

Chapter 9

Elementary Recursive Function Theory

9.1 Acceptable Indexings

In a previous Section, we have exhibited a specific indexing of the partial computable functions by encoding the RAM programs.

Using this indexing, we showed the existence of a universal function φ_{univ} and of a computable function c , with the property that for all $x, y \in \mathbb{N}$,

$$\varphi_{c(x,y)} = \varphi_x \circ \varphi_y.$$

It is natural to wonder whether the same results hold if a different coding scheme is used or if a different model of computation is used, for example, Turing machines.

What we are aiming at is to find some simple properties of “nice” coding schemes that allow one to proceed without using explicit coding schemes, as long as the above properties hold.

Remarkably, such properties exist.

Furthermore, any two coding schemes having these properties are equivalent in a strong sense (effectively equivalent), and so, one can pick any such coding scheme without any risk of losing anything else because the wrong coding scheme was chosen.

Such coding schemes, also called indexings, or Gödel numberings, or even programming systems, are called *acceptable indexings*.

Definition 9.1. An *indexing* of the partial computable functions is an infinite sequence $\varphi_0, \varphi_1, \dots$, of partial computable functions that includes all the partial computable functions of one argument (there might be repetitions, this is why we are not using the term enumeration). An indexing is *universal* if it contains the partial computable function φ_{univ} such that

$$\varphi_{univ}(i, x) = \varphi_i(x)$$

for all $i, x \in \mathbb{N}$. An indexing is *acceptable* if it is universal and if there is a total computable function c for composition, such that

$$\varphi_{c(i,j)} = \varphi_i \circ \varphi_j$$

for all $i, j \in \mathbb{N}$.

A very useful property of acceptable indexings is the so-called “*s-m-n Theorem*”.

Using the slightly loose notation $\varphi(x_1, \dots, x_n)$ for $\varphi(\langle x_1, \dots, x_n \rangle)$, the s-m-n theorem says the following.

Given a function φ considered as having $m + n$ arguments, if we fix the values of the first m arguments and we let the other n arguments vary, we obtain a function ψ of n arguments. Then, the index of ψ depends in a computable fashion upon the index of φ and the first m arguments x_1, \dots, x_m .

We can “pull” the first m arguments of φ into the index of ψ .

Theorem 9.1. (*The “s-m-n Theorem”*) *For any acceptable indexing $\varphi_0, \varphi_1, \dots$, there is a total computable function s , such that, for all $i, m, n \geq 1$, for all x_1, \dots, x_m and all y_1, \dots, y_n , we have*

$$\varphi_{s(i,m,x_1,\dots,x_m)}(y_1, \dots, y_n) = \varphi_i(x_1, \dots, x_m, y_1, \dots, y_n).$$

As a first application of the s-m-n Theorem, we show that any two acceptable indexings are effectively inter-translatable.

Theorem 9.2. *Let $\varphi_0, \varphi_1, \dots$, be a universal indexing, and let ψ_0, ψ_1, \dots , be any indexing with a total computable s-1-1 function, that is, a function s such that*

$$\psi_{s(i,1,x)}(y) = \psi_i(x, y)$$

for all $i, x, y \in \mathbb{N}$. Then, there is a total computable function t such that $\varphi_i = \psi_{t(i)}$.

Using Theorem 9.2, if we have two acceptable indexings $\varphi_0, \varphi_1, \dots$, and ψ_0, ψ_1, \dots , there exist total computable functions t and u such that

$$\varphi_i = \psi_{t(i)} \quad \text{and} \quad \psi_i = \varphi_{u(i)}$$

for all $i \in \mathbb{N}$.

Also note that if the composition function c is primitive recursive, then any s-m-n function is primitive recursive, and the translation functions are primitive recursive.

Actually, a stronger result can be shown. It can be shown that for any two acceptable indexings, there exist total computable *injective* and *surjective* translation functions.

In other words, any two acceptable indexings are recursively isomorphic (Roger's isomorphism theorem). Next, we turn to algorithmically unsolvable, or *undecidable*, problems.

9.2 Undecidable Problems

We saw in Section 8.2 that the halting problem for RAM programs is undecidable. In this section, we take a slightly more general approach to study the undecidability of problems, and give some tools for resolving decidability questions.

First, we prove again the undecidability of the halting problem, but this time, for *any* indexing of the partial computable functions.

Theorem 9.3. (*Halting Problem, Abstract Version*)

Let ψ_0, ψ_1, \dots , be any indexing of the partial computable functions. Then, the function f defined such that

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

is not computable.

Proof. Assume that f is computable, and let g be the function defined such that

$$g(x) = f(x, x)$$

for all $x \in \mathbb{N}$. Then g is also computable.

Let θ be the function defined such that

$$\theta(x) = \begin{cases} 0 & \text{if } g(x) = 0, \\ \text{undefined} & \text{if } g(x) = 1. \end{cases}$$

We claim that θ is not even partial computable. Observe that θ is such that

$$\theta(x) = \begin{cases} 0 & \text{if } \psi_x(x) \text{ is undefined,} \\ \text{undefined} & \text{if } \psi_x(x) \text{ is defined.} \end{cases}$$

If θ was partial computable, it would occur in the list as some ψ_i , and we would have

$$\theta(i) = \psi_i(i) = 0 \quad \text{iff} \quad \psi_i(i) \text{ is undefined,}$$

a contradiction. Therefore, f and g can't be computable. \square

The function g defined in the proof of Theorem 9.3 is the characteristic function of a set denoted as K , where

$$K = \{x \mid \psi_x(x) \text{ is defined}\}.$$

The set K is an example of a set which is not computable. Since this fact is quite important, we give the following definition.

Definition 9.2. A subset of Σ^* (or a subset of \mathbb{N}) is *computable* (or *recursive*, or *decidable*) iff its characteristic function is a total computable function (total recursive function).

Using Definition 9.2, Theorem 9.3 can be restated as follows.

Lemma 9.4. *For any indexing $\varphi_0, \varphi_1, \dots$ of the partial computable functions (over Σ^* or \mathbb{N}), the set $K = \{x \mid \varphi_x(x) \text{ is defined}\}$ is not computable.*

Computable sets allow us to define the concept of a decidable (or undecidable) problem.

The idea is to generalize the situation described in Section 8.2 and Section 8.4, where a set of objects, the RAM programs, is encoded into a set of natural numbers, using a coding scheme.

Definition 9.3. Let C be a countable set of objects, and let P be a property of objects in C . We view P as the set

$$\{a \in C \mid P(a)\}.$$

A *coding-scheme* is an injective function $\#: C \rightarrow \mathbb{N}$ that assigns a unique code to each object in C .

The property P is *decidable (relative to $\#$)* iff the set

$$\{\#(a) \mid a \in C \text{ and } P(a)\}$$

is computable.

The property P is *undecidable (relative to $\#$)* iff the set

$$\{\#(a) \mid a \in C \text{ and } P(a)\}$$

is not computable.

Observe that the decidability of a property P of objects in C depends upon the coding scheme $\#$.

Thus, if we are cheating in using a non-effective coding scheme, we may declare that a property is decidable even though it is not decidable in some reasonable coding scheme.

Consequently, we require a coding scheme $\#$ to be *effective* in the following sense.

Given any object $a \in C$, we can effectively (i.e., algorithmically) determine its code $\#(a)$.

Conversely, given any integer $n \in \mathbb{N}$, we should be able to tell effectively if n is the code of some object in C , and if so, to find this object.

In practice, it is always possible to describe the objects in C as strings over some (possibly complex) alphabet Σ (sets of trees, graphs, etc).

For example, the equality of the partial functions φ_x and φ_y can be coded as the set

$$\{\langle x, y \rangle \mid x, y \in \mathbb{N}, \varphi_x = \varphi_y\}.$$

We now show that most properties about programs (except the trivial ones) are undecidable.

First, we show that it is undecidable whether a RAM program halts for every input. In other words, it is undecidable whether a procedure is an algorithm. We actually prove a more general fact.

Lemma 9.5. *For any acceptable indexing $\varphi_0, \varphi_1, \dots$ of the partial computable functions, the set*

$$\text{TOTAL} = \{x \mid \varphi_x \text{ is a total function}\}$$

is not computable.

Proof. The proof uses a technique known as reducibility.

We try to reduce a set A known to be *noncomputable* to TOTAL via a computable function $f: A \rightarrow \text{TOTAL}$, so that

$$x \in A \quad \text{iff} \quad f(x) \in \text{TOTAL}.$$

If TOTAL were computable, its characteristic function g would be computable, and thus, the function $g \circ f$ would be computable, a contradiction, since A is assumed to be noncomputable.

In the present case, we pick $A = K$.

To find the computable function $f: K \rightarrow \text{TOTAL}$, we use the s-m-n Theorem.

Let θ be the function defined below: for all $x, y \in \mathbb{N}$,

$$\theta(x, y) = \begin{cases} \varphi_x(x) & \text{if } x \in K, \\ \text{undefined} & \text{if } x \notin K. \end{cases}$$

Note that θ does not depend on y .

The function θ is partial computable. Indeed, we have

$$\theta(x, y) = \varphi_x(x) = \varphi_{univ}(x, x).$$

Thus, θ has some index j , so that $\theta = \varphi_j$, and by the s-m-n Theorem, we have

$$\varphi_{s(j,1,x)}(y) = \varphi_j(x, y) = \theta(x, y).$$

Let f be the computable function defined such that

$$f(x) = s(j, 1, x)$$

for all $x \in \mathbb{N}$. Then, we have

$$\varphi_{f(x)}(y) = \begin{cases} \varphi_x(x) & \text{if } x \in K, \\ \text{undefined} & \text{if } x \notin K \end{cases}$$

for all $y \in \mathbb{N}$.

Thus, observe that $\varphi_{f(x)}$ is a total function iff $x \in K$, that is,

$$x \in K \quad \text{iff} \quad f(x) \in \text{TOTAL},$$

where f is computable. As we explained earlier, this shows that TOTAL is not computable. \square

The above argument can be generalized to yield a result known as Rice's Theorem.

Let $\varphi_0, \varphi_1, \dots$ be any indexing of the partial computable functions, and let C be any set of partial computable functions.

We define the set P_C as

$$P_C = \{x \in \mathbb{N} \mid \varphi_x \in C\}.$$

We can view C as a property of some of the partial computable functions. For example

$$C = \{\text{all total computable functions}\}.$$

We say that C is *nontrivial* if C is neither empty nor the set of all partial computable functions.

Equivalently C is nontrivial iff $P_C \neq \emptyset$ and $P_C \neq \mathbb{N}$.

Theorem 9.6. (*Rice's Theorem*) *For any acceptable indexing $\varphi_0, \varphi_1, \dots$ of the partial computable functions, for any set C of partial computable functions, the set*

$$P_C = \{x \in \mathbb{N} \mid \varphi_x \in C\}$$

is noncomputable unless C is trivial.

Proof. Assume that C is nontrivial. A set is computable iff its complement is computable (the proof is trivial).

Hence, we may assume that the totally undefined function is not in C , and since $C \neq \emptyset$, let ψ be some other function in C .

We produce a computable function f such that

$$\varphi_{f(x)}(y) = \begin{cases} \psi(y) & \text{if } x \in K, \\ \text{undefined} & \text{if } x \notin K, \end{cases}$$

for all $y \in \mathbb{N}$.

We get f by using the s-m-n Theorem. Let $\psi = \varphi_i$, and define θ as follows:

$$\theta(x, y) = \varphi_{univ}(i, y) + (\varphi_{univ}(x, x) - \varphi_{univ}(x, x)),$$

where $-$ is the primitive recursive function for truncated subtraction.

Clearly, θ is partial computable, and let $\theta = \varphi_j$.

By the s-m-n Theorem, we have

$$\varphi_{s(j,1,x)}(y) = \varphi_j(x, y) = \theta(x, y)$$

for all $x, y \in \mathbb{N}$. Letting f be the computable function such that

$$f(x) = s(j, 1, x),$$

by definition of θ , we get

$$\varphi_{f(x)}(y) = \theta(x, y) = \begin{cases} \psi(y) & \text{if } x \in K, \\ \text{undefined} & \text{if } x \notin K. \end{cases}$$

Thus, f is the desired reduction function.

Now, we have

$$x \in K \quad \text{iff} \quad f(x) \in P_C,$$

and thus, the characteristic function C_K of K is equal to $C_P \circ f$, where C_P is the characteristic function of P_C .

Therefore, P_C is not computable, since otherwise, K would be computable, a contradiction. \square

Rice's Theorem shows that all nontrivial properties of the input/output behavior of programs are undecidable! In particular, the following properties are undecidable.

Lemma 9.7. *The following properties of partial computable functions are undecidable.*

- (a) *A partial computable function is a constant function.*
- (b) *Given any integer $y \in \mathbb{N}$, is y in the range of some partial computable function.*
- (c) *Two partial computable functions φ_x and φ_y are identical.*
- (d) *A partial computable function φ_x is equal to a given partial computable function φ_a .*
- (e) *A partial computable function yields output z on input y , for any given $y, z \in \mathbb{N}$.*
- (f) *A partial computable function diverges for some input.*
- (g) *A partial computable function diverges for all input.*

A property may be undecidable although it is partially decidable. By partially decidable, we mean that there exists a computable function g that enumerates the set $P_C = \{x \mid \varphi_x \in C\}$.

This means that there is a computable function g whose range is P_C .

We say that P_C is *computably enumerable*. Indeed, g provides a computable enumeration of P_C , with possible repetitions.

9.3 Listable (Recursively Enumerable) Sets

Consider the set

$$A = \{x \in \mathbb{N} \mid \varphi_x(a) \text{ is defined}\},$$

where $a \in \mathbb{N}$ is any fixed natural number.

By Rice's Theorem, A is not computable (check this).

We claim that A is the range of a computable function g . For this, we use the T -predicate.

We produce a function which is actually primitive recursive.

First, note that A is nonempty (why?), and let x_0 be any index in A .

We define g by primitive recursion as follows:

$$g(0) = x_0,$$
$$g(x + 1) = \begin{cases} \Pi_1(x) & \text{if } T(\Pi_1(x), a, \Pi_2(x)), \\ x_0 & \text{otherwise.} \end{cases}$$

Since this type of argument is new, it is helpful to explain informally what g does.

For every input x , the function g tries finitely many steps of a computation on input a of some partial computable function.

The computation is given by $\Pi_2(x)$, and the partial function is given by $\Pi_1(x)$.

Since Π_1 and Π_2 are projection functions, when x ranges over \mathbb{N} , both $\Pi_1(x)$ and $\Pi_2(x)$ also range over \mathbb{N} .

Such a process is called a *dovetailing* computation.

Therefore all computations on input a for all partial computable functions will be tried, and the indices of the partial computable functions converging on input a will be selected.

Definition 9.4. A subset X of \mathbb{N} is *listable*, or *computably enumerable* (or *recursively enumerable*) iff either $X = \emptyset$, or X is the range of some total computable function. Similarly, a subset X of Σ^* is *listable*, or *computably enumerable* (or *recursively enumerable*) iff either $X = \emptyset$, or X is the range of some total computable function.

For short, a *computably enumerable set* is also called an *c.e. set*. A computably enumerable set is sometimes called a *partially decidable* set.

The following Lemma relates computable sets and computably enumerable sets.

Lemma 9.8. *A set A is computable iff both A and its complement \overline{A} are listable.*

Proof. Assume that A is computable. Then, it is trivial that its complement is also computable.

Hence, we only have to show that a computable set is listable.

The empty set is listable by definition. Otherwise, let $y \in A$ be any element. Then, the function f defined such that

$$f(x) = \begin{cases} x & \text{iff } C_A(x) = 1, \\ y & \text{iff } C_A(x) = 0, \end{cases}$$

for all $x \in \mathbb{N}$ is computable and has range A .

Conversely, assume that both A and \overline{A} are computably enumerable.

If either A or \overline{A} is empty, then A is computable.

Otherwise, let $A = f(\mathbb{N})$ and $\overline{A} = g(\mathbb{N})$, for some computable functions f and g .

We define the function C_A as follows:

$$C_A(x) = \begin{cases} 1 & \text{if } f(\min y[f(y) = x \vee g(y) = x]) = x, \\ 0 & \text{otherwise.} \end{cases}$$

The function C_A lists A and \overline{A} in parallel, waiting to see whether x turns up in A or in \overline{A} .

Note that x must eventually turn up either in A or in \overline{A} , so that C_A is a total computable function. \square

Our next goal is to show that the listable sets can be given several equivalent definitions. We will often abbreviate computably enumerable as *c.e.*

Lemma 9.9. *For any subset A of \mathbb{N} , the following properties are equivalent:*

- (1) *A is empty or A is the range of a primitive recursive function.*
- (2) *A is listable.*
- (3) *A is the range of a partial computable function.*
- (4) *A is the domain of a partial computable function.*

More intuitive proofs of the implications (3) \Rightarrow (4) and (4) \Rightarrow (1) can be given.

Assume that $A \neq \emptyset$ and that $A = \text{range}(g)$, where g is a partial computable function.

Assume that g is computed by a RAM program P .

To compute $f(x)$, we start computing the sequence

$$g(0), g(1), \dots$$

looking for x . If x turns up as say $g(n)$, then we output n .

Otherwise the computation diverges. Hence, the domain of f is the range of g .

Assume now that A is the domain of some partial computable function g , and that g is computed by some Turing machine M .

We construct another Turing machine performing the following steps:

(0) Do one step of the computation of $g(0)$

...

(n) Do $n + 1$ steps of the computation of $g(0)$

Do n steps of the computation of $g(1)$

...

Do 2 steps of the computation of $g(n - 1)$

Do 1 step of the computation of $g(n)$

During this process, whenever the computation of some $g(m)$ halts, we output m .

In this fashion, we will enumerate the domain of g , and since we have constructed a Turing machine that halts for every input, we have a total computable function.

The following Lemma can easily be shown using the proof technique of Lemma 9.9.

Lemma 9.10. *The following properties hold.*

(1) *There is a computable function h such that*

$$\text{range}(\varphi_x) = \text{dom}(\varphi_{h(x)})$$

for all $x \in \mathbb{N}$.

(2) *There is a computable function k such that*

$$\text{dom}(\varphi_x) = \text{range}(\varphi_{k(x)})$$

and $\varphi_{k(x)}$ is total computable, for all $x \in \mathbb{N}$ such that $\text{dom}(\varphi_x) \neq \emptyset$.

Using Lemma 9.9, we can prove that K is an c.e. set.

Indeed, we have $K = \text{dom}(f)$, where

$$f(x) = \varphi_{univ}(x, x)$$

for all $x \in \mathbb{N}$.

The set

$$K_0 = \{\langle x, y \rangle \mid \varphi_x(y) \text{ converges}\}$$

is also an c.e. set, since $K_0 = \text{dom}(g)$, where

$$g(z) = \varphi_{univ}(\Pi_1(z), \Pi_2(z)),$$

which is partial computable.

The sets K and K_0 are examples of c.e. sets that are not computable.

We can now prove that there are sets that are not c.e.

Lemma 9.11. *For any indexing of the partial computable functions, the complement \overline{K} of the set*

$$K = \{x \in \mathbb{N} \mid \varphi_x(x) \text{ converges}\}$$

is not listable.

Proof. If \overline{K} was listable, since K is also listable, by Lemma 9.8, the set K would be computable, a contradiction. \square

The sets \overline{K} and $\overline{K_0}$ are examples of sets that are not c.e.

This shows that the c.e. sets are not closed under complementation. However, we leave it as an exercise to prove that the c.e. sets are closed under union and intersection.

We will prove later on that TOTAL is not c.e.

This is rather unpleasant. Indeed, this means that there is no way of effectively listing all algorithms (all total computable functions).

Hence, in a certain sense, the concept of partial computable function (procedure) is more natural than the concept of a (total) computable function (algorithm).

The next two Lemmas give other characterizations of the c.e. sets and of the computable sets.

Lemma 9.12. *The following properties hold.*

- (1) *A set A is c.e. iff either it is finite or it is the range of an injective computable function.*
- (2) *A set A is c.e. if either it is empty or it is the range of a monotonic partial computable function.*
- (3) *A set A is c.e. iff there is a Turing machine M such that, for all $x \in \mathbb{N}$, M halts on x iff $x \in A$.*

Lemma 9.13. *A set A is computable iff either it is finite or it is the range of a strictly increasing computable function.*

Another important result relating the concept of partial computable function and that of a c.e. set is given below.

Theorem 9.14. *For every unary partial function f , the following properties are equivalent:*

(1) *f is partial computable.*

(2) *The set*

$$\{\langle x, f(x) \rangle \mid x \in \text{dom}(f)\}$$

is c.e.

Using our indexing of the partial computable functions and Lemma 9.9, we obtain an indexing of the c.e. sets.

Definition 9.5. For any acceptable indexing $\varphi_0, \varphi_1, \dots$ of the partial computable functions, we define the enumeration W_0, W_1, \dots of the c.e. sets by setting

$$W_x = \text{dom}(\varphi_x).$$

We now describe a technique for showing that certain sets are c.e. but not computable, or complements of c.e. sets that are not computable, or not c.e., or neither c.e. nor the complement of a c.e. set.

This technique is known as *reducibility*.

9.4 Reducibility and Complete Sets

We already used the notion of reducibility in the proof of Lemma 9.5 to show that TOTAL is not computable.

Definition 9.6. Let A and B be subsets of \mathbb{N} (or Σ^*). We say that the set A is *many-one reducible* to the set B if there is a *total computable* function $f: \mathbb{N} \rightarrow \mathbb{N}$ (or $f: \Sigma^* \rightarrow \Sigma^*$) such that

$$x \in A \quad \text{iff} \quad f(x) \in B \quad \text{for all } x \in \mathbb{N}.$$

We write $A \leq B$, and for short, we say that A is *reducible* to B .

Lemma 9.15. *Let A, B, C be subsets of \mathbb{N} (or Σ^*). The following properties hold:*

- (1) *If $A \leq B$ and $B \leq C$, then $A \leq C$.*
- (2) *If $A \leq B$ then $\overline{A} \leq \overline{B}$.*
- (3) *If $A \leq B$ and B is c.e., then A is c.e.*
- (4) *If $A \leq B$ and A is not c.e., then B is not c.e.*
- (5) *If $A \leq B$ and B is computable, then A is computable.*
- (6) *If $A \leq B$ and A is not computable, then B is not computable.*

Another important concept is the concept of a complete set.

Definition 9.7. A c.e. set A is *complete w.r.t. many-one reducibility* iff every c.e. set B is reducible to A , i.e., $B \leq A$.

For simplicity, we will often say *complete* for *complete w.r.t. many-one reducibility*.

Theorem 9.16. *The following properties hold:*

- (1) *If A is complete, B is c.e., and $A \leq B$, then B is complete.*
- (2) *K_0 is complete.*
- (3) *K_0 is reducible to K .*

As a corollary of Theorem 9.16, the set K is also complete.

Definition 9.8. Two sets A and B have the same *degree of unsolvability* or are *equivalent* iff $A \leq B$ and $B \leq A$.

Since K and K_0 are both complete, they have the same degree of unsolvability.

We will now investigate the reducibility and equivalence of various sets. Recall that

$$\text{TOTAL} = \{x \in \mathbb{N} \mid \varphi_x \text{ is total}\}.$$

We define EMPTY and FINITE, as follows:

$$\begin{aligned} \text{EMPTY} &= \{x \in \mathbb{N} \mid \varphi_x \text{ is undefined for all input}\}, \\ \text{FINITE} &= \{x \in \mathbb{N} \mid \varphi_x \text{ has a finite domain}\}. \end{aligned}$$

Then,

$$\overline{\text{FINITE}} = \{x \in \mathbb{N} \mid \varphi_x \text{ has an infinite domain}\},$$

so that,

$$\text{EMPTY} \subset \text{FINITE} \text{ and } \text{TOTAL} \subset \overline{\text{FINITE}}.$$

Lemma 9.17. *We have $K_0 \leq \overline{\text{EMPTY}}$.*

Lemma 9.18. *The following properties hold:*

- (1) *EMPTY is not c.e.*
- (2) *$\overline{\text{EMPTY}}$ is c.e.*
- (3) *\overline{K} and EMPTY are equivalent.*
- (4) *$\overline{\text{EMPTY}}$ is complete.*

Lemma 9.19. *The following properties hold:*

- (1) TOTAL and $\overline{\text{TOTAL}}$ are not c.e.
- (2) FINITE and $\overline{\text{FINITE}}$ are not c.e.

From Lemma 9.19, we have $\text{TOTAL} \leq \overline{\text{FINITE}}$. It turns out that $\overline{\text{FINITE}} \leq \text{TOTAL}$, and TOTAL and $\overline{\text{FINITE}}$ are equivalent.

Lemma 9.20. *The sets TOTAL and $\overline{\text{FINITE}}$ are equivalent.*

We now turn to the recursion Theorem.

9.5 The Recursion Theorem

The recursion Theorem, due to Kleene, is a fundamental result in recursion theory.

Theorem 9.21. (*Recursion Theorem, Version 1*) *Let $\varphi_0, \varphi_1, \dots$ be any acceptable indexing of the partial computable functions. For every total computable function f , there is some n such that*

$$\varphi_n = \varphi_{f(n)}.$$

The recursion Theorem can be strengthened as follows.

Theorem 9.22. (*Recursion Theorem, Version 2*) *Let $\varphi_0, \varphi_1, \dots$ be any acceptable indexing of the partial computable functions. There is a total computable function h such that for all $x \in \mathbb{N}$, if φ_x is total, then*

$$\varphi_{\varphi_x(h(x))} = \varphi_{h(x)}.$$

A third version of the recursion Theorem is given below.

Theorem 9.23. (*Recursion Theorem, Version 3*) For all $n \geq 1$, there is a total computable function h of $n + 1$ arguments, such that for all $x \in \mathbb{N}$, if φ_x is a total computable function of $n + 1$ arguments, then

$$\varphi_{\varphi_x(h(x,x_1,\dots,x_n),x_1,\dots,x_n)} = \varphi_{h(x,x_1,\dots,x_n)},$$

for all $x_1, \dots, x_n \in \mathbb{N}$.

As a first application of the recursion theorem, we can show that there is an index n such that φ_n is the constant function with output n .

Loosely speaking, φ_n prints its own name. Let f be the computable function such that

$$f(x, y) = x$$

for all $x, y \in \mathbb{N}$.

By the s-m-n Theorem, there is a computable function g such that

$$\varphi_{g(x)}(y) = f(x, y) = x$$

for all $x, y \in \mathbb{N}$.

By the recursion Theorem 9.21, there is some n such that

$$\varphi_{g(n)} = \varphi_n,$$

the constant function with value n .

As a second application, we get a very short proof of Rice's Theorem.

Let C be such that $P_C \neq \emptyset$ and $P_C \neq \mathbb{N}$, and let $j \in P_C$ and $k \in \mathbb{N} - P_C$.

Define the function f as follows:

$$f(x) = \begin{cases} j & \text{if } x \notin P_C, \\ k & \text{if } x \in P_C, \end{cases}$$

If P_C is computable, then f is computable. By the recursion Theorem 9.21, there is some n such that

$$\varphi_{f(n)} = \varphi_n.$$

But then, we have

$$n \in P_C \quad \text{iff} \quad f(n) \notin P_C$$

by definition of f , and thus,

$$\varphi_{f(n)} \neq \varphi_n,$$

a contradiction.

Hence, P_C is not computable.

As a third application, we have the following Lemma.

Lemma 9.24. *Let C be a set of partial computable functions and let*

$$A = \{x \in \mathbb{N} \mid \varphi_x \in C\}.$$

The set A is not reducible to its complement \overline{A} .

The recursion Theorem can also be used to show that functions defined by recursive definitions other than primitive recursion are partial computable.

This is the case for the function known as *Ackermann's function*, defined recursively as follows:

$$\begin{aligned} f(0, y) &= y + 1, \\ f(x + 1, 0) &= f(x, 1), \\ f(x + 1, y + 1) &= f(x, f(x + 1, y)). \end{aligned}$$

It can be shown that this function is not primitive recursive. Intuitively, it outgrows all primitive recursive functions.

However, f is computable, but this is not so obvious.

We can use the recursion Theorem to prove that f is computable. Consider the following definition by cases:

$$\begin{aligned} g(n, 0, y) &= y + 1, \\ g(n, x + 1, 0) &= \varphi_{univ}(n, x, 1), \\ g(n, x + 1, y + 1) &= \varphi_{univ}(n, x, \varphi_{univ}(n, x + 1, y)). \end{aligned}$$

Clearly, g is partial computable. By the s-m-n Theorem, there is a computable function h such that

$$\varphi_{h(n)}(x, y) = g(n, x, y).$$

By the recursion Theorem, there is an m such that

$$\varphi_{h(m)} = \varphi_m.$$

Therefore, the partial computable function $\varphi_m(x, y)$ satisfies the definition of Ackermann's function.

We showed in a previous Section that $\varphi_m(x, y)$ is a total function, and thus, Ackermann's function is a total computable function.

Hence, the recursion Theorem justifies the use of certain recursive definitions. However, note that there are some recursive definition that are only satisfied by the completely undefined function.

In the next Section, we prove the extended Rice Theorem.

9.6 Extended Rice Theorem

The extended Rice Theorem characterizes the sets of partial computable functions C such that P_C is c.e.

First, we need to discuss a way of indexing the partial computable functions that have a finite domain.

Using the uniform projection function Π , we define the primitive recursive function F such that

$$F(x, y) = \Pi(y + 1, \Pi_1(x) + 1, \Pi_2(x)).$$

We also define the sequence of partial functions P_0, P_1, \dots as follows:

$$P_x(y) = \begin{cases} F(x, y) - 1 & \text{if } 0 < F(x, y) \text{ and } y < \Pi_1(x) + 1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Lemma 9.25. *Every P_x is a partial computable function with finite domain, and every partial computable function with finite domain is equal to some P_x .*

The easy part of the extended Rice Theorem is the following lemma.

Recall that given any two partial functions $f: A \rightarrow B$ and $g: A \rightarrow B$, we say that g *extends* f iff $f \subseteq g$, which means that $g(x)$ is defined whenever $f(x)$ is defined, and if so, $g(x) = f(x)$.

Lemma 9.26. *Let C be a set of partial computable functions. If there is an c.e. set A such that, $\varphi_x \in C$ iff there is some $y \in A$ such that φ_x extends P_y , then $P_C = \{x \mid \varphi_x \in C\}$ is c.e.*

Proof. Lemma 9.26 can be restated as

$$P_C = \{x \mid \exists y \in A, P_y \subseteq \varphi_x\}.$$

If A is empty, so is P_C , and P_C is c.e.

Otherwise, let f be a computable function such that

$$A = \text{range}(f).$$

Let ψ be the following partial computable function:

$$\psi(z) = \begin{cases} \Pi_1(z) & \text{if } P_{f(\Pi_2(z))} \subseteq \varphi_{\Pi_1(z)}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is clear that

$$P_C = \text{range}(\psi).$$

To see that ψ is partial computable, write $\psi(z)$ as follows:

$$\psi(z) = \begin{cases} \Pi_1(z) & \text{if } \forall w \leq \Pi_1(f(\Pi_2(z))) \\ & [F(f(\Pi_2(z)), w) > 0 \supset \\ & \varphi_{\Pi_1(z)}(w) = F(f(\Pi_2(z)), w) - 1], \\ \text{undefined} & \text{otherwise.} \end{cases}$$

□

To establish the converse of Lemma 9.26, we need two Lemmas.

Lemma 9.27. *If P_C is c.e. and $\varphi \in C$, then there is some $P_y \subseteq \varphi$ such that $P_y \in C$.*

As a corollary of Lemma 9.27, we note that TOTAL is not c.e.

Lemma 9.28. *If P_C is c.e., $\varphi \in C$, and $\varphi \subseteq \psi$, where ψ is a partial computable function, then $\psi \in C$.*

Observe that Lemma 9.28 yields a new proof that $\overline{\text{TOTAL}}$ is not c.e. Finally, we can prove the extended Rice Theorem.

Theorem 9.29. (*Extended Rice Theorem*) *The set P_C is c.e. iff there is a c.e. set A such that*

$$\varphi_x \in C \quad \text{iff} \quad \exists y \in A (P_y \subseteq \varphi_x).$$

Proof. Let $P_C = \text{dom}(\varphi_i)$. Using the s-m-n Theorem, there is a computable function k such that

$$\varphi_{k(y)} = P_y$$

for all $y \in \mathbb{N}$.

Define the c.e. set A such that

$$A = \text{dom}(\varphi_i \circ k).$$

Then,

$$y \in A \quad \text{iff} \quad \varphi_i(k(y)) \downarrow \quad \text{iff} \quad P_y \in C.$$

Next, using Lemma 9.27 and Lemma 9.28, it is easy to see that

$$\varphi_x \in C \quad \text{iff} \quad \exists y \in A (P_y \subseteq \varphi_x).$$

□

9.7 Creative and Productive Sets

In this section, we discuss some special sets that have important applications in logic: *creative and productive sets*.

The concepts to be described are illustrated by the following situation. Assume that

$$W_x \subseteq \overline{K}$$

for some $x \in \mathbb{N}$.

We claim that

$$x \in \overline{K} - W_x.$$

Indeed, if $x \in W_x$, then $\varphi_x(x)$ is defined, and by definition of K , we get $x \notin \overline{K}$, a contradiction.

Therefore, $\varphi_x(x)$ must be undefined, that is,

$$x \in \overline{K} - W_x.$$

The above situation can be generalized as follows.

Definition 9.9. A set A is *productive* iff there is a total computable function f such that

$$\text{if } W_x \subseteq A \text{ then } f(x) \in A - W_x$$

for all $x \in \mathbb{N}$. The function f is called the *productive function of A* . A set A is *creative* if it is c.e. and if its complement \overline{A} is productive.

As we just showed, K is creative and \overline{K} is productive. The following facts are immediate consequences of the definition.

- (1) A productive set is not c.e.
- (2) A creative set is not computable.

Creative and productive sets arise in logic.

The set of theorems of a logical theory is often creative. For example, the set of theorems in Peano's arithmetic is creative. This yields incompleteness results.

Lemma 9.30. *If a set A is productive, then it has an infinite c.e. subset.*

Another important property of productive sets is the following.

Lemma 9.31. *If a set A is productive, then $\overline{K} \leq A$.*

Using Lemma 9.31, the following results can be shown.

Lemma 9.32. *The following facts hold.*

- (1) *If A is productive and $A \leq B$, then B is productive.*
- (2) *A is creative iff A is equivalent to K .*
- (3) *A is creative iff A is complete,*

