Part A

The Structure of the Degrees

Chapter I Recursive Functions

This chapter is introductory in nature. We summarize material which is normally covered in a first course in Recursion Theory and which will be assumed within this book. Recursive and partial recursive functions are introduced and Church's Thesis is discussed. Relative recursion is then defined, and the Enumeration and Recursion Theorems are stated without proof. The reader familiar with this material should quickly skim through the chapter in order to become familiar with our notation. We refer the reader to the first five chapters of Cutland [1980] for a careful rigorous treatment of the material introduced in this chapter.

1. The Recursive and Partial Recursive Functions

The search for algorithms has pervaded Mathematics throughout its history. It was not until this century, however, that rigorous mathematical definitions of *algorithm* were discovered, giving rise to the class of partial recursive functions.

This book deals with a classification of total functions of the form $f: N \to N$ in terms of the information required to compute such a function. The rules for carrying out such computations are *algorithms* (partial functions $\varphi: N^k \to N$ for some $k \in N$) with access to information possessed by *oracles*. The easiest functions to compute are those for which no oracular information is required, the recursive functions. Thus we begin by defining the (total) recursive functions, and then indicate how to modify this definition to obtain the class of partial recursive functions. The section concludes with discussions of Church's Thesis and of general *spaces* on which recursive functions can be defined.

1.1 Definition. Let $R \subseteq N^{k+1}$. $\mu y[(x_1, \dots, x_k, y) \in R]$ is the least y such that $(x_1, \dots, x_k, y) \in R$ if such a y exists, and is undefined otherwise. Henceforth, we will refer to μ as the *least number operator*.

1.2 Definition. The class $\mathscr{R}$ of *recursive functions* is the smallest class of functions with domain N^k for some $k \in N$ and range N which contains:

(i) The *zero function*: $Z(x) = 0$ for all $x \in N$;
(ii) The *successor function*: $S(x) = x + 1$ for all $x \in N$;

(iii) The *projection functions*: $P^n_j(x_0, \ldots, x_n) = x_j$ for all $n, x_0, \ldots, x_n \in N$ and $j \leqslant n$;

and is closed under:

(iv) *Substitution*: For all $m, k \in N$, if all of $g(x_0, \ldots, x_m)$, $h_0(y_0, \ldots, y_k), \ldots, h_m(y_0, \ldots, y_k)$ are elements of $\mathscr{R}$, then

$$f(y_0, \ldots, y_k) = g(h_0(y_0, \ldots, y_k), \ldots, h_m(y_0, \ldots, y_k))$$

is an element of $\mathscr{R}$;

(v) *Recursion*: For all $n \in N$, if $g(x_0, \ldots, x_n)$ and $h(x_0, \ldots, x_{n+2})$ are elements of $\mathscr{R}$, then $f(x_0, \ldots, x_{n+1})$ is an element of $\mathscr{R}$, where

$$f(x_0, \ldots, x_n, 0) = g(x_0, \ldots, x_n)$$

and

$$f(x_0, \ldots, x_n, y + 1) = h(x_0, \ldots, x_n, y, f(x_0, \ldots, x_n, y));$$

(vi) The *least number operator*: For all $n \in N$, if $g(x_0, \ldots, x_n, y)$ is an element of $\mathscr{R}$ and $\forall x_0, \ldots, x_n \exists y[g(x_0, \ldots, x_n, y) = 1]$ then

$$f(x_0, \ldots, x_n) = \mu y[g(x_0, \ldots, x_n, y) = 1]$$

is an element of $\mathscr{R}$.

An element of $\mathscr{R}$ is called a *recursive function*.

1.3 Definition. Fix $n \in N$ and let $\mathscr{C}$ be a countable class of partial functions of n natural number variables. An *enumeration* of $\mathscr{C}$ is a partial function $\varphi: N^{n+1} \to N$ which lists the elements of $\mathscr{C}$, i.e.,

(i) $\forall \psi \in \mathscr{C}\, \exists k \in N(\lambda x_1, \ldots, x_n \varphi(k, x_1, \ldots, x_n) = \psi(x_0, \ldots, x_n))$

and

(ii) $\forall k \in N(\lambda x_1, \ldots, x_n \varphi(k, x_1, \ldots, x_n) \in \mathscr{C})$.

1.4 Example. Let $\mathscr{C} = \{f_i : i \in N\}$ where $f_i(x) = i$ for all $x \in N$. Then $g: N^2 \to N$ defined by $g(n, x) = n$ is an enumeration of $\mathscr{C}$.

The Enumeration Theorem for partial recursive functions of one variable is an important tool used in almost every proof in this book. What we would like to have is a recursive enumeration of the class of recursive functions of one variable. Unfortunately, such an enumeration does not exist (see Exercise 1.10). All that is needed, however, is a *partial recursive* enumeration of the class of *partial recursive* functions. With this in mind, we now introduce the class of partial recursive functions.

1.5 Remark. The obstacle to obtaining a recursive enumeration of the class of recursive functions of one variable lies in 1.2(vi), the application of the least number operator to obtain new recursive functions. There is no algorithm which will identify whether or not $\forall x_0, \ldots, x_n \exists y[g(x_0, \ldots, x_n, y) = 1]$. This difficulty can be circumvented by producing an algorithm which assigns natural numbers (called *Gödel numbers*) to computations carried out in 1.2(i)–(vi). One then searches for the least numbered computation which yields $g(x_0, \ldots, x_n, y) = 1$ for some y, say $y = y_0$, and defines

$$f(x_0, \ldots, x_n) = \begin{cases} y_0 & \text{if } y_0 \text{ is ever found,} \\ \uparrow & \text{otherwise.} \end{cases}$$

Such a procedure was carried out by Kleene, giving rise to the class of *partial recursive functions*, $\mathscr{P}$. This class contains all the recursive functions, together with some additional functions, none of which are total.

During the 1930's and 1940's, several attempts were made to give a rigorous mathematical definition of *algorithm*. One of these definitions was the class of partial recursive functions described in Remark 1.5. All of the definitions were eventually shown to be equivalent, and the equivalence of some of the early definitions prompted Church to propose his thesis, which asserts:

1.6 Church's Thesis. A function is partial recursive if and only if there is an algorithm which computes the function on its domain, and diverges outside the domain of the function.

Church's Thesis asserts that the intuitive notion of *algorithm* is equivalent to the mathematically precise notion of *partial recursive function*. The thesis is almost universally accepted, and its use has become general mathematical practice. We will be using Church's Thesis freely and without any explicit warning throughout this book, by describing the computation of a function and automatically assuming that the resulting function is partial recursive. A rigorous proof could be given in every case, but would be very tedious.

In this age of digital computers, the reader might feel most comfortable with the following description of partial recursive functions. A function f is partial recursive if there is a program for a digital computer (no restrictions on memory size are placed on such a computer, so that we assume that the computer has available to it an infinite supply of memory space, only finitely much of which is used at a given time) such that whenever x is fed as input to the computer, the computer will spew out $f(x)$ after spending a finite amount of time performing computations as directed by the program (no restrictions, however, are placed on the amount of time available) if $x \in \text{dom}(f)$, and the computer will give no answer (perhaps computing forever) if $x \notin \text{dom}(f)$.

To this point, we have only considered functions from N^k into N for some $k > 0$. Shoenfield [1971] has noted that N^k and N can be replaced by any *spaces*, i.e., domains which can *effectively* be placed in one-one correspondence with N. Henceforth, any space will be acceptable as either the domain or range of a recursive function. Typical spaces which we will be using later are mentioned in the next example.

1.7 Example. The following are spaces:

(i) N^k, the set of all k-tuples of natural numbers, for all $k \geqslant 1$.

(ii) $N^{<\omega}$, the set of all finite sequences of natural numbers. Henceforth we will denote $N^{<\omega}$ by $\mathscr{S}$, and call an element of $\mathscr{S}$ a *string*.

(iii) $\mathscr{S}_f = \{\sigma \in \mathscr{S} : \sigma(n) < f(n) \text{ for all } n \in N \text{ such that } \sigma(n)\downarrow\}$, where $f: N \to N$ is a recursive function and $f(x) \neq 0$ for all $x \in N$. $\mathscr{S}_f$ is called the space of *f-valued strings*. If f is the constant function $f(x) = c$ for all x, then $\mathscr{S}_c$ will be used in place of $\mathscr{S}_f$. Thus $\mathscr{S}_2$ is the space of all finite sequences of 0s and 1s.

The following notation will be used:

1.8 Definition. Let $\sigma, \tau \in \mathscr{S}$ be given. We say that $\sigma \subseteq \tau$ if for all $i \in N$, if $\sigma(i)\downarrow$ then $\tau(i)\downarrow$ and $\sigma(i) = \tau(i)$. Given $f: N \to N$, we say that $\sigma \subseteq f$ if for all $i \in N$, if $\sigma(i)\downarrow$ then $\sigma(i) = f(i)$. $\subset$ will denote $\subseteq$ and $\neq$. $\mathrm{lh}(\sigma) = |\{i : \sigma(i)\downarrow\}|$ is the *length* of σ. $\sigma * \tau$ is the string of length $\mathrm{lh}(\sigma) + \mathrm{lh}(\tau)$ defined by

$$\sigma * \tau(x) = \begin{cases} \sigma(x) & \text{if} \quad x < \mathrm{lh}(\sigma), \\ \tau(x - \mathrm{lh}(\sigma)) & \text{if} \quad \mathrm{lh}(\sigma) \leqslant x < \mathrm{lh}(\sigma) + \mathrm{lh}(\tau). \end{cases}$$

In the next section, we will discuss relative recursiveness. We will then be able to classify arbitrary functions $f: N \to N$ on the basis of how much additional information is required (from an oracle) in order to compute f.

1.9–1.13 Exercises

1.9 The class of *primitive recursive functions* is the smallest class of functions containing the functions mentioned in 1.2(i)–(iii) and closed under the operations of 1.2(iv)–(v).

(i) Show that there is a recursive enumeration of the class of primitive recursive functions of one variable. (*Hint*: Recursively assign Gödel numbers to computations, and define the enumeration $F(e, x)$, where $\lambda x F(e, x)$ is the function with Gödel number e.)

(ii) Show that there is a recursive function which is not primitive recursive. (*Hint*: Diagonalize against an enumeration of the primitive recursive functions.)

1.10 Show that there is no recursive enumeration of the class of recursive functions. (*Hint*: If there were such an enumeration, a diagonalization as in 1.9(ii) would produce a contradiction.)

1.11 Explain why your proof of 1.10 will not generalize to show that there is no partial recursive enumeration of the class of partial recursive functions.

1.12 Let S be a space.

(i) Show that S^n is a space.

(ii) Show that $S^{<\omega}$ is a space.

(iii) Show that there is a $T \subseteq S$ such that T is not a space.

(*Hint*: Use a cardinality argument after showing that there are only countably many algorithms.)

1.13 Let S and T be spaces. Show that:

(i) $S \times T$ is a space.

(ii) For all $n \in N$, $[0, n] \times S$ and $S \times [0, n]$ are spaces.

2. *Relative Recursion*

Recursion Theory classifies total functions on the basis of how much additional information must be provided by an oracle to compute the given function. This classification relies on the notion of relative recursion.

Relative recursion is defined by expanding the class of initial functions in Definition 1.1. We will use this notion in Chap. 2 to form an algebraic structure from $\{f: N \to N\}$.

2.1 Definition. Let $f: N^m \to N$ be given. The class $\mathscr{R}_f$ of *functions recursive in* f is the smallest class of functions containing f and the functions mentioned in 1.2(i)–(iii) and closed under the operations of 1.2(iv)–(vi). An element g of $\mathscr{R}_f$ is said to be *recursive in* f, written $g \leqslant_T f$.

Recursiveness in f, or relative recursion, was introduced by Turing [1939] whose name gave rise to the T in $\leqslant_T$. $\leqslant_T$ is frequently referred to as *Turing reducibility*.

The partial recursive functions, as we indicated earlier, are those partial functions which an idealized digital computer can compute. The domain of such a function is the set of numbers which, when fed as input to the computer, will eventually cause the computer to output a number.

Given $f: N^k \to N$, the *partial functions computable from* f can similarly be described through the use of a digital computer with access to the *oracle* f. The notion of computer program is generalized to allow instructions of the form *if* $f(x_1, \dots, x_k) = y$ *proceed to a certain instruction*; *otherwise*, *proceed to another specified instruction*. As we proceed through the computation of a partial function g computable from f to which z has been fed as input, whenever we reach a step in the program of the type just described above, the computer does something non-constructive; it asks the f oracle whether $f(x_1, \dots, x_k) = y$, and once the answer is received from the f oracle, continues the computation utilizing the information provided by the oracle. Complete knowledge of f allows us to compute $g(z)$ whenever $g(z)\downarrow$, so g is computable from f. Futhermore, if $g(z)\downarrow$, then since any computation is completed in finitely many steps, only a finite amount of information about f is used.

2.2 Remark. There is also a version of Church's Thesis for relative recursion which will be used freely throughout this book. It asserts that the partial functions computable from f are exactly those for which there exists an algorithm using an f oracle as above to perform the computation.

It will frequently be more convenient to use sets than functions when discussing relative recursiveness. This is easily accomplished by identifying a set with its characteristic function, which we define as follows:

2.3 Definition. Let $A \subseteq N$. The *characteristic function* of A, χ_A, is defined by

$$\chi_A(n) = \begin{cases} 0 & \text{if } \ x \notin A \\ 1 & \text{if } \ x \in A. \end{cases}$$

More generally, a relation $R \subseteq N^k$ will be identified with its *characteristic function*

χ_R, defined by

$$\chi_R(x_1, \ldots, x_k) = \begin{cases} 0 & \text{if} \quad R(x_1, \ldots, x_k) \text{ is false,} \\ 1 & \text{if} \quad R(x_1, \ldots, x_k) \text{ is true.} \end{cases}$$

2.4 Definition. A relation R is said to be *recursive in* the function f if χ_R is recursive in f.

2.5 Remarks. A summary of the history of relative recursion appears in Kleene and Post [1954]. Three versions appear in the literature, all of which were later proved to be equivalent. They were formulated by Turing [1939] generalizing the machines introduced by Turing [1937]; Kleene [1943] extending the definition of recursive functions; and Post [1948] extending the concept of canonical sets which was introduced in Post [1943].

2.6 Exercise. Let $f, g: N \to N$ be given such that $\{i: f(i) \neq g(i)\}$ is finite. Show that $f \leqslant_T g$.

3. *The Enumeration and Recursion Theorems*

Two basic theorems of Recursion Theory are stated in this section. The first of these, the Enumeration Theorem, will be used in virtually every proof in this book. The Recursion Theorem will be used to enable us to simplify proofs of certain theorems. Complete proofs or detailed sketches of the proofs of these theorems can be found in Soare [1984], Rogers [1967], Cutland [1980] and Kleene [1952]. A nice proof of the Recursion Theorem can also be found in Owings [1973].

The Enumeration Theorem asserts the existence of a function $\varphi(\sigma, e, x, s)$ which uniformly induces a whole class of enumerations. The definition of $\varphi(\sigma, e, x, s)$ reflects the following intuition. A computer is programmed, with e coding the program. The input x is then fed to the computer, and the computer then performs s steps as directed by the program. During these s steps, the computer may come across program instructions of the form *if $f(x) = y$ proceed to a certain instruction, and if $f(x) \neq y$ proceed to another specified, but different instruction.* When faced with such a choice, the computer asks "is $\sigma(x) = y$?". If $x \geqslant \mathrm{lh}(\sigma)$, then there will be no output. If $x < \mathrm{lh}(\sigma)$, then σ answers the question for f and the computation continues. If there is no output from the computer after s steps, the computation ceases and $\varphi(\sigma, e, x, s)\uparrow$. If a number is outputted by the end of the sth step, $\varphi(\sigma, e, x, s)\downarrow$ and is set equal to this output.

3.1 Enumeration Theorem. *There is a partial recursive function $\varphi: \mathscr{S} \times N^3 \to N$ with the following properties*:

(i) (*Use property*)

$$\forall \sigma, \tau \in \mathscr{S}\ \forall e, x, s, y \in N(\sigma \subseteq \tau \,\&\, \varphi(\sigma, e, x, s)\downarrow = y \to \varphi(\tau, e, x, s)\downarrow = y),$$

i.e., if a number is given as output, then oracle information extending the original information will not alter the output.

(ii) (*Permanence property*)

$$\forall \sigma \in \mathscr{S}\, \forall e, x, s, t, y \in N(s \leqslant t \;\&\; \varphi(\sigma, e, x, s)\downarrow = y \rightarrow \varphi(\sigma, e, x, t)\downarrow = y),$$

i.e., once a number is given as output, additional steps do not change this output.

(iii) (*Uniform enumeration property*) *Given* $f: N \rightarrow N$ and $\theta: N \rightarrow N$ *computable from* f, *then there is an* $e \in N$ *such that for all* x, $y \in N$

$$\theta(x) = y \leftrightarrow \exists \sigma\, \exists s(\sigma \subseteq f \;\&\; \varphi(\sigma, e, x, s)\downarrow = y),$$

i.e., for every partial function θ *computable from* f, *there is a program coded into* φ *which computes* θ. (*Note that if* θ *is not given but rather defined by the above formula, then* θ *is computable from* f.)

(iv) (*Recursiveness property*) *The domain of* φ *is a recursive subset of* $\mathscr{S} \times N^3$ (*since* s *bounds the length of a permissible computation*).

(v) (*Uniform coding property*) *Given a sequence* $\{\theta_i : i \in N\}$ *of functions computable from* $f: N \rightarrow N$ *such that the definition of* θ_i *is given by a finite set of instructions using parameter* i, *then there is a recursive function* g *such that for all* $x, y \in N$

$$\theta_i(x) = y \leftrightarrow \exists \sigma\, \exists s(\varphi(\sigma, g(i), x, s)\downarrow = y \;\&\; \sigma \subseteq f),$$

i.e., if the definitions of a class of functions are given uniformly recursively, then there is a recursive function which gives codes for programs computing each of these functions.

3.2 Remark. Let φ be the function given by the Enumeration Theorem. For all $\theta: N \rightarrow N$, define the functional Φ^θ by

(i) $$\Phi^\theta(e, x) = y \Leftrightarrow \exists \sigma \in \mathscr{S}(\sigma \subseteq \theta \;\&\; \varphi(\sigma, e, x, \mathrm{lh}(\sigma))\downarrow = y).$$

Then Φ^θ is computable from θ, and Φ^f provides an enumeration of $\mathscr{R}_f$, uniformly in f. In particular, if f is any recursive function, then Φ^f is a partial recursive enumeration of the class of partial recursive functions. If $f = \emptyset$, then we write Φ in place of Φ^f.

For the remainder of this book, φ will denote the function given by the enumeration theorem, and Φ^θ will be defined as in 3.2(i). For each $e \in N$, we will also fix the function $\Phi_e^\theta = \lambda x \Phi^\theta(e, x)$.

Property 3.1(v) is known as the *s-m-n* Theorem. Another way of stating this theorem is as follows: Let $h(x, y)$ be a function computable from f. Then there is a function g computable from f such that for all $x \in N$, $h(x, y) = \Phi^f_{g(x)}$. Thus, for example, if $h(x, y) = x$, then there is a recursive function g such that for each $x \in N$, $g(x)$ is an index for the constant function x as a recursive function.

The Recursion Theorem is a basic theorem about the enumerations mentioned in Remark 3.2. It is frequently referred to as the Fixed Point Theorem.

3.3 Recursion Theorem. *Let the recursive function* $h: N \rightarrow N$ *be given. Then there is an* $e \in N$ *such that* $\Phi_e^f = \Phi^f_{h(e)}$ *for all* $f: N \rightarrow N$.

3.4 Definition. If $\Phi_e^f = \Phi_{h(e)}^f$ then Φ_e^f is called a *fixed point* (of the enumeration Φ^f) for h.

We will be using the Recursion Theorem in the following way. Given a function $f: N \to N$ and $e \in N$, we will start with the partial function Φ_e^f and, uniformly in e, we will recursively construct a partial function $\Phi_{h(e)}^f$. An application of the Recursion Theorem will allow us to choose an e such that our starting function Φ_e^f and our constructed function $\theta_{h(e)}^f$ are identical. Thus in certain situations, the Recursion Theorem allows us to construct a function while simultaneously using information about the function in its construction. By the uses of the Recursion Theorem, the information used about the function will have to be specified at an earlier stage, although this fact is hidden in the actual applications.

3.5 Remark. The Enumeration and Recursion Theorems were discovered by Kleene (see Kleene [1952]).

3.6–3.8 Exercises. The definitions in 3.6 and 3.7 describe recursive procedures which define one partial recursive function in terms of another. For each definition, apply the Recursion Theorem to obtain a fixed point. Is this fixed point a total function? What is the fixed point?

3.6 $$\Phi_{h(e)}(n) = \begin{cases} 0 & \text{if} \quad n = 0, \\ \Phi_e(n-1) & \text{if} \quad n > 0 \;\&\; \Phi_e(n-1)\downarrow, \\ \uparrow & \text{otherwise.} \end{cases}$$

3.7 $$\Phi_{h(e)}(n) = \begin{cases} n & \text{if} \quad \Phi_e(n)\downarrow \neq n, \\ \uparrow & \text{otherwise.} \end{cases}$$

3.8 Prove the Recursion Theorem. (*Hint*: Given $m \in N$, define

$$\psi(m, x) = \begin{cases} \Phi_{\Phi_m(m)}(x) & \text{if} \quad \Phi_m(m)\downarrow, \\ \uparrow & \text{otherwise.} \end{cases}$$

By the uniform coding property, find a recursive function g such that $\psi(m, x) = \Phi_{g(m)}(x)$ for all m and x. Given a recursive function f, let e be a Gödel number for fg. Show that $n = g(e)$ is a fixed point for f.)