Chapter 6

Elementary Recursive Function Theory

6.1 Acceptable Indexings

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

Using this indexing, we showed the existence of a universal function φ_{univ} and of a recursive 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 6.1.1 An *indexing* of the partial recursive functions is an infinite sequence $\varphi_0, \varphi_1, \ldots$, of partial recursive functions that includes all the partial recursive 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 recursive 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 recursive 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 $\varphi(x_1, \ldots, x_n)$ for $\varphi(\langle x_1, \ldots, x_n \rangle)$, 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 recursive fashion upon the index of φ and the first m arguments $x_1, \ldots, x_m$.

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

Theorem 6.1.2 *(The "s-m-n Theorem") For any acceptable indexing $\varphi_0, \varphi_1, \ldots$, there is a total recursive function s, such that, for all $i, m, n \geq 1$, for all $x_1, \ldots, x_m$ and all $y_1, \ldots, y_n$, we have*

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

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

Theorem 6.1.3 *Let $\varphi_0, \varphi_1, \ldots,$ be a universal indexing, and let $\psi_0, \psi_1, \ldots,$ be any indexing with a total recursive 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 recursive function t such that $\varphi_i = \psi_{t(i)}$.

Using Theorem 6.1.3, if we have two acceptable indexings $\varphi_0, \varphi_1, \ldots,$ and $\psi_0, \psi_1, \ldots,$ there exist total recursive 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 recursive *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.

6.2 Undecidable Problems

We saw in Section 5.2 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 recursive functions.

Theorem 6.2.1 *(Halting Problem, Abstract Version) Let $\psi_0, \psi_1, \ldots,$ be any indexing of the partial recursive 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 recursive.

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

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

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

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 recursive. 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 recursive, 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 recursive. $\square$

The function g defined in the proof of Theorem 6.2.1 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 recursive. Since this fact is quite important, we give the following definition.

Definition 6.2.2 A subset of Σ^* (or a subset of $\mathbb{N}$) is *recursive* iff its characteristic function is a total recursive function.

Using Definition 6.2.2, Theorem 6.2.1 can be restated as follows.

Lemma 6.2.3 *For any indexing $\varphi_0, \varphi_1, \ldots$ of the partial recursive functions (over Σ^* or $\mathbb{N}$), the set $K = \{x \mid \varphi_x(x) \text{ is defined}\}$ is not recursive.*

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

The idea is to generalize the situation described in Section 5.2 and Section 5.3, where a set of objects, the RAM programs, is encoded into a set of natural numbers, using a coding scheme.

Definition 6.2.4 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 $\#: C \to \mathbb{N}$ 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 recursive.

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

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

is not recursive.

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 decidabe 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 6.2.5 *For any acceptable indexing $\varphi_0, \varphi_1, \ldots$ of the partial recursive functions, the set*

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

is not recursive.

Proof. The proof uses a technique known as reducibility.

We try to reduce a set A known to be *nonrecursive* to TOTAL via a recursive function $f\colon A \to \text{TOTAL}$, so that

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

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

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

To find the recursive function $f\colon K \to \text{TOTAL}$, 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 recursive. 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 recursive 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 recursive. As we explained earlier, this shows that TOTAL is not recursive. $\square$

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

Let $\varphi_0, \varphi_1, \ldots$ be any indexing of the partial recursive functions, and let C be any set of partial recursive 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 recursive functions. For example

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

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

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

Theorem 6.2.6 *(Rice's Theorem) For any acceptable indexing $\varphi_0, \varphi_1, \ldots$ of the partial recursive functions, for any set C of partial recursive functions, the set*

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

is nonrecursive unless C is trivial.

Proof. Assume that C is nontrivial. A set is recursive iff its complement is recursive (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 recursive 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 recursive, 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 recursive 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 recursive, since otherwise, K would be recursive, 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 6.2.7 *The following properties of partial recursive functions are undecidable.*

(a) A partial recursive function is a constant function.

(b) Given any integer $y \in \mathbb{N}$, is y in the range of some partial recursive function.

(c) Two partial recursive functions φ_x and φ_y are identical.

(d) A partial recursive function φ_x is equal to a given partial recursive function φ_a.

(e) A partial recursive function yields output z on input y, for any given $y, z \in \mathbb{N}$.

(f) A partial recursive function diverges for some input.

(g) A partial recursive function diverges for all input.

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

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

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