): Ind(e) $\wedge$ Word(x) $\wedge$ $(z)_{z < lh(y)}$(Word$((y)_z) \wedge$

$$(y)_0 = x \wedge (z)_{z < lh(y) \dot{-} 2}(\text{SCons(e, } (y)_{z+1}, (y)_z)) \wedge$$

$$((\text{lh(y)} = 1 \wedge \text{End(e, } (y)_0)) \vee (\text{lh(y)} > 1 \wedge (\text{TCons(e, } (y)_{lh(y) \dot{-} 1}, (y)_{lh(y) \dot{-} 2})$$

$$\vee (\text{SCons(e, } (y)_{lh(y) \dot{-} 1}, (y)_{lh(y) \dot{-} 2}) \wedge \text{End(e, } (y)_{lh(y) \dot{-} 1})))))$$

Let A be any alphabet $\{S_{j0}, \ldots, S_{jm}\}$, and let $W_A(u)$ be the recursive predicate which holds if and only if u is the Gödel number of a word of A: $u = 1 \vee (z)_{z < lh(u)}((u)_z = 2j_0 + 3 \vee \ldots \vee (u)_z = 2j_m + 3)$. Now, let $\mathfrak{A}$ be any normal algorithm over the alphabet A, and let e be the index of the algorithm schema for $\mathfrak{A}$. Define the partial recursive function $\varphi(x) = \mu y((W_A(x) \wedge \text{Der(e, x, y)}) \vee \sim W_A(x))$. But, $\psi_{\mathfrak{A}}(x) = (\varphi(x))_{lh(\varphi(x)) \dot{-} 1}$, and so, $\psi_{\mathfrak{A}}$ is partial recursive. If $\mathfrak{A}$ is applicable to every word in A, then φ is recursive; hence, so is $\psi_{\mathfrak{A}}$.

EXERCISES

5.6. Let A be an alphabet. Show that there is a normal algorithm $\mathfrak{B}$ over A ∪ M such that, for any normal algorithm $\mathfrak{A}$ in A determined by an algorithm schema with index e, $\mathfrak{B}(\bar{e} * P) \approx \mathfrak{A}(P)$ for any word P in A. ($\mathfrak{B}$ can be considered a *universal* algorithm for A.)

COROLLARY 5.11. *Let φ be a partial function. If φ is partially Markov-computable, then φ is partial recursive, and, if φ is Markov-computable, then φ is recursive.*

PROOF. Let $\mathfrak{A}$ be a normal algorithm over M such that $\varphi(n_1, \ldots, n_k) = m$ if and only if $\mathfrak{A}(\overline{(n_1, \ldots, n_k)}) = \bar{m}$. By Proposition 5.10, the function $\psi_{\mathfrak{A}}$ is partial recursive. Define the recursive function $\gamma(x) = \mathrm{lh}(x) \dot{-} 1$. If $x = \prod_{i=0}^{n}(p_i)^5$, then $n = \gamma(x)$. Let

$$\xi(n_1, \ldots, n_k) = g(\overline{(n_1, \ldots, n_k)}) = g(1^{n_1+1} * 1^{n_2+1} * \ldots * 1^{n_k+1})$$

$$= \left[\prod_{i=0}^{n_1+1}(p_i)^5\right] \cdot (p_{n_1+2})^7 \cdot \left[\prod_{i=0}^{n_2+1}(p_{i+n_1+3})^5\right] \cdot (p_{n_1+n_2+5})^7 \cdot \ldots$$

$$\ldots \cdot (p_{n_1+\ldots+n_{k-1}+2k\dot{-}3})^7 \cdot \left[\prod_{i=0}^{n_k+1}(p_{i+n_1+\ldots+n_{k-1}+2k\dot{-}2})^5\right]$$

ξ is clearly recursive. Then $\varphi = \gamma \circ \psi_{\mathfrak{A}} \circ \xi$ is partial recursive. If φ is Markov-computable, then $\mathfrak{A}$ can be assumed applicable to every word in M. (Set up the algorithm schema for $\mathfrak{A}$ so that it takes every word in M not of the form $\bar{n}_1 * \ldots * \bar{n}_k$ into the empty word.) Then, by Proposition 5.10, $\psi_{\mathfrak{A}}$ is recursive. Hence $\varphi = \gamma \circ \psi_{\mathfrak{A}} \circ \xi$ is recursive.

EXERCISE 5.7. *Show that every total partial recursive function is recursive.*

Thus, the equivalence between partial recursiveness and partial Markov-computability (and between recursiveness and Markov-computability) has been established by Corollary 5.11 and Proposition 5.8. Church's Thesis asserts that recursiveness is equivalent to effective computability (and, in an extended form, that partial recursiveness is equivalent to partial effective computability). In terms of algorithms, Markov has formulated the corresponding *Normalization Principle*: Every algorithm in A is fully equivalent relative to A to some normal algorithm over A. Now, Church's Thesis (in the extended form) and Markov's Principle are equivalent. First, assume Church's Thesis. Let $\mathfrak{A}$ be an algorithm in an alphabet A. Then $\psi_{\mathfrak{A}}$ is a partial effectively computable function. Hence, by Church's Thesis, $\psi_{\mathfrak{A}}$ is partial recursive, and so, by Proposition 5.9, $\mathfrak{A}$ is fully equivalent relative to A to some normal algorithm $\mathfrak{B}$, i.e., Markov's Principle holds. Conversely, assume Markov's Principle. Let φ be a partial effectively computable function. Let $\mathfrak{B}_\varphi$ be the corresponding algorithm in M. By Markov's Principle, $\mathfrak{B}_\varphi$ is fully equivalent to a normal algorithm relative to M. Hence, φ is partially Markov-computable, and, by Corollary 5.11, φ is partial recursive. Thus, Church's Thesis holds.

Of course, because of the vagueness of the intuitive notions of effectively computable function and algorithm, it is impossible to *prove* the validity of Church's Thesis or Markov's Principle. Nor is there any a priori reason to support these hypotheses. There is no apparent reason why the use of productions alone should account for all effective operations. One can only expect incomplete confirmation, not a rigorous proof. It is clear that every partial recursive function is a partial effectively computable function.† The converse assertion, that every partial effectively computable function is partial recursive (or, equivalently, that every algorithm in an alphabet A is fully equivalent relative to A to some normal algorithm) has been confirmed for every known partial effectively computable function. There is some additional evidence in favor of Church's Thesis, namely, the odd fact that quite dissimilar attempts to precisely define the notion of partial effectively computable function have all proved to be equivalent. We have seen this already for partial recursiveness and partial Markov-computability. Other approaches, by Turing and by Herbrand and Gödel, will be shown later to lead to the same result. In addition, Church's theory of λ-computability [1941] and Post's theory of normal systems [1943] also yield notions equivalent to that of partial-recursive function or normal algorithm. (Arguments for Church's Thesis may be found in Kleene [1952], §§ 62, 70. Also consult Hermes [1965].)

EXERCISES

5.8. Show that the Normalization Principle is equivalent to the assertion that every algorithm in an alphabet A is equivalent relative to A to some normal algorithm over A.

5.9. Given an alphabet B and an alphabet $A = \{a_1, \ldots, a_k\}$ disjoint from B. Let b, c be distinct symbols not in B ∪ A. For any symbol a, we denote by a^i the word $\underbrace{aa \ldots a}_{i \text{ times}}$. The *translation* $T(a_i)$ of a_i is defined to be the word cb^ic, and the translation $T(u)$ of any symbol u in B is u itself; the translation $T(P)$ of a word $P = d_1 \ldots d_n$ in B ∪ A is defined to be $T(d_1) \ldots T(d_n)$, while $T(\Lambda) = \Lambda$. Note that $T(P) = P$ for any word P in B.

(a) Show that the schema

$$\begin{aligned} \alpha\xi &\to T(\xi)\alpha \qquad (\xi \text{ in } B \cup A) \\ \alpha &\to \cdot\Lambda \\ \Lambda &\to \alpha \end{aligned}$$

defines a normal algorithm $\mathfrak{T}$ over B ∪ A ∪ {b, c} such that $\mathfrak{T}(P) = T(P)$ for any word P in B ∪ A. (Assume that α is not in B ∪ A ∪ {b, c}.)

†The reader should notice that partial effective computability does not necessarily imply human computability. Partial effective computability means that the values of the function can be computed, according to a fixed procedure, in a finite number of steps. Some of the computations needed to obtain the values of a partial recursive function involve so many steps that the human race may not exist long enough to carry them out.

(b) Give the schema for a normal algorithm $\mathfrak{B}$ over B ∪ A ∪ $\{b, c\}$ such that $\mathfrak{B}(T(P)) = P$ for any word P in B ∪ A.

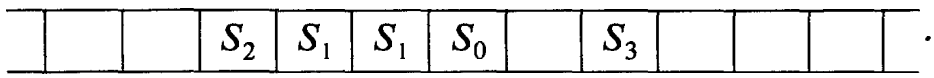
(c) Let $\mathfrak{C}$ be any normal algorithm in B ∪ A. For any production $P \to (\cdot)Q$ of the schema for $\mathfrak{C}$, the translation of this production is taken to be the production $T(P) \to (\cdot)T(Q)$. The translation of all the productions in the schema for $\mathfrak{C}$ gives an algorithm schema defining a normal algorithm $T(\mathfrak{C})$ in B ∪ $\{b, c\}$. If $\mathfrak{T}$ is the algorithm of Part (a), show that $(T(\mathfrak{C}))(\mathfrak{T}(P)) \approx \mathfrak{T}(\mathfrak{C}(P))$.

(d) Prove that any normal algorithm over B is fully equivalent relative to B to some normal algorithm in B ∪ $\{b, c\}$. (That the number of additional symbols can be reduced from two to one has been shown by Nagornyi [1953]. However, in the same paper, Nagornyi states that there is a normal algorithm over B, the doubling algorithm (Exercise 5.2(d), page 225), which is not equivalent relative to B to any normal algorithm in B itself. This is an easy exercise for the reader.)

2. Turing Algorithms

Attempting to give a precise definition of effective computability, Turing [1936] proposed that a certain class of abstract machines could perform any "mechanical" computing procedure. Such machines are now called Turing machines in honor of their inventor, and can be described in the following way.

There is a two-way potentially infinite tape divided up into squares.



The tape is said to be potentially infinite in the sense that, although at any moment it is finite in length, additional squares always can be added to the right- and left-hand ends of the tape. There is a finite set of *tape symbols* $S_0, S_1, \ldots, S_n$ called the *alphabet* of the machine; at every moment, each square of the tape is occupied by at most one symbol. The machine has a finite set of *internal states* $\{q_0, q_1, \ldots, q_m\}$. At any given moment, the machine is in exactly one of these states. Finally, there is a reading head which, at any given time, stands over some square of the tape. The machine does not act continuously, but only at discrete moments of time. If, at any moment t, the reading head is scanning (i.e., is standing over) a square containing a symbol S_i and the machine is in the internal state q_j, then the action of the machine is determined, and it will do one of four things: (1) it may erase the symbol S_i and print a new symbol S_k; (2) it may move left one square; (3) it may move right one square; (4) it may stop. In cases (1)–(3), the machine goes into a new internal state q_r, and is ready to act again at time t + 1. We shall assume that the symbol S_0 represents a blank, so that the reading head may always be assumed to be scanning a symbol. The first three actions of the machine just described can be represented by quadruples: either (1) $q_j S_i S_k q_r$, or (2) $q_j S_i L q_r$, or (3) $q_j S_i R q_r$. The first two symbols stand for the present internal state and scanned symbol, the third symbol represents the action of the machine (print S_k, or move left, or move right one square), and the fourth symbol gives the internal state of the machine after the action has been performed.

If a tape is put into a Turing machine and the reading head is placed on a certain square, and if the machine is started off in one of its internal states, then the machine begins to operate on the tape: printing and erasing symbols and moving from one square to an adjacent one. If the machine ever stops, the resulting tape is said to be the output of the machine applied to the given tape. Now we can associate with any Turing machine T the following algorithm $\mathfrak{B}$ in the alphabet A of T. Take any word P in the alphabet A and print it from left to right in the squares of an empty tape. Place this tape in the machine with the reading head scanning the left-most square. Start the machine in the internal state q_0. If the machine ever stops, the word of A appearing on the tape is the value of the algorithm $\mathfrak{B}$. $\mathfrak{B}$ is called a *Turing algorithm*. (The word appearing on the tape is defined to be the sequence of symbols beginning with the left-most symbol and moving right to the right-most symbol. Remember that a blank square encountered in this motion is assumed to have the symbol S_0 printed in it.) We have not specified yet the mechanism by which a machine knows when to stop; this will be done below.

Any Turing machine can be determined precisely by a finite set of quadruples of the three kinds: (1) $q_jS_iS_kq_r$; (2) $q_jS_iLq_r$; (3) $q_jS_iRq_r$, such that no two quadruples have the same first two symbols. In fact, we now shall define a *Turing machine* to be such a finite set of quadruples. The *alphabet* of any Turing machine T is the set of tape symbols S_m appearing in any of the quadruples. The *internal states* of the machine are the symbols q_s appearing in the quadruples. We assume that q_0 is an internal state of every Turing machine.

An *instantaneous tape description* of a Turing machine T is a word such that (i) all symbols in the word but one are tape symbols S_m; (ii) the only symbol which is not a tape symbol is an internal state q_s; (iii) q_s is not the last symbol of the word.† We say that T *moves* one instantaneous tape description α into another one β (abbreviated $\alpha \underset{T}{\rightarrow} \beta$) if and only if either (a) α is of the form Pq_jS_iQ, β is of the form Pq_rS_kQ, and $q_jS_iS_kq_r$ is one of the quadruples of T; or (b) α is of the form $PS_sq_jS_iQ$, β is $Pq_rS_sS_iQ$, and $q_jS_iLq_r$ is one of the quadruples of T; or (c) α is of the form q_jS_iQ, β is $q_rS_0S_iQ$, and $q_jS_iLq_r$ is one of the quadruples of T; or (d) α is of the form $Pq_jS_iS_kQ$, β is $PS_iq_rS_kQ$, and $q_jS_iRq_r$ is one of the quadruples of T; or (e) α is of the form Pq_jS_i, β is $PS_iq_rS_0$, and $q_jS_iRq_r$ is one of the quadruples of T.‡

†An instantaneous tape description describes the condition of the machine and the tape at a given moment. When read from left to right, the tape symbols in the description represent the symbols on the tape at the moment. The internal state q_s in the description is the internal state of the machine at the moment, and the tape symbol occurring immediately to the right of q_s in the tape description represents the symbol being scanned by the machine at the moment.

‡Observe that, according to our intuitive picture, "T moves α into β" implies that if the condition at time t of the Turing machine and tape is described by α, then the condition at time t + 1 is described by β. Notice that, according to clause (c), whenever the machine reaches the left-hand end of the tape and is ordered to move left, a blank square is attached to the tape on the left; similarly, by clause (e), a blank square is added on the right when the machine reaches the right-hand end of the tape and has to move right.

We say that T *stops* at an instantaneous tape description α if and only if there is no instantaneous tape description β such that $\alpha \underset{T}{\rightarrow} \beta$. (This happens when $q_j S_i$ occurs in α but $q_j S_i$ are not the first two symbols of a quadruple of T.)

A *computation* of a Turing machine T is a finite sequence of instantaneous tape descriptions $\alpha_0, \ldots, \alpha_m$ ($m \geqslant 0$) such that the internal state occurring in α_0 is q_0; for $0 \leqslant i < m$, $\alpha_i \underset{T}{\rightarrow} \alpha_{i+1}$; and T stops at α_m. This computation is said to begin with α_0 and end with α_m. The algorithm $\mathfrak{B}_{T,C}$ in any alphabet C containing the alphabet A of T is defined as follows: for any words P, Q in C, $\mathfrak{B}_{T,C}(P) = Q$ if and only if there is a computation of T which begins with the instantaneous tape description $q_0 P$ and ends with an instantaneous tape description of the form $R_1 q_j R_2$, where $Q = R_1 R_2$. An algorithm $\mathfrak{A}$ in an alphabet D is called *Turing-computable* if and only if there is a Turing machine T with alphabet A and an alphabet C containing A $\cup$ D such that $\mathfrak{B}_{T,C}$ and $\mathfrak{A}$ are fully equivalent relative to D.

We let 1 stand for S_1. Remember that $\bar{m}$ stands for 1^{m+1}, for any natural number m. Also, let $*$ be an abbreviation of S_2. Given a partial number-theoretic function $f(x_1, \ldots, x_n)$, we say that a Turing machine T (whose alphabet A includes $\{1, *\}$) *computes* f if and only if, for any natural numbers $k_1, \ldots, k_n$, and any word Q, $\mathfrak{B}_{T,A}(\overline{k_1} * \overline{k_2} * \cdots * \overline{k_n}) = Q$ if and only if Q is $R_1 \overline{f(k_1, \ldots, k_n)} R_2$, where both R_1 and R_2 are certain (possibly empty) words consisting only of S_0's. (The form $R_1 \overline{f(k_1, \ldots, k_n)} R_2$ is allowed for the result since S_0 is interpreted as a blank.) The function f is called *Turing-computable* if and only if there is a Turing machine T which computes f.

Examples

1. Consider the Turing machine T defined by the following quadruples.

$$q_0 1 L q_1$$

$$q_1 S_0 1 q_2$$

The alphabet of T is $\{1, S_0\}$. T computes the successor function, since $q_0 \bar{k} \underset{T}{\rightarrow} q_1 S_0 \bar{k} \underset{T}{\rightarrow} q_2 \overline{k+1}$. In general, T takes any $q_0 1 P$ into $q_2 1 1 P$, and T takes any word not beginning with 1 into itself.

2. The machine defined by the quadruples

$$q_0 1 L q_1$$

$$q_1 S_0 1 q_0$$

when started on a word beginning with 1 keeps on adding 1's to the left and never stops.

3. The Turing machine given by the quadruples

$$q_0 S_0 R q_0$$

$$q_0 S_2 R q_0$$

$$\vdots$$

$$q_0 S_k R q_0$$

$$q_0 1\ 1 q_1$$

moves right until it locates the first occurrence (if any) of the symbol 1 and then stops.

4. Let us find a Turing machine T which computes the addition function. Take as the quadruples for T:

$$q_0 1 S_0 q_0$$

$$q_0 S_0 R q_1$$

$$q_1 1 R q_1$$

$$q_1 * 1 q_2$$

$$q_2 1 R q_2$$

$$q_2 S_0 L q_3$$

$$q_3 1 S_0 q_3$$

Then

$$q_0 \overline{m} * \overline{n} = q_0 1^{m+1} * 1^{n+1} \underset{T}{\rightarrow} q_0 S_0 1^m * 1^{n+1} \underset{T}{\rightarrow} S_0 q_1 1^m * 1^{n+1} \underset{T}{\rightarrow} S_0 1 q_1 1^{m \dot{-} 1} * 1^{n+1}$$

$$\underset{T}{\rightarrow} \cdots \underset{T}{\rightarrow} S_0 1^m q_1 * 1^{n+1} \underset{T}{\rightarrow} S_0 1^m q_2 1 1^{n+1} \underset{T}{\rightarrow} S_0 1^{m+1} q_2 1^{n+1} \underset{T}{\rightarrow} S_0 1^{m+1} 1 q_2 1^n \underset{T}{\rightarrow}$$

$$\cdots \underset{T}{\rightarrow} S_0 1^{m+1} 1^{n+1} q_2 S_0 \underset{T}{\rightarrow} S_0 1^{m+1} 1^n q_3 1 S_0 \underset{T}{\rightarrow} S_0 1^{m+1} 1^n q_3 S_0 S_0 =$$

$$S_0 1^{m+n+1} q_3 S_0 S_0 = S_0 \overline{m + n} q_3 S_0 S_0.$$

EXERCISES

5.10. What function f(x) is computed by the following Turing machine?

$$q_0 1 S_0 q_0$$

$$q_0 S_0 R q_1$$

$$q_1 1 1 q_0$$

$$q_1 S_0 1 q_2$$

5.11. Show that the initial primitive recursive functions $U_i^n(x_1, \ldots, x_n)$ are Turing computable.

5.12. Write down the quadruples of a Turing machine which computes the function $f(x) = [x/2]$, the greatest integer $\leqslant x/2$.

5.13. Show that the function $m \dot{-} n$ is Turing-computable. (For more examples, cf. Davis [1958, Chapter 1].)

PROPOSITION 5.12. *Let* T *be a Turing machine with alphabet* A. *Let* C *be an extension of* A, *i.e.*, $C \supseteq A$. *Then there is a normal algorithm* $\mathfrak{A}$ *over* C *which is fully equivalent to the Turing algorithm* $\mathfrak{B}_{T,C}$ *relative to* C.

PROOF. Let $D = C \cup \{q_{k_0}, \ldots, q_{k_m}\}$, where $q_{k_0}, \ldots, q_{k_m}$ are the internal states of T, and $q_{k_0} = q_0$. Write down the algorithm schema for $\mathfrak{A}$ as follows: first, for all quadruples $q_j S_i S_k q_r$ of T, take the productions $q_j S_i \rightarrow q_r S_k$. Second, for each quadruple $q_j S_i L q_r$, take the productions $S_l q_j S_i \rightarrow q_r S_l S_i$ for all symbols S_l of C; then take the production $q_j S_i \rightarrow q_r S_0 S_i$. Third, for each quadruple

$q_jS_iRq_r$, take the productions $q_jS_iS_l \to S_iq_rS_l$ for all symbols S_l of C; then take the production $q_jS_i \to S_iq_rS_0$. Fourth, write down the productions $q_{k_i} \to \cdot\,\Lambda$ for each internal state q_{k_i} of T, and, finally, take $\Lambda \to q_0$. This schema defines an algorithm $\mathfrak{A}$ over C, and it is easy to see that, for any word P of C, $\mathfrak{B}_{T,C}(P) \approx \mathfrak{A}(P)$.

COROLLARY 5.13. *Every Turing-computable function f is partially Markov-computable; hence (by Corollary* 5.11*), f is partial recursive, and, if f is total, then f is recursive.*

PROOF. Let $f(x_1, \ldots, x_n)$ be Turing-computable by a Turing machine T with alphabet $A \supseteq \{1, *\}$. Then by Proposition 5.12, there is a normal algorithm $\mathfrak{A}$ over A such that $\mathfrak{A}$ is fully equivalent to $\mathfrak{B}_{T,A}$ relative to A, where $\mathfrak{B}_{T,A}(\overline{k_1} * \cdots * \overline{k_n}) \approx R_1\overline{f(k_1, \ldots, k_n)}R_2$, R_1 and R_2 being (possibly empty) sequences of S_0's. Let $\mathfrak{C}_1$ be a normal algorithm over $\{1, *, S_0\}$ such that $\mathfrak{C}_1$ erases all S_0's occurring before the first 1 or $*$; as a schema for $\mathfrak{C}_1$ we may take

$$\alpha S_0 \to \alpha$$

$$\alpha 1 \to \cdot 1$$

$$\alpha * \to \cdot *$$

$$\alpha \to \cdot \Lambda$$

$$\Lambda \to \alpha$$

Also, let $\mathfrak{C}_2$ be a normal algorithm over $\{1, *, S_0\}$ such that $\mathfrak{C}_2$ erases all S_0's occurring after the last 1 or $*$ of a word in $\{1, *\}$; a schema for $\mathfrak{C}_2$ is

$$\alpha * \to * \alpha$$

$$\alpha 1 \to 1 \alpha$$

$$\alpha S_0 \to \alpha$$

$$\alpha \to \cdot \Lambda$$

$$\Lambda \to \alpha$$

Now, let $\mathfrak{C}$ be the normal algorithm $\mathfrak{C}_2 \circ \mathfrak{C}_1 \circ \mathfrak{A}$. Then for any $k_1, \ldots, k_n$, $\mathfrak{A}(\overline{k_1} * \ldots * \overline{k_n}) \approx \mathfrak{B}_{T,A}(\overline{k_1} * \ldots * \overline{k_n}) \approx R_1\overline{f(k_1, \ldots, k_n)}R_2$, where R_1 and R_2 are sequences of S_0's. Then

$$\mathfrak{C}_1\big(R_1\,\overline{f(k_1, \ldots, k_n)}\,R_2\big) = \overline{f(k_1, \ldots, k_n)}\,R_2$$

and $\mathfrak{C}_2\big(\overline{f(k_1, \ldots, k_n)}R_2\big) = \overline{f(k_1, \ldots, k_n)}$. Hence, f is partially Markov-computable by $\mathfrak{C}$.

PROPOSITION 5.14. *Let $\mathfrak{A}$ be a normal algorithm in an alphabet* A *not containing S_0 or δ. Then there is a Turing machine* T *such that the Turing algorithm $\mathfrak{B} = \mathfrak{B}_{T,(A \cup \{S_0, \delta\})}$ in the alphabet $A \cup \{S_0, \delta\}$ has the following property: for any word W in* A, *$\mathfrak{B}$ is applicable to W if and only if $\mathfrak{A}$ is, and*

$\mathfrak{B}(W)$ *is of the form* $S_0^m \mathfrak{A}(W) S_0^n$, *where* m *and* n *are non-negative integers.* (The reason for the difference between $\mathfrak{A}$ and $\mathfrak{B}$ is that, while we agree to consider S_0 as a blank on a Turing machine tape, S_0 is treated like any other symbol in the theory of algorithms.)

PROOF. We may assume, by suitable reindexing, that $\mathrm{A} = \{S_1, S_2, \ldots, S_k\}$. Let $P \to (\cdot)Q$ be an arbitrary production. We shall construct Turing machine quadruples which will have the effect of replacing the left-most occurrence (if any) of P in a word W by Q. If $P \neq \Lambda$, let P be $b_0 \ldots b_r$. Then, take the following quadruples.

q_0	S_i	R	q_0	$(S_i \in \mathrm{A},\ S_i \neq b_0)$
q_0	b_0	δ	q_0	
q_0	δ	R	q_2	
q_2	b_1	R	q_3	
q_2	S_i	S_i	q_{r+2}	$(S_i \in \mathrm{A} \cup \{S_0\},\ S_i \neq b_1)$
q_3	b_2	R	q_4	
q_3	S_i	S_i	q_{r+2}	$(S_i \in \mathrm{A} \cup \{S_0\},\ S_i \neq b_2)$
	⋮			
q_r	b_{r-1}	R	q_{r+1}	
q_r	S_i	S_i	q_{r+2}	$(S_i \in \mathrm{A} \cup \{S_0\},\ S_i \neq b_{r-1})$
q_{r+1}	b_r	R	q_{r+4}	
q_{r+1}	S_i	S_i	q_{r+2}	$(S_i \in \mathrm{A} \cup \{S_0\},\ S_i \neq b_r)$
q_{r+2}	S_i	L	q_{r+2}	$(S_i \in \mathrm{A} \cup \{S_0\})$
q_{r+2}	δ	b_0	q_{r+3}	
q_{r+3}	b_0	R	q_0	
q_0	S_0	L	q_{r+5}	
q_{r+5}	S_i	L	q_{r+5}	$(S_i \in \mathrm{A})$
q_{r+5}	δ	b_0	q_{r+5}	
q_{r+5}	S_0	R	q_Y	(where Y is an integer greater than all the other indices, to be specified later)

These quadruples have the following effect on a word W. (Notice that we have not used q_1; q_1 will have a special purpose later on.) If W has no occurrence of P, then we wind up with the instantaneous tape description $q_Y W$; if W has an occurrence of P, and $W = W_1 P W_2$, where the indicated P is the left-most

occurrence of P in W, then we wind up with $W_1Pq_{r+4}W_2$. In the latter case, we must now add some quadruples which will replace the indicated occurrence of P by Q. Let Q be $c_0 \ldots c_s$. There are three cases:

(1) s = r, i.e., P and Q have the same length. Then we add:

q_{r+4}	S_i	L	q_{r+7}	$(S_i \in A \cup \{S_0\})$
q_{r+7}	b_r	c_r	q_{r+8}	
q_{r+8}	c_r	L	q_{r+9}	
q_{r+9}	b_{r-1}	c_{r-1}	q_{r+10}	
q_{r+10}	c_{r-1}	L	q_{r+11}	
	$\vdots$			
q_{3r+7}	b_0	c_0	q_{3r+8}	
q_{3r+8}	S_i	L	q_{3r+8}	$(S_i \in A)$
q_{3r+8}	S_0	R	q_u	$u = \begin{cases} 0 \text{ if } P \to (\cdot)Q \text{ is simple} \\ 1 \text{ if } P \to (\cdot)Q \text{ is terminal} \end{cases}$

Then, applying these quadruples to $W_1Pq_{r+4}W_2$, we obtain $q_uW_1QW_2$.

(2) s < r. Q is shorter than P. Add the quadruples

q_{r+4}	S_i	L	q_{r+7}	$(S_i \in A \cup \{S_0\})$
q_{r+7}	b_r	c_s	q_{r+8}	
q_{r+8}	c_s	L	q_{r+8}	
	$\vdots$			
q_{r+7+2s}	b_{r-s}	c_0	$q_{r+7+2s+1}$	
$q_{r+7+2s+1}$	c_0	L	$q_{r+7+2s+2}$	
$q_{r+7+2s+2}$	b_{r-s-1}	S_0	$q_{r+7+2s+2}$	
$q_{r+7+2s+2}$	S_0	L	$q_{r+7+2s+3}$	
$q_{r+7+2s+3}$	b_{r-s-2}	S_0	$q_{r+7+2s+3}$	
$q_{r+7+2s+3}$	S_0	L	$q_{r+7+2s+4}$	
	$\vdots$			
q_{2r+s+8}	b_0	S_0	q_{2r+s+8}	

After these quadruples work on $W_1Pq_{r+4}W_2$, we have

$$W_1q_{2r+s+8}S_0^{r-s}QW_2$$

Now we must provide some quadruples which will move W_1 r − s squares to the right to obtain W_1QW_2 (preceded by some S_0's). Let M be an integer larger than all the indices of the q_i's and S_i's above, say, M = 3r + 9.

$$
\begin{array}{lllll}
 & q_{2r+s+8} & S_0 & L & q_M & \\
 & q_M & S_j & \delta & q_{M+j} & (S_j \in \mathrm{A}) \\
 & q_{M+j} & \delta & R & q_{M+j} & \\
 & q_{M+j} & S_0 & R & q_{M+j} & \\
 & q_{M+j} & S_l & L & q_{2M+j} & (S_l \in \mathrm{A}) \\
 & q_{2M+j} & S_0 & S_j & q_{2M+j} & \\
 & q_{2M+j} & S_j & L & q_{3M+j} & \\
 & q_{3M+j} & S_0 & L & q_{3M+j} & \\
 & q_{3M+j} & \delta & S_0 & q_{4M+j} & \\
(j = 1, 2, \ldots, k) & q_{4M+j} & S_0 & L & q_{5M+j} & \\
 & q_{5M+j} & S_0 & R & q_{6M+j} & \\
 & q_{6M+j} & S_0 & R & q_{6M+j} & \\
 & q_{6M+j} & S_l & S_l & q_u & (S_l \in \mathrm{A}) \\
 & & & & & u = \begin{cases} 0 \text{ if } P \to (\cdot)Q \text{ is simple} \\ 1 \text{ if } P \to (\cdot)Q \text{ is terminal} \end{cases} \\
 & q_{5M+j} & S_l & S_l & q_M & (S_l \in \mathrm{A})
\end{array}
$$

Beginning with $W_1 q_{2r+s+8} S_0^{r-s} QW_2$, these quadruples produce $(S_0)^p q_u W_1 QW_2$ (where p is a positive integer).

(3) $s > r$, i.e., Q is longer than P. This is left to the reader as an exercise. The treatment is analogous to that of case (2). (If P or Q is empty, the slight modifications necessary in the above constructions are left to be filled in by the reader.)

Now, let us assume that $\mathfrak{A}$ is a normal algorithm in the alphabet A = $\{S_1, \ldots, S_k\}$ not containing S_0 or δ, and that the algorithm $\mathfrak{A}$ is defined by the algorithm schema $P_1 \to (\cdot)Q_1, \ldots, P_h \to (\cdot)Q_h$. We define a Turing machine T as follows: in the work above, take $P \to (\cdot)Q$ to be $P_1 \to (\cdot)Q_1$ and list the appropriate quadruples (it will suffice to take Y to be a number 100 times greater than the sum of k and the number of occurrences of symbols in the

schema). These quadruples have the following effect: given $q_1 W$, if W does not contain P_1, we wind up with $q_Y W$; if $W = W_1 P W_2$, and this indicates the left-most occurrence of P in W, then we finally obtain $(S_0)^v q_u W_1 Q W_2$ (where v is a non-negative integer; and u = 0 if $P_1 \to (\cdot) Q_1$ is simple, and u = 1 if $P_1 \to (\cdot) Q_1$ is terminal). Next, we consider $P_2 \to (\cdot) Q_2$ and form the quadruples for this production as indicated above, except that we raise the subscripts on all q_i's by the amount Y (but q_u is left untouched). The subscripts are raised by Y so that these quadruples will not interfere with the action of the quadruples corresponding to $P_1 \to (\cdot) Q_1$. The new quadruples will go into action only after a word W has been found not to contain P_1; they have the effect of searching W for an occurrence of P_2, and, if one is found, replacing the left-most occurrence of P_2 by Q_2, and winding up back in the initial state q_0 ready for action again by the first group of quadruples if $P_2 \to (\cdot) Q_2$ is simple or winding up in the terminal state q_1 if $P_2 \to (\cdot) Q_2$ is terminal. We now repeat the same process with $P_3 \to (\cdot) Q_3$, this time adding 2Y to the subscripts of the q_i's, etc. It should be clear that the Turing machine T so defined mimics the action of the normal algorithm $\mathfrak{A}$ in such a way that, for any word W in A, $\mathfrak{B} = \mathfrak{B}_{T, A \cup \{S_0, \delta\}}$ is applicable to W if and only if $\mathfrak{A}$ is, and $\mathfrak{B}(W)$ is of the form $(S_0)^m \mathfrak{A}(W)(S_0)^n$, where m and n are non-negative integers. (For a similar proof, cf. Asser [1959]. An indirect proof could have been given by showing that every partial recursive function is Turing-computable and then using Corollary 5.11. Study of Hermes' method of linking Turing machines and his flow-charts (cf. Hermes [1965], § 7) would clarify the procedures used in the proof above).

COROLLARY 5.15. *Every partially Markov-computable function is Turing-computable.* (Hence, every partial recursive function is Turing-computable. For another proof, cf. Kleene [1952], § 68.)

PROOF. From Proposition 5.14 and the definition of Turing-computable function.

Thus, the Turing-machine approach to effective computability is equivalent to that by means of normal algorithms or by recursive functions. A Turing machine seems to be an abstract form of a digital computer (except that no attention is given to speed or convenience of operation). Intuitively, then, the fact that Turing-computable functions are identical with partial recursive functions further substantiates Church's Thesis. In addition, one can show that making additional complications in the structure of Turing machines (such as adding more tapes and reading heads, or using a two-dimensional tape) does not change the class of Turing-computable functions. (Further arguments along these lines may be found in Kleene [1952], pp. 317–323 and 376–381.)

3. Herbrand-Gödel Computability. Recursively Enumerable Sets.

The idea of defining all computable functions in terms of fairly simple systems of equations was proposed by Herbrand and developed by Gödel [1934]. The exposition given here is a version of the presentation in Kleene [1952], Chapter XI.

We define first the *terms*.

(a) All variables are terms.
(b) 0 is a term.
(c) If t is a term, then $(t)'$ is a term.
(d) If $t_1, \ldots, t_n$ are terms and f_j^n is a function letter, $f_j^n(t_1, \ldots, t_n)$ is a term.

For every natural number n, we define the corresponding *numeral* $\bar{n}$ as follows: (1) $\bar{0}$ is 0; (2) $\overline{n+1}$ is $(\bar{n})'$. Thus, every numeral is a term.

An *equation* is a formula $r = s$ where r and s are terms. A *system* E of equations is a finite sequence $r_1 = s_1, r_2 = s_2, \ldots, r_k = s_k$ of equations such that r_k is of the form $f_j^n(t_1, \ldots, t_n)$. The function letter f_j^n is called the *principal letter* of the system E. Those function letters (if any) which appear only on the right side of equations of E are called the *initial letters* of E; any function letter other than the principal letter which appears on the left side of some equations and also on the right side of some equations is called an *auxiliary letter* of E.

We have two rules of inference:

R_1: An equation e_2 is a consequence of an equation e_1 by R_1 if and only if e_2 arises from e_1 by substituting any numeral $\bar{n}$ for all occurrences of a variable.

R_2: An equation e is a consequence by R_2 of equations $f_h^m(\bar{n}_1, \ldots, \bar{n}_m) = \bar{p}$ and $r = s$ if and only if e arises from $r = s$ by replacing one or more occurrences of $f_h^m(\bar{n}_1, \ldots, \bar{n}_m)$ in s by $\bar{p}$, and $r = s$ contains no variables.

A *proof* of an equation e from a set B of equations is a sequence $e_0, \ldots, e_q$ of equations such that $e_q = e$ and, if $0 \leqslant i \leqslant q$, then either (1) e_i is an equation of B, or (2) e_i is a consequence by R_1 of a preceding equation e_j $(j < i)$, or (3) e_i is a consequence by R_2 of two preceding equations e_j and e_m $(j < i, m < i)$. We use the notation $B \vdash e$ to state that there is a proof from B of e (or, in other words, that e is *derivable* from B).

Example. Let E be the system

$$f_1^1(x_1) = (x_1)'$$
$$f_1^2(x_1, x_2) = f_1^3(\bar{2}, x_2, f_1^1(x_1))$$

The principal letter of E is f_1^2; f_1^1 is an auxiliary letter, and f_1^3 an initial letter. The sequence of equations

$$f_1^2(x_1, x_2) = f_1^3(\bar{2}, x_2, f_1^1(x_1))$$
$$f_1^2(\bar{2}, x_2) = f_1^3(\bar{2}, x_2, f_1^1(\bar{2}))$$
$$f_1^2(\bar{2}, \bar{1}) = f_1^3(\bar{2}, \bar{1}, f_1^1(\bar{2}))$$
$$f_1^1(x_1) = (x_1)'$$
$$f_1^1(\bar{2}) = (\bar{2})' \quad (\text{i.e., } f_1^1(\bar{2}) = \bar{3})$$
$$f_1^2(\bar{2}, \bar{1}) = f_1^3(\bar{2}, \bar{1}, \bar{3})$$

is a proof of $f_1^2(\bar{2}, \bar{1}) = f_1^3(\bar{2}, \bar{1}, \bar{3})$ from E.

A number-theoretic partial function $\varphi(x_1, \ldots, x_n)$ is said to be *computable by a system* E of equations if and only if the principal letter of E is a letter f_j^n with n arguments, and, for any natural numbers $k_1, \ldots, k_n$, p,

$$E \vdash f_j^n(\bar{k}_1, \ldots, \bar{k}_n) = \bar{p} \text{ if and only if } \varphi(k_1, \ldots, k_n) = p.$$

The function φ is called *Herbrand-Gödel computable* (for short, HG-computable) if and only if there is some system E of equations by which φ is computable.

Examples.

1. Let E be the system $f_1^1(x_1) = 0$. Then E computes the zero function Z. Hence, Z is HG-computable.

2. Let E be the system $f_1^1(x_1) = (x_1)'$. Then E computes the successor function N. Hence, N is HG-computable.

3. Let E be the system $f_i^n(x_1, \ldots, x_n) = x_i$. Then E computes the projection function U_i^n. Hence, U_i^n is HG-computable.

4. Let E be the system

$$f_1^2(x_1, 0) = x_1$$
$$f_1^2(x_1, (x_2)') = (f_1^2(x_1, x_2))'$$

Then E computes the addition function.

5. Let E be the system

$$f_1^1(x_1) = 0$$
$$f_1^1(x_1) = x_1.$$

The function $\varphi(x_1)$ computed by E is the partial function with domain $\{0\}$ such that $\varphi(0) = 0$. For every $k \neq 0$, $E \vdash f_1^1(\bar{k}) = \bar{0}$ and $E \vdash f_1^1(\bar{k}) = \bar{k}$. Hence, $\varphi(x_1)$ is not defined for $x_1 \neq 0$.

EXERCISES

5.14. What functions are HG-computable by the following systems of equations?
(a) $f_1^1(0) = 0 \quad f_1^1((x_1)') = x_1$
(b) $f_1^2(x_1, 0) = x_1 \quad f_1^2(0, x_2) = 0 \quad f_1^2((x_1)', (x_2)') = f_1^2(x_1, x_2)$
(c) $f_1^1(x_1) = 0 \quad f_1^1(x_1) = 0'$
(d) $f_1^2(x_1, 0) = x_1 \quad f_1^2(x_1, (x_2)') = (f_1^2(x_1, x_2))' \quad f_1^1(f_1^2(x_1, x_1)) = 0$

5.15. Show that the following functions are HG-computable.
(a) $|x_1 - x_2|$
(b) $x_1 \cdot x_2$
(c) $\varphi(x) = \begin{cases} 0 & \text{when } x \text{ is even} \\ 1 & \text{when } x \text{ is odd} \end{cases}$.

PROPOSITION 5.16. *Every partial recursive function is* HG-*computable.*

PROOF.

(1) Examples 1–3 above have shown that the initial functions Z, N, U_i^n are HG-computable.

(2) Substitution (Rule IV). Let $\varphi(x_1, \ldots, x_n) = \eta(\psi_1(x_1, \ldots, x_n), \ldots, \psi_m(x_1, \ldots, x_n))$ where $\eta, \psi_1, \ldots, \psi_m$ have been shown to be HG-computable. Let E_i be a system of equations computing ψ_i, with principal letter f_i^n, and let E_{m+1} be a system of equations computing η, with principal letter f_{m+1}^m. By changing indices, we may assume that no two of $E_1, \ldots, E_m, E_{m+1}$ have any function letters in common. Construct a system E for φ by listing $E_1, \ldots, E_{m+1}$, and then adding the equation $f_{m+2}^n(x_1, \ldots, x_n) = f_{m+1}^m(f_1^n(x_1, \ldots, x_n), \ldots, f_m^n(x_1, \ldots, x_n))$. (We may assume that f_{m+2}^n does not occur in $E_1, \ldots, E_{m+1}$.) It is clear that, if $\varphi(k_1, \ldots, k_n) = p$, then $E \vdash f_{m+2}^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{p}$. Conversely, if $E \vdash f_{m+2}^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{p}$, then $E \vdash f_1^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{p}_1, \ldots, E \vdash f_m^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{p}_m$ and $E \vdash f_{m+1}^m(\overline{p}_1, \ldots, \overline{p}_m) = \overline{p}$. Hence, it readily follows that $E_1 \vdash f_1^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{p}_1, \ldots, E_m \vdash f_m^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{p}_m$ and $E_{m+1} \vdash f_{m+1}^n(\overline{p}_1, \ldots, \overline{p}_m) = \overline{p}$. Consequently, $\psi_1(k_1, \ldots, k_n) = p_1, \ldots, \psi_m(k_1, \ldots, k_n) = p_m$ and $\eta(p_1, \ldots, p_m) = p$. So, $\varphi(k_1, \ldots, k_n) = p$. (The details of this proof are left as an exercise. Hints may be found in Kleene [1952], Chapter XI, especially pages 262–270.) Hence φ is HG-computable.

(3) Recursion (Rule V). Let

$$\varphi(x_1, \ldots, x_n, 0) = \psi(x_1, \ldots, x_n)$$
$$\varphi(x_1, \ldots, x_n, (x_{n+1}) + 1) = \theta(x_1, \ldots, x_{n+1}, \varphi(x_1, \ldots, x_{n+1}))$$

where ψ and θ are HG-computable. Assume that E_1 is a system of equations computing ψ with principal letter f_1^n, and that E_2 is a system of equations computing θ, with principal letter f_1^{n+2}. Then form a system for computing φ by adding to E_1 and E_2

$$f_1^{n+1}(x_1, \ldots, x_n, 0) = f_1^n(x_1, \ldots, x_n)$$
$$f_1^{n+1}(x_1, \ldots, x_n, (x_{n+1})') = f_1^{n+2}(x_1, \ldots, x_{n+1}, f_1^{n+1}(x_1, \ldots, x_{n+1}))$$

(We assume that E_1 and E_2 have no function letters in common.) Clearly, if $\varphi(k_1, \ldots, k_n, k) = p$, then $E \vdash f_1^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{k}) = \overline{p}$. Conversely, one can prove easily by induction on k that, if $E \vdash f_1^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{k}) = \overline{p}$, then $\varphi(k_1, \ldots, k_n, k) = p$. Therefore, φ is HG-computable. (The case when the recursion has no parameters is even easier to handle, and is left as an exercise.)

(4) μ-operator (Rule VI). Let $\varphi(x_1, \ldots, x_n) = \mu y(\psi(x_1, \ldots, x_n, y) = 0)$ and assume that ψ is HG-computable by a system E_1 of equations with principal letter f_1^{n+1}. By Parts (1)–(3), we know that every primitive recursive function is HG-computable. In particular, multiplication is HG-computable; hence there is a system E_2 of equations, having no function letters in common with E_1, and with principal letter f_2^2 such that $E_2 \vdash f_2^2(\overline{k_1}, \overline{k_2}) = \overline{p}$ if and only if $k_1 \cdot k_2 = p$. We form a system E_3 by adding to E_1 and E_2 the equations

$$f_3^{n+1}(x_1, \ldots, x_n, 0) = 1$$
$$f_3^{n+1}(x_1, \ldots, x_n, (x_{n+1})') = f_2^2(f_3^{n+1}(x_1, \ldots, x_n, x_{n+1}), f_1^{n+1}(x_1, \ldots, x_n, x_{n+1}))$$

One can prove by induction that E_3 computes the function $\Pi_{y<z}\psi(x_1, \ldots, x_n, y)$, i.e., $E_3 \vdash f_3^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{k}) = \overline{p}$ if and only if $\Pi_{y<k}\psi(k_1, \ldots, k_n, y) = p$. Now construct the system E by adding to E_3 the equations

$$f_4^3((x_1)', 0, x_3) = x_3$$

$$f_3^n(x_1, \ldots, x_n) = f_4^3\big(f_3^{n+1}(x_1, \ldots, x_n, x_{n+1}), f_3^{n+1}(x_1, \ldots, x_n, (x_{n+1})'), x_{n+1}\big)$$

Then E computes the function $\varphi(x_1, \ldots, x_n) = \mu y(\psi(x_1, \ldots, x_n, y) = 0)$. For, if $\mu y(\psi(k_1, \ldots, k_n, y) = 0) = q$, then $E_3 \vdash f_3^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{q}) = \overline{p}'$, where $p + 1 = \Pi_{y<q}\psi(k_1, \ldots, k_n, y)$, and $E_3 \vdash f_3^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{q}') = 0$. Hence $E \vdash f_3^n(\overline{k_1}, \ldots, \overline{k_n}) = f_4^3(\overline{p}', 0, \overline{q})$. But, $E \vdash f_4^3(\overline{p}', 0, \overline{q}) = \overline{q}$, and so, $E \vdash f_3^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{q}$. Conversely, if $E \vdash f_3^n(\overline{k_1}, \ldots, \overline{k_n}) = \overline{q}$, then $E \vdash f_4^3(\overline{m}', 0, \overline{q}) = \overline{q}$, where $E_3 \vdash f_3^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{q}) = (\overline{m})'$ and $E_3 \vdash f_3^{n+1}(\overline{k_1}, \ldots, \overline{k_n}, \overline{q}') = 0$. Hence $\Pi_{y<q}\psi(k_1, \ldots, k_n, y) = m + 1 \neq 0$ and $\Pi_{y<q+1}\psi(k_1, \ldots, k_n, y) = 0$. So, $\psi(k_1, \ldots, k_n, y) \neq 0$ for $y < q$, and $\psi(k_1, \ldots, k_n, q) = 0$. Thus, $\mu y(\psi(k_1, \ldots, k_n, y) = 0) = q$. Therefore, φ is HG-computable.

We now shall proceed to show that every Herbrand-Gödel computable function is partial recursive, by means of an arithmetization of the apparatus of Herbrand-Gödel computability. We shall use the same arithmetization that was used for first-order theories (cf. Chapter 3, § 4). (We take the symbol $'$ to be an abbreviation for f_1^1. Remember that $r = s$ is an abbreviation for $A_1^2(r, s)$. The only individual constant is 0.) In particular (cf. pp. 153–154), the following relations and functions are primitive recursive:

FL(x): x is the Gödel number of a function letter.

$$(Ey)_{y<x}(Ez)_{z<x}(x = 9 + 8(2^y \cdot 3^z) \wedge y > 0 \wedge z > 0)$$

EVbl(x): x is the Gödel number of an expression consisting of a variable.
EFL(x): x is the Gödel number of an expression consisting of a function letter.
Nu(x): x is the Gödel number of a numeral.
Trm(x): x is the Gödel number of a term.
Num(x) = the Gödel number of the numeral $\overline{x}$.
$\mathrm{Arg}_T(x)$ = the number of arguments of a function letter f, if x is the Gödel number of f.
$x * y$ = the Gödel number of an expression AB if x is the Gödel number of the expression A and y is the Gödel number of the expression B.
Subst(a, b, u, v): v is the Gödel number of a variable x_i, u is the Gödel number of a term t, b is the Gödel number of an expression $\mathfrak{A}$, and a is the Gödel number of the result of substituting t for all occurrences of x_i in $\mathfrak{A}$.

The following are also primitive recursive.

Eqt(x): x is the Gödel number of an equation:

$$\mathrm{lh}(x) = 3 \wedge \mathrm{Trm}((x)_1) \wedge \mathrm{Trm}((x)_2) \wedge (x)_0 = 107$$

(Remember that = is A_1^2, whose Gödel number is 107.)

Syst(x): x is the Gödel number of a system of equations:

$$(y)_{y<\mathrm{lh}(x)}\mathrm{Eqt}((x)_y) \wedge \mathrm{FL}((((x)_{\mathrm{lh}(x)\dot{-}1})_1)_0)$$

Occ(u, v): u is the Gödel number of a term t or equation $\mathfrak{B}$ and v is the Gödel number of a term which occurs in t or $\mathfrak{B}$.

$$(\mathrm{Trm}(u) \vee \mathrm{Eqt}(u)) \wedge \mathrm{Trm}(v) \wedge (\mathrm{Ex})_{x<u}(\mathrm{Ey})_{y<u}(u = x * v * y \vee u = x * v \vee u = v * y \vee u = v)$$

$\mathrm{Cons}_1(u, v)$: u is the Gödel number of an equation e_1, and v is the Gödel number of an equation e_2, and e_2 is a consequence of e_1 by Rule R_1:

$$\mathrm{Eqt}(u) \wedge \mathrm{Eqt}(v) \wedge (\mathrm{Ex})_{x<u}(\mathrm{Ey})_{y<v}(\mathrm{Nu}(y) \wedge \mathrm{Subst}(v, u, y, x) \wedge \mathrm{Occ}(u, x))$$

$\mathrm{Cons}_2(u, z, v)$: u, z, v are Gödel numbers of equations e_1, e_2, e_3, respectively, and e_3 is a consequence of e_1 and e_2 by Rule R_2.

$$\begin{aligned}&\mathrm{Eqt}(u) \wedge \mathrm{Eqt}(z) \wedge \mathrm{Eqt}(v) \wedge \sim(\mathrm{Ex})_{x\le z}(\mathrm{EVbl}(x) \wedge \mathrm{Occ}(z, x)) \wedge\\ &\mathrm{FL}(((z)_1)_0) \wedge (x)_{0<x<\mathrm{lh}((z)_1)} \sim \mathrm{FL}(((z)_1)_x) \wedge\\ &(x)_{x<\mathrm{lh}((z)_2)} \sim \mathrm{FL}(((z)_2)_x) \wedge \mathrm{Occ}((u)_2, (z)_1) \wedge ((\mathrm{Ey})_{y<u}\\ &(\mathrm{Ew})_{w<u}((u)_2 = y * (z)_1 * w \wedge v = 2^{107}3^{(u)_1}5^{y*(z)_2*w}) \vee\\ &((u)_2 = (z)_1 \wedge v = 2^{107}3^{(u)_1}5^{(z)_2})))\end{aligned}$$

Ded(u, z): u is the Gödel number of a system of equations E, and z is the Gödel number of a proof from E.

$$\begin{aligned}&\mathrm{Syst}(u) \wedge (x)_{x<\mathrm{lh}(z)}((\mathrm{Ew})_{w<\mathrm{lh}(u)}(u)_w = (z)_x \vee\\ &(\mathrm{Ey})_{y<x}\mathrm{Cons}_1((z)_y, (z)_x) \vee (\mathrm{Ey})_{y<x}(\mathrm{Ev})_{v<x}\mathrm{Cons}_2((z)_y, (z)_v, (z)_x))\end{aligned}$$

$S_n(u, x_1, \ldots, x_n, z)$: u is the Gödel number of a system of equations E whose principal letter is of the form f_j^n, and z is the Gödel number of a proof from E of an equation of the form $f_j^n(\bar{x}_1, \ldots, \bar{x}_n) = \bar{p}$.

$$\begin{aligned}&\mathrm{Ded}(u, z) \wedge \mathrm{Arg}_T((((u)_{\mathrm{lh}(u)\dot{-}1})_1)_0 = n \wedge (((z)_{\mathrm{lh}(z)\dot{-}1})_1)_0 =\\ &(((u)_{\mathrm{lh}(u)\dot{-}1})_1)_0 \wedge (y)_{0<y<\mathrm{lh}(((z)_{\mathrm{lh}(z)\dot{-}1})_1)} \sim \mathrm{FL}((((z)_{\mathrm{lh}(z)\dot{-}1})_1)_y)\\ &\wedge \mathrm{Nu}(((z)_{\mathrm{lh}(z)\dot{-}1})_2) \wedge ((z)_{\mathrm{lh}(z)\dot{-}1})_1 = 2^{(((u)_{\mathrm{lh}(u)\dot{-}1})_1)_0} * 2^3 * 2^{\mathrm{Num}(x_1)} * 2^7 *\\ &2^{\mathrm{Num}(x_2)} * 2^7 * \cdots * 2^7 * 2^{\mathrm{Num}(x_n)} * 2^5\end{aligned}$$

Remember that g(( ) = 3, g( )) = 5, g(,) = 7.

$U(x) = \mu y_{y<x}(\mathrm{Num}(y) = ((x)_{1h(x) \dot{-} 1})_2)$. (If x is the Gödel number of a proof of an equation $r = \bar{p}$, then $U(x) = p$.)

PROPOSITION 5.17 (Kleene [1936a]). *If* $\varphi(x_1, \ldots, x_n)$ *is* HG-*computable by a system of equations* E *with Gödel number* e, *then*

$$\varphi(x_1, \ldots, x_n) = U(\mu y(S_n(e, x_1, \ldots, x_n, y)))$$

Hence, every HG-*computable function* φ *is partial recursive, and, if* φ *is total, then* φ *is recursive.*

PROOF. $\varphi(k_1, \ldots, k_n) = p$ if and only if $E \vdash f_j^n(\overline{k_1}, \ldots, \overline{k_n}) = \bar{p}$, where f_j^n is the principal letter of E. $\varphi(k_1, \ldots, k_n)$ is defined if and only if $(Ey)S_n(e, k_1, \ldots, k_n, y)$. If $\varphi(k_1, \ldots, k_n)$ is defined, $\mu y(S_n(e, k_1, \ldots, k_n, y))$ is the Gödel number of a proof from E of an equation $f_j^n(\overline{k_1}, \ldots, \overline{k_n}) = \bar{p}$. Hence, $U(\mu y(S_n(e, k_1, \ldots, k_n, y))) = p = \varphi(k_1, \ldots, k_n)$. Also, since S_n is primitive recursive, $\mu y(S_n(e, x_1, \ldots, x_n, y))$ is partial recursive. If φ is total, then $(x_1) \ldots (x_n)(Ey)S_n(e, x_1, \ldots, x_n, y)$; hence, $\mu y(S_n(e, x_1, \ldots, x_n, y))$ is recursive, and then, so is $U(\mu y(S_n(e, x_1, \ldots, x_n, y)))$.

Thus, the class of Herbrand-Gödel computable functions is identical with the class of partial recursive functions. This is further evidence for Church's Thesis.

It is sometimes more convenient to use instead of S_n the primitive recursive predicate

$$T_n(z, x_1, \ldots, x_n, y): S_n(z, x_1, \ldots, x_n, y) \wedge (u)_{u<y} \sim S_n(z, x_1, \ldots, x_n, u)$$

Clearly, if $T_n(z, x_1, \ldots, x_n, y)$, then $S_n(z, x_1, \ldots, x_n, y)$. In addition, in contrast to S_n, if $T_n(z, x_1, \ldots, x_n, y)$ and $T_n(z, x_1, \ldots, x_n, v)$, then $y = v$. It is obvious that

$$(Ey)S_n(z, x_1, \ldots, x_n, y) \equiv (Ey)T_n(z, x_1, \ldots, x_n, y)$$

and

$$U(\mu y S_n(z, x_1, \ldots, x_n, y)) = U(\mu y T_n(z, x_1, \ldots, x_n, y))$$

whenever either side is defined. From Propositions 5.16 and 5.17, it follows that every partial recursive function is expressible in the form $U(\mu y T_n(e, x_1, \ldots, x_n, y))$ where e is the Gödel number of a system of equations computing the function. Conversely, for any natural number e, $U(\mu y T_n(e, x_1, \ldots, x_n, y))$ is a partial recursive function. Thus, as z varies over all natural numbers, $U(\mu y T_n(z, x_1, \ldots, x_n, y))$ gives an enumeration (with repetitions) of *all* partial recursive functions of n arguments. A number e such that $\varphi(x_1, \ldots, x_n) = U(\mu y T_n(e, x_1, \ldots, x_n, y))$ is called an *index* of the function φ. The Gödel number of any system of equations computing φ is an index of φ; there are infinitely many indices of φ. (Exercise.)

By an index of a recursive relation R we mean an index of the characteristic function of R. Then

$$R(x_1, \ldots, x_n) \equiv (Ey)(T_n(e, x_1, \ldots, x_n, y) \wedge U(y) = 0)$$

where e is an index of R.

LEMMA 5.18

(1) *For* $n > 0$, *if* $R(x_1, \ldots, x_n, y)$ *is a recursive predicate, then there exist natural numbers* e_1, e_2 *such that*

$$(Ey)R(x_1, \ldots, x_n, y) \equiv (Ey)T_n(e_1, x_1, \ldots, x_n, y)$$

and

$$(y)R(x_1, \ldots, x_n, y) \equiv (y) \sim T_n(e_2, x_1, \ldots, x_n, y)$$

(2) *For* $n > 0$, *if* $R(x_1, \ldots, x_n, z, y)$ *is a recursive predicate, there exist natural numbers* e_3, e_4 *such that*

$$(z)(Ey)R(x_1, \ldots, x_n, z, y) \equiv (z)(Ey)T_{n+1}(e_3, x_1, \ldots, x_n, z, y)$$

and

$$(Ez)(y)R(x_1, \ldots, x_n, z, y) \equiv (Ez)(y) \sim T_{n+1}(e_4, x_1, \ldots, x_n, z, y)$$

and so on, for three or more quantifiers.

PROOF.

(1) Let $\varphi(x_1, \ldots, x_n, y)$ be the characteristic function of R; then φ is recursive, and $\mu y(\varphi(x_1, \ldots, x_n, y) = 0)$ is partial recursive. Let e_1 be the Gödel number of a system of equations computing $\mu y(\varphi(x_1, \ldots, x_n, y) = 0)$. Then, $(Ey)R(x_1, \ldots, x_n, y)$ if and only if $\mu y(\varphi(x_1, \ldots, x_n, y) = 0)$ is defined; hence, $(Ey)R(x_1, \ldots, x_n, y) \equiv (Ey)(T_n(e_1, x_1, \ldots, x_n, y))$. Applying this result to $\sim R$, we obtain a number e_2 such that $(Ey) \sim R(x_1, \ldots, x_n, y) \equiv (Ey)T_n(e_2, x_1, \ldots, x_n, y)$. Hence, $(y)R(x_1, \ldots, x_n, y) \equiv (y) \sim T_n(e_2, x_1, \ldots, x_n, y)$. (2) follows from (1), taking $n + 1$ instead of n.

Thus, as u varies, $(Ey)T_n(u, x_1, \ldots, x_n, y)$ enumerates all relations $(Ey)R(x_1, \ldots, x_n, y)$, where R is recursive, and $(y) \sim T_n(u, x_1, \ldots, x_n, y)$ enumerates all relations $(y)R(x_1, \ldots, x_n, y)$, where R is recursive; etc.

PROPOSITION 5.19 (Kleene [1943; 1952, § 57], Mostowski [1947])

(1) *If* $R(x, y)$ *is recursive, there are natural numbers* e_1, e_2 *such that*

$$\sim ((Ey)R(e_1, y) \equiv (y) \sim T_1(e_1, e_1, y))$$

and

$$\sim ((y)R(e_2, y) \equiv (Ey)T_1(e_2, e_2, y))$$

(2) *If* R(x) *is recursive, there are natural numbers* e_1, e_2 *such that*

$$\sim (R(e_1) \equiv (y) \sim T_1(e_1, e_1, y))$$

and

$$\sim (R(e_2) \equiv (Ey)T_1(e_2, e_2, y))$$

(3) *Both* $(y) \sim T_1nx, x, y)$ *and* $(Ey)T_1(x, x, y)$ *are not recursive.*

(4) *Consider the following list* (where R is any recursive relation):

$$(R(x_1, \ldots, x_n) \quad \begin{matrix} (Ey_1)R(x_1, \ldots, x_n, y_1) & (Ey_1)(y_2)R(x_1, \ldots, x_n, y_1, y_2) \\ (y_1)R(x_1, \ldots, x_n, y_1) & (y_1)(Ey_2)R(x_1, \ldots, x_n, y_1, y_2) \end{matrix}$$

$$\begin{matrix} (Ey_1)(y_2)(Ey_3)R(x_1, \ldots, x_n, y_1, y_2, y_3) \ldots \\ (y_1)(Ey_2)(y_3)R(x_1, \ldots, x_n, y_1, y_2, y_3) \ldots \end{matrix}$$

If we let $\Pi_0^n = \Sigma_0^n =$ the set of all recursive relations with n arguments; and, for $k > 0$, $\Sigma_k^n =$ the set of all relations with n arguments expressible in the "prenex form" $(Ey_1)(y_2) \ldots (Qy_k)R(x_1, \ldots, x_n, y_1, y_2, \ldots, y_k)$, consisting of k alternating quantifiers beginning with an existential quantifier and followed by a recursive relation R; and $\Pi_k^n =$ the set of all relations with n arguments expressible in "prenex form" $(y_1)(Ey_2) \ldots (Qy_k)R(x_1, \ldots, x_n, y_1, y_2, \ldots, y_k)$, consisting of k alternating quantifiers beginning with a universal quantifier and followed by a recursive relation R, then the list above can be written

$$\Sigma_0^n \quad \begin{matrix} \Sigma_1^n & \Sigma_2^n & \Sigma_3^n & \ldots \\ \Pi_1^n & \Pi_2^n & \Pi_3^n & \ldots \end{matrix}$$

(In the "prenex form," (Qy_k) represents either a universal or existential quantifier.)

(a) Every relation of any form listed above is expressible in any form indicated in any of the succeeding columns on the right, i.e., $\Sigma_k^n \subseteq \Sigma_j^n \cap \Pi_j^n$ and $\Pi_k^n \subseteq \Sigma_j^n \cap \Pi_j^n$ for all $j > k$.

(b) There is a relation of each form, except the left-most, which is not expressible in the other form indicated in the same column, and, hence, by (a), not in any of the previous columns on the left, i.e., $\Sigma_k^n - \Pi_k^n \neq 0$ and $\Pi_k^n - \Sigma_k^n \neq 0$ for $k > 0$.

(c) Every arithmetic relation (cf. p. 151, Exercise 3.34) is expressible in at least one of these forms.

(d) (Post) For any relation $Q(x_1, \ldots, x_n)$, Q is recursive if and only if both Q and $\sim Q$ are both expressible in the form $(Ey_1)R(x_1, \ldots, x_n, y_1)$, where R is recursive, i.e., $\Sigma_1^n \cap \Pi_1^n = \Sigma_0^n$.

(e) If $Q_1 \in \Sigma_k^n$ and $Q_2 \in \Sigma_k^n$, then $Q_1 \vee Q_2$ and $Q_1 \wedge Q_2$ are in Σ_k^n; if Q_1 and Q_2 are in Π_k^n, then $Q_1 \vee Q_2$ and $Q_1 \wedge Q_2$ are in Π_k^n.

(f) In contradistinction to (d), if $k > 0$,

$$(\Sigma_{k+1}^n \cap \Pi_{k+1}^n) - (\Sigma_k^n \cup \Pi_k^n) \neq 0$$

PROOF.

(1) Assume R(x, y) recursive. Then, by Lemma 5.18, there are numbers e_1, e_2 such that $(Ey)R(x, y) \equiv (Ey)T_1(e_1, x, y)$ and $(y)R(x, y) \equiv (y) \sim T_1(e_2, x, y)$. In the first equivalence, let $x = e_1$, and, in the second, let $x = e_2$.

(2) Assume R(x) recursive. Then $R(x) \wedge y = y$ is recursive; clearly, $(Ey)(R(x) \wedge y = y) \equiv R(x)$ and $(y)(R(x) \wedge y = y) \equiv R(x)$. Apply (1).

(3) Assume $(y) \sim T_1(x, x, y)$ is recursive. By (2), there is an integer e_1 such that $\sim((y) \sim T_1(e_1, e_1, y) \equiv (y) \sim T_1(e_1, e_1, y))$, which is a contradiction. Similarly, if $(Ey)T_1(x, x, y)$ is recursive, then, by (2), there is an integer e_2 such that $\sim((Ey)T_1(e_2, e_2, y) \equiv (Ey)T_1(e_2, e_2, y))$, which is a contradiction.

(4) (a) $(Ez_1)(y_1)(Ez_2)(y_2) \ldots (Ez_k)(y_k)R(x_1, \ldots, x_n, z_1, y_1, \ldots, z_k, y_k) \equiv (u)(Ez_1)(y_1) \ldots (Ez_k)(y_k)(R(x_1, \ldots, x_n, z_1, y_1, \ldots, z_k, y_k) \wedge u = u) \equiv (Ez_1)(y_1) \ldots (Ez_k)(y_k)(Eu)(R(x_1, \ldots, x_n, z_1, y_1, \ldots, z_k, y_k) \wedge u = u)$. Hence any relation expressible in one of the forms in the list is expressible in both forms in any succeeding column.

(b) Let us just take a typical case. Consider $(Ev)(z)(Ey)T_{n+2}(x_1, x_1, x_2, \ldots, x_n, v, z, y)$. Assume that this is expressible in the form $(v)(Ez)(y)R(x_1, \ldots, x_n, v, z, y)$, where R is recursive. By Lemma 5.18, this relation is equivalent to $(v)(Ez)(y) \sim T_{n+2}(e, x_1, \ldots, x_n, v, z, y)$ for some e. But when $x_1 = e$, this is a contradiction.

(c) Every wf of the first-order theory S can be put into prenex normal form. It suffices to note that, if R is recursive, then $(Eu)(Ev)R(u, v)$ is equivalent to $(Ez)R(\sigma_1^2(z), \sigma_2^2(z))$, where σ_1^2, σ_2^2 are the recursive inverse mappings of the one-one correspondence σ^2 between pairs of natural numbers and natural numbers (cf. 145–146). Also, $(u)(v)R(u, v)$ is equivalent to $(z)R(\sigma_1^2(z), \sigma_2^2(z))$. Hence, successive quantifiers of the same kind (existential or universal) can be condensed into one such quantifier.

(d) If Q is recursive, so is $\sim Q$; if $P(x_1, \ldots, x_n)$ is recursive, then $P(x_1, \ldots, x_n) \equiv (Ey)(P(x_1, \ldots, x_n) \wedge y = y)$. Conversely, assume Q is expressible as $(Ey)R_1(x_1, \ldots, x_n, y)$, and $\sim Q$ as $(Ey)R_2(x_1, \ldots, x_n, y)$, where R_1 and R_2 are recursive. Hence, $(x_1) \ldots (x_n)(Ey)(R_1(x_1, \ldots, x_n, y) \vee R_2(x_1, \ldots, x_n, y))$. So, $\varphi(x_1, \ldots, x_n) = \mu y(R_1(x_1, \ldots, x_n, y) \vee R_2(x_1, \ldots, x_n, y))$ is recursive. Then $Q(x_1, \ldots, x_n) \equiv R_1(x_1, \ldots, x_n, \varphi(x_1, \ldots, x_n))$, and, therefore, Q is recursive.

(e) Use the following facts; If x is not free in $\mathfrak{A}$, $\vdash (Ex)(\mathfrak{A} \vee \mathfrak{B}) \equiv (\mathfrak{A} \vee (Ex)\mathfrak{B})$, $\vdash (Ex)(\mathfrak{A} \wedge \mathfrak{B}) \equiv (\mathfrak{A} \wedge (Ex)\mathfrak{B})$, $\vdash (x)(\mathfrak{A} \vee \mathfrak{B}) \equiv (\mathfrak{A} \vee (x)\mathfrak{B})$, $\vdash (x)(\mathfrak{A} \wedge \mathfrak{B}) \equiv (\mathfrak{A} \wedge (x)\mathfrak{B})$.

(f) We shall suggest here a proof in the case n = 1; the other cases are then easy consequences. Let $Q(x) \in \Sigma^1_k - \Pi^1_k$. Define P(x) as $(Ez)((x = 2z \wedge Q(z)) \vee (x = 2z + 1 \wedge \sim Q(z)))$. It is easy to prove that $P \notin \Sigma^1_k \cup \Pi^1_k$ and that $P \in \Sigma^1_{k+1}$. To show that $P \in \Pi^1_{k+1}$, note that P(x) holds if and only if

$$(Ez)(x = 2z \wedge Q(z)) \vee ((Ez_{z<x}(x = 2z + 1) \wedge (z)(x = 2z + 1 \supset \sim Q(z)))$$

(Cf. Rogers [1959]).

EXERCISES

5.16. This exercise will show the existence of a recursive, non-primitive recursive function.

1. Let $[\sqrt{n}\,]$ be the largest integer $\leqslant \sqrt{n}$. Show that $[\sqrt{n}\,]$ is defined by the recursion

$$\kappa(0) = 0$$
$$\kappa(n + 1) = \kappa(n) + \overline{sg}\,|(n + 1) - (\kappa(n) + 1)^2|$$

Hence, $[\sqrt{n}\,]$ is primitive recursive.

2. The function $\text{Quadrem}(n) = n \dot{-} [\sqrt{n}\,]^2$ is primitive recursive and represents the difference between n and the largest square $\leqslant n$.

3. Let $\rho(x, y) = ((x + y)^2 + y)^2 + x$; $\rho_1(n) = \text{Quadrem}(n)$ and $\rho_2(n) = \text{Quadrem}([\sqrt{n}\,])$. These functions are primitive recursive. Prove:

(a) $\rho_1(\rho(x, y)) = x$ and $\rho_2(\rho(x, y)) = y$.

(b) $\rho(\rho_1(n), \rho_2(n)) = n$.

(c) ρ is a one-one function from ω^2 onto ω.

(d) $\rho_1(0) = \rho_2(0) = 0$ and

$$\left.\begin{aligned}\rho_1(n + 1) &= \rho_1(n) + 1\\ \rho_2(n + 1) &= \rho_2(n)\end{aligned}\right\} \text{ if } \rho_1(n + 1) \neq 0$$

(e) Define for each $n \geqslant 3$, $\rho^n(x_1, \ldots, x_n) = \rho(\rho^{n-1}(x_1, \ldots, x_{n-1}), x_n)$. Let $\rho^2 = \rho$. Then each ρ^n is primitive recursive. Define $\rho^n_i(k) = \rho^{n-1}_i(\rho_1(k))$ for $1 \leqslant i \leqslant n - 1$, and $\rho^n_n(k) = \rho_2(k)$. Then each $\rho^n_i (1 \leqslant i \leqslant n)$ is primitive recursive, $\rho^n_i(\rho^n(x_1, \ldots, x_n)) = x_i$ and $\rho^n(\rho^n_1(k), \rho^n_2(k), \ldots, \rho^n_n(k)) = k$. Hence, ρ^n is a one-one mapping of ω^n onto ω, and the ρ^n_i's are the corresponding "inverse" functions. The ρ^n's and ρ^n_i's are obtained from ρ, ρ_1, ρ_2 by substitution.

4. The recursion rule (V) (cf. page 138) can be limited to the form

$$\psi(x_1, \ldots, x_{n+1}, 0) = x_{n+1} \qquad (n \geqslant 0)$$

$$\psi(x_1, \ldots, x_{n+1}, y + 1) = \varphi(x_1, \ldots, x_{n+1}, y, \psi(x_1, \ldots, x_{n+1}, y))$$

Suggestion: given

$$\theta(x_1, \ldots, x_n, 0) = \gamma(x_1, \ldots, x_n)$$

$$\theta(x_1, \ldots, x_n, y + 1) = \delta(x_1, \ldots, x_n, y, \theta(x_1, \ldots, x_n, y))$$

Define ψ as above, letting $\varphi(x_1, \ldots, x_{n+1}, y, z) = \delta(x_1, \ldots, x_n, y, z)$. Then $\theta(x_1, \ldots, x_n, y) = \psi(x_1, \ldots, x_n, \gamma(x_1, \ldots, x_n), y)$.

5. Assuming ρ, ρ_1, ρ_2 as additional initial functions, we can limit uses of the recursion rule (V) to the one-parameter form:

$$\psi(x, 0) = \alpha(x)$$
$$\psi(x, y + 1) = \beta(x, y, \psi(x, y))$$

Hint: Let $n \geqslant 2$. Given

$$\theta(x_1, \ldots, x_n, 0) = \gamma(x_1, \ldots, x_n)$$
$$\theta(x_1, \ldots, x_n, y + 1) = \delta(x_1, \ldots, x_n, y, \theta(x_1, \ldots, x_n, y)$$

Let $\eta(u, y) = \theta(\rho_1^n(u), \ldots, \rho_n^n(u), y)$. Define η by a permissible recursion.

6. Assuming ρ, ρ_1, ρ_2 as additional initial functions, we can use $\delta(y, \psi(x, y))$ instead of $\beta(x, y, \psi(x, y))$ in Part (5). (Hint: given

$$\psi(x, 0) = \alpha(x)$$
$$\psi(x, y + 1) = \beta(x, y, \psi(x, y))$$

let $\psi_1(x, y) = \rho(x, \psi(x, y))$. Then $x = \rho_1(\psi_1(x, y))$ and $\psi(x, y) = \rho_2(\psi_1(x, y))$. Define ψ_1 by an appropriate recursion.)

7. Assuming ρ, ρ_1, ρ_2 as additional initial functions, we can limit uses of the recursion rule (V) to the form

$$\psi(x, 0) = x$$
$$\psi(x, y + 1) = \beta(y, \psi(x, y))$$

Hint: use Part (6). Given

$$\varphi(x, 0) = \alpha(x)$$
$$\varphi(x, y + 1) = \beta(y, \varphi(x, y))$$

Define ψ as above. Then $\varphi(x, y) = \psi(\alpha(x), y)$.

8. Assuming ρ, ρ_1, ρ_2, $+, \cdot$, sg, as additional initial functions, we can limit all uses of the recursion rule (V) to those with one parameter of the form

$$f(0) = 0$$
$$f(y + 1) = h(y, f(y))$$

Hint: given, by Part (7),

$$\psi(x, 0) = x$$
$$\psi(x, y + 1) = \beta(y, \psi(x, y))$$

Let $f(n) = \psi(\rho_2(n), \rho_1(n))$. Then

$$f(0) = \psi(\rho_2(0), \rho_1(0)) = \psi(0, 0) = 0$$

$$f(n + 1) = \psi(\rho_2(n + 1), \rho_1(n + 1))$$

$$= \begin{cases} \rho_2(n + 1) \text{ if } \rho_1(n + 1) = 0 \\ \beta(\rho_1(n + 1) \dot{-} 1, \psi(\rho_2(n + 1), \rho_1(n + 1) \dot{-} 1)) \text{ if } \rho_1(n + 1) \neq 0 \end{cases}$$

$$= \begin{cases} \rho_2(n + 1) \text{ if } \rho_1(n + 1) = 0 \\ \beta(\rho_1(n), \psi(\rho_1(n), \rho_2(n))) \text{ if } \rho_1(n + 1) \neq 0 \end{cases}$$

$$= \begin{cases} \rho_1(n + 1) \text{ if } \rho_2(n + 1) = 0 \\ \beta(\rho_1(n), f(n)) \text{ if } \rho_1(n + 1) \neq 0 \end{cases}$$

$$= \rho_2(n + 1) \cdot \overline{\text{sg}}(\rho_1(n + 1)) + \beta(\rho_1(n), f(n)) \cdot \text{sg}(\rho_1(n + 1))$$

$$= h(n, f(n))$$

(Note that sg is obtainable by a recursion of the appropriate kind.)

Then $\psi(x, y) = f(\rho(y, x))$.

9. All primitive recursive functions are obtainable from the initial functions Z, N, U_i^n, ρ, ρ_1, ρ_2, $+, \cdot$, $\overline{sg}$ by substitution and the recursion rule (V) in the form

$$f(0) = 0$$
$$f(y + 1) = h(y, f(y))$$

(Restatement of Part (8).)

10. In Part (9), h(y, f(y)) can be replaced by h(f(y)). Hint: given

$$f(0) = 0$$
$$f(y + 1) = h(y, f(y))$$

Let $g(u) = \rho(u, f(u))$, and $\varphi(w) = \rho(\rho_1(w) + 1, h(\rho_1(w), \rho_2(w)))$. Then

$$g(0) = 0$$
$$g(y + 1) = \varphi(g(y))$$

and

$$f(u) = \rho_2(g(u))$$

11. Show that the equations

$$\psi(n, 0) = n + 1$$
$$\psi(0, m + 1) = \psi(1, m)$$
$$\psi(n + 1, m + 1) = \psi(\psi(n, m + 1), m)$$

define a recursive function. (Hint: show that ψ is Herbrand-Gödel computable by the given equations, and then use Proposition 5.17.) In addition, prove:

(I) $\psi(n, m) > n$.

(II) ψ is monotonic in each variable, i.e., if $x < z$, then $\psi(x, y) < \psi(z, y)$ and $\psi(y, x) < \psi(y, z)$.

(III)' $\psi(n, m + 1) \geqslant \psi(n + 1, m)$.

(IV) For every primitive recursive function $f(x_1, \ldots, x_n)$, there is some fixed m such that $f(x_1, \ldots, x_n) < \psi(\max(x_1, \ldots, x_n), m)$ for all $x_1, \ldots, x_n$. (Hint: prove this first for the initial functions Z, N, U_i^n, ρ, ρ_1, ρ_2, $+, \cdot$, $\overline{sg}$, and then show that it is preserved by substitution and the recursion of Part (10) above.) Hence, for every primitive recursive function f(x) of one argument, there is some m such that $f(x) < \psi(x, m)$ for all x.

(V) Prove that $\psi(x, x) + 1$ is recursive, but not primitive recursive. (Hint: Part (IV).)

For other proofs of the existence of recursive, non-primitive recursive functions, cf. Ackermann [1928], Péter [1935, 1951], R. Robinson [1948].

A very important metamathematical notion is that of recursively enumerable set. A set of natural numbers is called *recursively enumerable* (r.e.) if and only if it is either empty or the range of a recursive function. Intuitively, if we accept Church's Thesis, then a recursively enumerable set is a collection of natural numbers which is generated by some mechanical process.

PROPOSITION 5.20

(1) *A set* B *is* r.e. *if and only if* $x \in B$ *is expressible in the form* $(Ey)R(x, y)$, *where* R *is recursive.* (*We can also allow* R *here to be primitive recursive.*)

(2) *A set* B *is* r.e. *if and only if it is empty or the range of a partial recursive function* (*or of a primitive recursive function*).

(3) *A set* B *is* r.e. *if and only if it is the domain of definition of a partial recursive function.*

(4) *A set* B *is recursive if and only if both* B *and its complement* $\bar{B}$† *are* r.e.

(5) *The set* $\{x|(Ey)T_1(x, x, y)\}$ *is* r.e., *but not recursive.*‡

PROOF.

(1) Assume B is r.e. If B is empty, then $x \in B \equiv (Ey)(x \neq x \wedge y \neq y)$. If B is non-empty, it is the range of a recursive function φ. Then $x \in B \equiv (Ey)(\varphi(y) = x)$. Conversely, assume $x \in B \equiv (Ey)R(x, y)$. If B is empty, B is r.e. If B is non-empty, let k be a fixed element of B. Define:

$$\theta(z) = \begin{cases} k & \text{if } \sim R((z)_0, (z)_1) \\ (z)_0 & \text{if } R((z)_0, (z)_1) \end{cases}$$

Clearly, B is the range of θ, and θ is recursive. (By Lemma 5.18, if R is recursive, $(Ey)R(x, y) \equiv (Ey)T_1(e, x, y)$ for some e; but $T_1(e, x, y)$ is primitive recursive.)

(2) Assume B is the range of a partial recursive function φ. If B is empty, then B is r.e. If B is non-empty, let k be a fixed element of B. Now, there is a number e such that $\varphi(x) = U(\mu y T_1(e, x, y))$. Let

$$\theta(z) = \begin{cases} U((z)_1) & \text{if } T_1(e, (z)_0, (z)_1) \\ k & \text{if } \sim T_1(e, (z)_0, (z)_1) \end{cases}$$

Then θ is primitive recursive and B is the range of θ. Hence, B is r.e. This proof also shows that every non-empty r.e. set is the range of a primitive recursive function.

(3) Assume B r.e. If B is empty, B is the domain of the partial recursive function $\mu y(x + y + 1 = 0)$. If B is non-empty, B is the range of a recursive function f. Let g be the partial recursive function such that $g(y) = \mu x(f(x) = y)$. Then B is the domain of g. Conversely, assume B is the domain of a partial recursive function φ. Then there is a number e such that $\varphi(x) = U(\mu y T_1(e, x, y))$. Hence $\varphi(x) = z \equiv (Ey)(T_1(e, x, y) \wedge U(y) = z)$. But $x \in B \equiv (Ez)(\varphi(x) = z)$. So, $x \in B$ if and only if $(Ez)(Ey)(T_1(e, x, y) \wedge U(y) = z)$, and the latter is equivalent to $(Eu)(T_1(e, x, (u)_1) \wedge U((u)_1) = (u)_0)$; moreover, $T_1(e, x, (u)_1) \wedge U((u)_1) = (u)_0$ is recursive. Thus, by (1), B is r.e.

(4) From (1) and Proposition 5.19(4(d)). (The intuitive meaning of Part (4) is the following: if there are mechanical procedures for generating B and $\bar{B}$, then to

†I.e., $\omega - B$, where ω is the set of non-negative integers.

‡Remember that $\{x|P(x)\}$ stands for the set of all x such that P(x) holds.

determine whether any number n is in B we need only wait until n is generated by one of the machines and then observe which machine produced it.)

(5) From (1) and (4), and Proposition 5.19(3).

EXERCISES

Prove:

5.17. The inverse image of an r.e. set under a recursive function is r.e. (i.e., if f is recursive and B r.e., then $\{x|f(x) \in B\}$ is r.e.). The inverse image of a recursive set under a recursive function is recursive. The image of an r.e. set under a recursive function is r.e., but the image of a recursive set under a recursive function is not necessarily a recursive set.

5.18. An infinite set is recursive if and only if it is the range of a strictly increasing recursive function. (g is *strictly increasing* if $x < y$ implies $g(x) < g(y)$.)

5.19. Any infinite set is r.e. if and only if it is the range of a one-one recursive function.

5.20. Every infinite r.e. set contains an infinite recursive subset.

5.21. If A and B are r.e. sets, so are $A \cup B$ and $A \cap B$, but there exists an r.e. set A such that $\omega - A$ is not r.e.

5.22. Show that the assertion

> (‡) A set B is r.e. if and only if B is effectively enumerable (i.e., there is a mechanical procedure for generating all the elements of B)

is equivalent to Church's Thesis.

5.23. Let K be a first-order theory with equality containing all the symbols of formal number theory S. A relation $B(x_1, \ldots, x_n)$ is said to be *weakly expressible* in K if and only if there is a wf $\mathfrak{B}(x_1, \ldots, x_n)$ of K such that, for any natural numbers $k_1, \ldots, k_n$, $B(k_1, \ldots, k_n)$ if and only if $\vdash_K \mathfrak{B}(\bar{k}_1, \ldots, \bar{k}_n)$.

(a) If K is consistent, show that every relation expressible in K is weakly expressible in K.

(b) If every recursive relation is expressible in K and K is ω-consistent, prove that every r.e. set is weakly expressible in K. (Remember that, when we refer here to an r.e. set B, we mean the corresponding relation "$x \in B$".)

(c) If K is such that the relations (a)–(d) of Propositions 3.25–3.26 are recursive, prove that any set which is weakly expressible in K is r.e.

(d) If formal number theory S is ω-consistent, prove that a set B is r.e. if and only if B is weakly expressible in S.

5.24. (a) (Craig [1953]) Let K be a first-order theory such that the set T_k of Gödel numbers of theorems of K is r.e. Show that K is recursively axiomatizable.

(b) For any wf $\mathfrak{A}$ of formal number theory S, let $\mathfrak{A}\sharp$ represent its translation into axiomatic set theory NBG (cf. p. 204). Let K be the set of wfs $\mathfrak{A}$ such that $\vdash_{NBG} \mathfrak{A}\sharp$. Prove that K is a (proper) recursively axiomatizable extension of S. (However, no "natural" set of axioms for K is known.)

By Proposition 5.20(3), a set is r.e. if and only if it is the domain ζ_n of the partial recursive function $U(\mu y T_1(n, x, y))$ for some n; hence, $x \in \zeta_n$ if and only if $(Ey)T_1(n, x, y)$. We call n an *index* of the r.e. set ζ_n. We thus have an enumeration (with repetitions) $\zeta_0, \zeta_1, \ldots$ of all r.e. sets.

An example of an r.e. set which is not recursive is the set of all x such that $(Ey)T_1(x, x, y)$. That it is r.e. follows from Proposition 5.20(2), and that it is not recursive follows from Proposition 5.19(3). By Proposition 5.20(4), it also follows that $(y) \sim T_1(x, x, y)$ is not r.e.

EXERCISES

5.25. A set B is called *creative* if and only if B is r.e. and there is a partial recursive function φ such that, for any n, if $\zeta_n \subseteq \overline{B}$, then $\varphi(n) \in \overline{B} - \zeta_n$. Prove that $\{x|(Ey)T_1(x, x, y)\}$ is creative. Show that every creative set is non-recursive.

5.26. A partial recursive function φ is called *potentially recursive* if and only if there is a recursive function ψ such that $\varphi(x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n)$ whenever $\varphi(x_1, \ldots, x_n)$ is defined. Prove that $\mu y T_1(x, x, y)$ is not potentially recursive.

5.27.[D] A set B is called *simple* if and only if B is r.e., $\overline{B}$ is infinite, and $\overline{B}$ contains no infinite r.e. set. Every simple set is non-recursive. Show that a simple set exists.

5.28. A *recursive permutation* is a one-one recursive function from ω onto ω. Sets X and Y are called *isomorphic* (written $X \cong Y$) if there is a recursive permutation which maps X onto Y. Prove:

[A](a) The recursive permutations form a group under the operation of composition.

(b) $\cong$ is an equivalence relation.

(c) If X is recursive (r.e., creative, simple) and $X \cong Y$, then Y is recursive (r.e., creative, simple).

Myhill [1955] has shown that any two creative sets are isomorphic. (Also cf. Bernays [1957].)

5.29. X is *many-one reducible* to Y (written $X\ R_m\ Y$) if there is a recursive function f such that $u \in X$ if and only if $f(u) \in Y$. X and Y are called *many-one equivalent* (written $X \equiv_m Y$) if $X\ R_m\ Y$ and $Y\ R_m\ X$. X is *one-one reducible* to Y (written $X\ R_1\ Y$) if there is a one-one recursive function f such that $u \in X$ if and only if $f(u) \in Y$. X and Y are called *one-one equivalent* (written $X \equiv_1 Y$) if $X\ R_1\ Y$ and $Y\ R_1\ X$. Prove:

(a) $\equiv_m$ and $\equiv_1$ are equivalence relations.

(b) If X is creative, Y is r.e., and $X\ R_m\ Y$, then Y is creative. It can be shown (Myhill [1955]) that if X is creative and Y is r.e. then $Y\ R_m X$.

(c) (Myhill [1955]) If $X\ R_1\ Y$ then $X\ R_m Y$, and if $X \equiv_1 Y$ then $X \equiv_m Y$. However, many-one reducibility does not imply one-one reducibility, and many-one equivalence does not imply one-one equivalence. (Hint: let X be a simple set, Z an infinite recursive subset of X, and $Y = X - Z$. Then $X\ R_1\ Y$, $Y\ R_m\ X$, but not $(Y\ R_1\ X)$.) It can be proved that $X \equiv_1 Y$ if and only if $X \cong Y$.

5.30. (Dekker [1955]) X is said to be *productive* if there is a partial recursive function f such that if $\zeta_n \subseteq X$ then $f(n) \in X - \zeta_n$. Prove: (a) If X is productive, then X is not r.e.; hence, both X and $\bar{X}$ are infinite. ^{D}(b) If X is productive, then X has an infinite r.e. subset. Hence, if X is productive, $\bar{X}$ is not simple. (c) If X is r.e., then X is creative if and only if $\bar{X}$ is productive. ^{D}(d) There exist $2^{\aleph_0}$ productive sets.

5.31. (Dekker-Myhill [1960]) X is *recursively equivalent* to Y (written $X \sim Y$) if there is a one-one partial-recursive function which maps X onto Y. Prove: (a) $\sim$ is an equivalence relation. ^{D}(b) X is said to be *immune* if X is infinite and X has no infinite r.e. subset. X is said to be *isolated* if X is not recursively equivalent to a proper subset of X. (The isolated sets may be considered the recursive counterparts of the Dedekind-finite sets.) An infinite set is isolated if and only if it is immune. ^{D}(c) There exist $2^{\aleph_0}$ immune sets.

Recursively enumerable sets are also important because, if we assume Church's Thesis, the set T_K of Gödel numbers of theorems of any axiomatizable first-order theory K is r.e. (The same holds true of arbitrary formal axiomatic systems.) For, the relation Pf(y, x) (y is the Gödel number of a proof in K of a wf with Gödel number x, cf. p. 156) is recursive, if the set of Gödel numbers of the axioms is recursive, i.e., if the theory is axiomatic and Church's Thesis holds. Hence, $x \in T_K$ if and only if (Ey)Pf(y, x), and therefore, T_K is r.e. If we accept Church's Thesis, then K is effectively decidable if and only if the r.e. set T_K is recursive. We showed in Corollary 3.41 that every consistent extension K of the theory RR is recursively undecidable, i.e., T_K is not recursive.

Much more general results along these lines can be proved (cf. Smullyan [1961], Feferman [1957], Putnam [1957], Ehrenfeucht and Feferman [1960], Myhill [1955]). For example, (1) if every recursive set is expressible in K, then K is essentially recursively undecidable, i.e., for every consistent extension of K, T_K is not recursive (cf. Exercise 5.33 below); (2) for any consistent first-order theory with equality K in which every recursive function is representable and which satisfies (i′) and (ii) of p. 162, the set T_K is creative. (We assume that K has among its terms the numerals $\bar{0}$, $\bar{1}$, $\bar{2}$, ...) For further study of r.e. sets, cf. Post [1944] and Rogers [1967].

EXERCISES

5.32. Given a set A of natural numbers, define $A^\star$ as follows: $u \in A^\star$ if and only if u is a Gödel number of a wf $\mathfrak{A}(x_1)$ and the Gödel number of $\mathfrak{A}(\bar{u})$ is in A. Prove that, if A is recursive, then $A^\star$ is recursive.

5.33. Let K be a consistent theory having the same symbols as S.

(a) Let T_K be the set of Gödel numbers of theorems of K. Prove that $(\bar{T}_K)^\star$ is not weakly expressible in K (cf. Exercise 5.23, p. 262).

(b) If every recursive set is weakly expressible in K, show that K is recursively undecidable.

(c) If every recursive set is expressible in K, prove that K is essentially recursively undecidable.

4. Undecidable Problems

A general class of problems is said to be undecidable if and only if there is no general effective (or mechanical) procedure for solving each problem in the given class. For example, given any polynomial in any number of variables with integral coefficients, is there a set of integral values of the variables for which the polynomial has the value 0? We may be able to answer this question for certain special polynomials, but it turns out that there is no general procedure which will solve this problem for all polynomials (cf. Matiyasevich [1970]).

If we can arithmetize the formulation of a general class of problems and thus assign to each problem a natural number, then this class is undecidable if and only if there is no effectively computable function h such that, if n is the number of a given problem, then h(n) gives the solution of the problem. If we accept Church's Thesis (as we shall do in this section) the function h has to be partial recursive, and we then have a precise mathematical question. Examples of important mathematical decision problems which have been solved (negatively) are the word problem for semi-groups (Post [1947], Kleene [1952], § 71), and the very difficult word problem for groups (Boone [1959], Novikov [1955], Britton [1958], Higman [1961]). In addition, the decision problem for various first-order theories has been shown to have a negative solution, i.e., the general problem as to whether any given wf is provable in the theory is undecidable (cf. Corollary 3.36, Corollary 3.37, Proposition 3.41, Corollary 3.45, Proposition 3.46). We shall now present some more examples of undecidable problems.

The sequence of functions $\psi_n(x) = U(\mu y T_1(n, x, y))$ gives an enumeration of all partial recursive functions of one variable. Is there an effective procedure to determine for any n whether ψ_n is recursive (i.e., whether ψ_n is defined for all x)? A positive answer is equivalent to the recursiveness of the set A of all numbers n such that ψ_n is recursive. We shall show that A is not even r.e. Assume A r.e., and let h be a recursive function with range A. Define a new function $f(x) = [\psi_{h(x)}(x)] + 1 = [U(\mu y T_1(h(x), x, y))] + 1$. Hence, f is recursive and so there is some m such that $f = \psi_m$ and $m \in A$. Then $\psi_m(x) = \psi_{h(x)}(x) + 1$. Since $m \in A$, there is some k such that $m = h(k)$. Taking $x = k$, we have $\psi_m(k) = \psi_m(k) + 1$, which is a contradiction. Thus, there is no effective procedure by which we can tell whether any system of equations determines a recursive function.

We can obtain a "local" form of this result. Is there an effective procedure determining for any given m, n whether $\psi_n(m)$ is defined? The answer is negative. For, assume that $\theta(x, y)$ is a recursive function such that

$$\theta(x, y) = \begin{cases} 0 & \text{if } \psi_x(y) \text{ is defined} \\ 1 & \text{if } \psi_x(y) \text{ is not defined} \end{cases}$$

Now, let $\alpha(z) = \mu y(\theta(z, z) = 1 \wedge y = y)$. Clearly,

$$\alpha(z) = \begin{cases} 0 & \text{if } \psi_z(z) \text{ is undefined} \\ \text{undefined} & \text{if } \psi_z(z) \text{ is defined} \end{cases}$$

But α is partial recursive, and so, $\alpha = \psi_k$ for some k. Then

$$\psi_k(k) = \alpha(k) = \begin{cases} 0 & \text{if } \psi_k(k) \text{ is undefined} \\ \text{undefined} & \text{if } \psi_k(k) \text{ is defined} \end{cases}$$

which is a contradiction. (Other undecidable problems can be found in Rogers [1967].)

EXERCISES

5.34. Given a Turing machine T, can one effectively decide, given any instantaneous description α, whether or not there is a computation of T beginning with α? (*Halting problem* for T.) Show that there is a Turing machine with undecidable halting problem. For further discussion of this and similar problems, cf. Davis [1958], Chapter 5.

5.35. Prove: There is no normal algorithm $\mathfrak{B}$ over $M = \{1, *\}$ such that $\mathfrak{B}$ is applicable to exactly those words $\bar{n}$ such that n is an index of a normal algorithm $\mathfrak{A}$ over M such that $\mathfrak{A}$ is not applicable to $\bar{n}$.

For further examples of undecidable problems in the theory of algorithms, cf. Markov [1954], Chapter V. Because of the essential equivalence of normal algorithms, Turing machines, and Herbrand-Gödel systems of equations, any undecidability result established in terms of one of these approaches usually can be translated into corresponding results for the other two.

5.36. The function f such that

$$f(x) = \begin{cases} 0 & \text{if } \psi_x(x) \text{ is defined} \\ 1 & \text{otherwise} \end{cases}$$

is not recursive.

5.37.[D] Show that there is a recursive function $\eta(x)$ such that, for any x, $\eta(x)$ is the index of the partial recursive function v, where

$$v(y) = \begin{cases} 0 & \text{if } \psi_x(x) \text{ if defined} \\ \text{undefined} & \text{if } \psi_x(x) \text{ is undefined} \end{cases}$$

5.38.[D] (Rogers [1967]) Show that the following relations are not recursive (and, therefore, by Church's Thesis, are undecidable).

(a) y is in the range of ψ_x.

(b) $\psi_x(y) = z$.

(c) $\psi_x = \psi_y$. (Hint: use 5.37 and 5.36.)

The reader should not get the impression that all decision problems have a negative solution. In Chapter 1 it was shown that truth tables provide an effective procedure to determine whether any given statement form is a tautology. On p. 170, it was shown that the pure monadic predicate calculus is effectively decidable (cf. Ackermann [1954] and Suranyi [1959] for many positive results of a similar kind). Presburger [1929] showed that the first-order

theory obtained from first-order theory number theory S by omitting the multiplication symbol and the recursion axioms for multiplication is decidable (cf. p. 134, Exercise 3.7); Szmielew [1955] proved the decidability of the first-order theory of abelian groups; and Tarski [1951] established the decidability of the first-order theory of real-closed fields, which is the elementary part of the theory of real numbers.