# Chapter 3

# Elementary Recursive Function Theory

## 3.1 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 $\varphi_{univ}$ 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 3.1.** An *indexing* of the partial computable functions is an infinite sequence $\varphi_0, \varphi_1, \ldots,$ 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 $\varphi_{univ}$ 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 $\varphi(x_1, \ldots, x_n)$ for $\varphi(\langle x_1, \ldots, x_n \rangle)$, the s-m-n theorem says the following.

Given a function $\varphi$ 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 $\psi$ of $n$ arguments. Then, the index of $\psi$ depends in a computable fashion upon the index of $\varphi$ and the first $m$ arguments $x_1, \ldots, x_m$.

We can "pull" the first $m$ arguments of $\varphi$ into the index of $\psi$.

**Theorem 3.1.** *(The "s-m-n Theorem") For any acceptable indexing $\varphi_0, \varphi_1, \ldots$, there is a total computable 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 3.2.** *Let $\varphi_0, \varphi_1, \ldots,$ be a universal indexing, and let $\psi_0, \psi_1, \ldots,$ 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 3.2, if we have two acceptable indexings $\varphi_0, \varphi_1, \ldots,$ and $\psi_0, \psi_1, \ldots,$ 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.

## 3.2 Undecidable Problems

We saw in Section 2.2 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 3.3.** *(Halting Problem, Abstract Version) Let $\psi_0, \psi_1, \ldots,$ 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 $\theta$ 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 $\theta$ is not even partial computable. Observe that $\theta$ 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 $\theta$ was partial computable, it would occur in the list as some $\psi_i$, 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. □

The function $g$ defined in the proof of Theorem 3.3 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 3.2.** A subset of $\Sigma^*$ (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 3.2, Theorem 3.3 can be restated as follows.

**Lemma 3.4.** *For any indexing $\varphi_0, \varphi_1, \ldots$ of the partial computable functions (over $\Sigma^*$ or $\mathbb{N}$), the set $K = \{x \mid \varphi_x(x)$ 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 2.2 and Section 2.6, where a set of objects, the RAM programs, is encoded into a set of natural numbers, using a coding scheme.

**Definition 3.3.** 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 $\#\colon C \to \mathbb{N}$ 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 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 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 $\Sigma$ (sets of trees, graphs, etc).

For example, the equality of the partial functions $\varphi_x$ and $\varphi_y$ 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.

## 3.3 Reducibility and Rice's Theorem

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 3.5.** *For any acceptable indexing $\varphi_0, \varphi_1, \ldots$ of the partial computable functions, the set*

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

*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 $f\colon A \to \text{TOTAL}$, 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 $f\colon K \to \text{TOTAL}$, we use the s-m-n Theorem.

Let $\theta$ 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 $\theta$ does not depend on $y$.

The function $\theta$ is partial computable. Indeed, we have

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

Thus, $\theta$ 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. □

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 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 3.6.** *(Rice's Theorem) For any acceptable indexing $\varphi_0, \varphi_1, \ldots$ 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 $\psi$ 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 $\theta$ 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, $\theta$ 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 $\theta$, 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. □

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 3.7.** *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 $\varphi_x$ and $\varphi_y$ are identical.*

*(d) A partial computable function $\varphi_x$ is equal to a given partial computable function $\varphi_a$.*

*(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.

## 3.4 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 $\Pi_1$ and $\Pi_2$ 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 3.4.** 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 $\Sigma^*$ 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 3.8.** *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. □

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 3.9.** *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 $A = range(g)$, 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

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

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 3.9.

**Lemma 3.10.** *The following properties hold.*

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

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

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

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

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

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

Using Lemma 3.9, we can prove that $K$ is an c.e. set.

Indeed, we have $K = dom(f)$, 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 $K_0 = dom(g)$, 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 3.11.** *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 3.8, the set $K$ would be computable, a contradiction. □

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 3.12.** *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 3.13.** *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 3.14.** *For every unary partial function $f$, the following properties are equivalent:*

*(1) $f$ is partial computable.*

*(2) The set*

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

*is c.e.*

Using our indexing of the partial computable functions and Lemma 3.9, we obtain an indexing of the c.e. sets.

**Definition 3.5.** For any acceptable indexing $\varphi_0, \varphi_1, \ldots$ of the partial computable functions, we define the enumeration $W_0, W_1, \ldots$ of the c.e. sets by setting

$$W_x = dom(\varphi_x).$$

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*.

## 3.5 Reducibility and Complete Sets

We already used the notion of reducibility in the proof of Lemma 3.5 to show that TOTAL is not computable.

**Definition 3.6.** Let $A$ and $B$ be subsets of $\mathbb{N}$ (or $\Sigma^*$). We say that the set $A$ is *many-one reducible* to the set $B$ if there is a *total computable* function $f : \mathbb{N} \to \mathbb{N}$ (or $f : \Sigma^* \to \Sigma^*$) 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 3.15.** *Let $A, B, C$ be subsets of $\mathbb{N}$ (or $\Sigma^*$). 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 3.7.** 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 3.16.** *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 3.16, the set $K$ is also complete.

**Definition 3.8.** 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,

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

**Lemma 3.17.** *We have* $K_0 \leq \overline{\text{EMPTY}}$.

**Lemma 3.18.** *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 3.19.** *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 3.19, we have TOTAL $\leq \overline{\text{FINITE}}$. It turns out that $\overline{\text{FINITE}} \leq$ TOTAL, and TOTAL and $\overline{\text{FINITE}}$ are equivalent.

**Lemma 3.20.** *The sets* TOTAL *and* $\overline{\text{FINITE}}$ *are equivalent.*