DET NATURVIDENSKABELIGE FAKULTET
KØBENHAVNS UNIVERSITET

Bachelorprojekt i Matematik

Institut for Matematiske Fag, Københavns Universitet

Bachelor's Thesis in Mathematics

Department of Mathematical Sciences, University of Copenhagen

Hans Erik Bugge Grathwohl

Recursion Theory, Turing Degrees and Post's Problem

Vejleder: Mikael Rørdam

Ekstern Vejleder: Asger Dag Törnquist

Afleveringsdato: 10. juni 2011

Abstract

This thesis is an introduction to classical recursion theory. We will define two notions of computable functions, the Turing computable functions and the partial recursive functions, and argue that these notions are equivalent. The fundamental results and some historically significant theorems concerning these functions are proven, for example Kleene's Recursion Theorem and Rice's Theorem. We will define the notion of recursively enumerable sets, and characterize these in different ways. The notion of computability will then be relativized, such that we can classify sets and functions according to how much information is required to compute them, and we define Turing degrees and relate this to the arithmetical hierarchy. The finite injury priority method is then demonstrated twice, and with this we will prove the Friedberg-Muchnik Theorem and solve Post's problem by confirming the existence of r.e. degrees $\mathbf{0} < \mathbf{a} < \mathbf{0}'$.

Resumé

Dette projekt er en introduktion til klassisk rekursionsteori. Vi vil definere to former for beregnelige funktioner, de Turing-beregnelige funktioner og de partielle rekursive funktioner, og argumentere for at disse to begreber er ækvivalente. De fundamentale resultater samt nogle sætninger af historisk betydning vil blive vist, for eksempel Kleenes Rekursionssætning og Rices sætning. Vi vil definere de rekursivt nummererbare mængder og karakterisere disse på forskellige måder. Beregnelighedsbegrebet bliver så relativiseret, sådan at vi kan klassificere mængder og funktioner udfra hvor meget information der er nødvendigt for at kunne beregne dem, og vi vil definere Turinggrader og relatere disse til det aritmetiske hierarki. Den endeligt afmærkende prioritetsmetode bliver demonstreret to gange, og denne vil benyttes til at bevise Friedberg-Muchniks sætning og løse Posts problem ved at bekræfte eksistensen af rekursivt nummererbare grader $\mathbf{0} < \mathbf{a} < \mathbf{0}'$.

Contents

Introduction

The purpose of this bachelor's thesis is to be an introduction to *classical recursion theory*. Classical recursion theory is the study of computable functions on the non-negative integers. It emerged in the time following Kurt Gödel's 1931 incompleteness theorem, which used primitive recursive functions in its proof, and it was pioneered primarily by Alonzo Church, Kurt Gödel, Stephen Kleene, Emil Post, and Alan Turing.

The first two chapters will concern the early results, roughly originating from the period 1931–1943. Many results from this period, however, are omitted. In particular, only one kind of reducibility is described, the Turing reducibility, whereas many-one reducibility and one-one reducibility are left out due to space considerations.

The first chapter will focus on results concerning partial recursive functions and recursively enumerable sets. The second chapter will concern the relativization of the concepts from Chapter 1, and will describe methods to classify sets, especially by describing the Arithmetical Hierarchy and relating this to the Turing degrees. Lastly, the third chapter will solve Post's problem by using a proof technique known as the *finite injury priority method*, a proof method invented in 1956 which turned out to be a very powerful technique, as many advanced results were later proven with variants of this method, especially with the variant known as the *infinite* injury priority argument.

The primary source for this thesis is Robert Soare's 1987 book [10], and many of the proofs, and most of the notation, originates from here. The third chapter, however, is based on Steffen Lempp's notes on priority arguments [5], because his versions of the priority arguments has a good intuition behind them, and because the framework he develops is well suited for harder priority arguments.

Notation. If nothing else is stated, then all sets will consists of non-negative integers $\omega = \{0, 1, 2, \ldots\}$. Whenever we refer to a number or integer, it is implicit that this means such a non-negative integer. Functions will always be from ω^n to ω, $n \geq 1$, total functions will generally be identified with lowercase letters f, g, and partial functions with lowercase greek letters φ, ψ.

Chapter 1

Computability and Recursively Enumerable Sets

In this chapter, some of the most basic concepts of interest in the study of recursion theory are introduced.

Recursion theory concerns *computable functions*. Informally, a function is computable if it is possible to determine its output in an *effective manner*, i.e., there has to be a finite set of instructions, which with an input x after a finite amount of steps yields the output y. To be able to study these functions we require a specific model of computability. We will introduce two different models of computability. One of these is *Turing computability*, introduced in 1936 by A.M. Turing in [12]. Here he introduced a machine, which was inspired by the observation of how a human person would compute the value of a function, and which limitations he would have. For example when performing the computation, he can only have a finite number of symbols under observation at a given time, and his memory is also finite, etc. This machine is now called a *Turing machine* and it can in essence perform the same atomic acts as a human computer.

The other model which has roots in Peano arithmetic concerns the *recursive functions* and is a more classical mathematical approach to the problem. Both models have its strengths. The Turing model contains some great analogies to how a result of some problem is calculated "in real life" by some human or computer, while the recursive functions are closely connected to arithmetic. As it turns out, we will not have to choose one model over the other, for they define the same notion of computability.

1.1 Turing Computability

We will define a *Turing machine* M as in [10]. It consists of a two-way infinite *tape* divided into *cells*, each containing a symbol from $S = \{B, 1\}$ (blank or 1), a *reading head* which can scan one cell at a time, and a finite set of internal *states* $Q = \{q_0, q_1, \dots, q_n\}$, $n \geq 1$. At each step, the machine can: change from one state to another; change the scanned symbol s to another symbol $s' \in S$; and move the reading head one step to the right (R) or left (L) on the tape. M is then controlled by a partial function

$$\delta : Q \times S \to Q \times S \times \{R, L\}$$

called the *transition function*, which, given the current state and the scanned symbol, tells the machine what to do in the next step. We will view δ as a finite list of quintuples $(q, s, q', s', X) \in \delta$, which we regard as the *Turing program.* The *input* integers $x_1, \dots, x_n$ to the machine are represented by strings of $x_i + 1$ consecutive 1's separated by a B, where all other cells are blank.

The machine begins in the *starting state* q_1, where the reading head is scanning the left-most 1 of the input. If the machine ever reaches the *halting state* q_0, we say that the machine *halts* with the *output* given by the total number of 1's on the tape.

We define a *configuration* c of M at a given step to be: the current state; the symbol being scanned; the symbols to the right of the reading head; and the symbols to the left of the reading head, viz.,

$$s_{-m} \ \cdots \ s_{-1} \ q_i \ s_0 \ s_1 \ \cdots \ s_n.$$

A *Turing computation* according to a halting Turing program P with input x is then a finite sequence of configurations, $c_0, c_1, \dots, c_n$, c_0 being the starting configuration, and c_n being the configuration when the halting state is reached. We arrive at the following definition.

Definition 1.1.1. A partial function ψ of n variables is *Turing computable* if there is a Turing program P, such that P with $x_1, \dots, x_n$ as input halts with output y iff $\psi(x_1, \dots, x_n) = y$.

Here we use the following definition of a partial function:

Definition 1.1.2. A *partial function* ψ on ω is a function $\psi : A \to \omega$ with $A \subseteq \omega$. If $x \in A$ we say that $\psi(x)$ is defined, and write it $\psi(x)\downarrow$, $\psi(x)\downarrow= y$ or just $\psi(x) = y$. If $x \notin A$ we say $\psi(x)$ is undefined and write it $\psi(x)\uparrow$. If $A = \omega$ we say that ψ is *total.*

1.2 Recursive Functions

Another way of defining a class of computable function, is by defining them recursively from a basic set of functions. We start by defining the *primitive recursive functions*, and then extend this to the *partial recursive functions*, following [10].

In the following definition, we introduce the three basic function types, and then two rules of construction which are used to build more advanced functions up from the basic functions.

Definition 1.2.1. We define the class of primitive recursive functions as the smallest class $\mathcal{C}$ of functions $\omega^n \to \omega$, $n \in \omega$, closed under the following schemata:

(i) The *successor function*, $S(x) = x + 1$ is in $\mathcal{C}$.

(ii) The *constant functions*, $C^n_m(x_1, \dots, x_n) = m$ are in $\mathcal{C}$, where $n, m \geq 0$.

(iii) The *projections*, $P^n_i(x_1, \dots, x_n) = x_i$, are in $\mathcal{C}$, where $n \geq 1$, $1 \leq i \leq n$.

(iv) (Composition) If $g_1, g_2, \dots, g_m$ and $h \in \mathcal{C}$, then

$$f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

is in $\mathcal{C}$ where $g_1, \dots, g_m$ are functions of n variables and h is a function of m variables.

(v) (Primitive Recursion) If $g, h \in \mathcal{C}$ and $n \geq 1$ then $f \in \mathcal{C}$ where

$$f(0, x_2, \dots, x_n) = g(x_2, \dots, x_n)$$

$$f(x_1 + 1, x_2, \dots, x_n) = h(x_1, f(x_1, x_2, \dots, x_n), x_2, \dots, x_n)$$

assuming g and h are functions of $n - 1$ and $n + 1$ variables respectively.

Contained in this $\mathcal{C}$ are now all the usual functions from basic number theory. A function f is in $\mathcal{C}$ if there is a *derivation* $f_1, \dots, f_n = f$, where each f_i is either one of the primitive functions in (i), (ii), (iii), or obtained from $\{f_j : j < i\}$ by (iv) or (v).

Consider for example multiplication. Let $f(x, y) = x \cdot y$. We see that $f(0, y) = 0 \cdot y = 0$, and $f(x + 1, y) = (x + 1) \cdot y = x \cdot y + x = f(x, y) + x$. So f has the following derivation. The applied schemata are noted to the right.

$$\begin{array}{lr}
f_1(x) = x + 1 & \text{(i)} \\
f_2(x_1, x_2) = x_1 & \text{(iii)} \\
f_3(x_1, x_2, x_3, x_4) = x_2 & \text{(iii)} \\
f_4 = f_1 \circ f_3 & \text{(iv)} \\
f_5(0, x_2, x_3) = f_2(x_2, x_3) & \\
f_5(x_1 + 1, x_2, x_3) = f_4(x_1, f_5(x_1, x_2, x_3), x_2, x_3) & \text{(v)} \\
f_6(x) = 0 & \text{(ii)} \\
f_7(0, x_2) = f_6(x_2) & \\
f_7(x_1 + 1, x_2) = f_5(x_1, f_7(x_1, x_2), x_2) & \text{(v)}
\end{array}$$

Note that $f_5(x_1, x_2, x_3) = x_1 + x_2$, so $f_7 = f$, and thus $f_1, \dots, f_7$ is a primitive recursive derivation of multiplication. On pp. 222-223 of [3], all of the usual number theoretic functions of ω are listed, and they are primitive recursive.

But our motivation for introducing recursive functions is not only to be able to recreate arithmetic—the goal is to include all computable functions in this mathematical notion. The primitive recursive functions, though, are not strong enough for this, as we will see.

Every primitive recursive functions can be derived in a finite number of steps from (i)-(v). Thus we can make a *listing* of all primitive recursive functions, i.e., a surjective map from ω to the class of primitive recursive functions (a useful technique called *encoding* which could be used for this, is introduced below). Let f_n denote the n'th primitive recursive function in this listing. Now consider the function $g(x) = f_x(x) + 1$. This is clearly computable—just compute $f_x(x)$ and add one—but we see that $f_n \neq g$ for all n, and thus g is not primitive recursive. This is an example of an argument which is called *Cantor's diagonal method*, or a *diagonal argument.*

To avoid this problem, we introduce the notion of partial recursive functions.

Definition 1.2.2. We define the class of *partial recursive functions* (*p.r. functions*) to be the smallest class $\mathcal{C}$ of partial functions closed under the schemata (i)-(v) of Definition 1.2.1, as well as the schema

(vi) (Unbounded Search) If $\theta(x_1,\dots,x_n,y) \in \mathcal{C}$ and

$$\psi(x_1,\dots,x_n) = \mu y\ [\theta(x_1,\dots,x_n,y)\downarrow = 0\ \wedge\ (\forall z \le y)[\theta(x_1,\dots,x_n,z)\downarrow]]$$

then $\psi \in \mathcal{C}$.

Here $\mu y\ R(y)$ means *the least y that fulfills $R(y)$*.

When a partial recursive function is total, we either call it a *total recursive function* or just a *recursive function.*

The afore mentioned diagonal argument does not work on these p.r. functions. We can still make a listing of these functions. Let θ_n be the n'th p.r. function under this listing, and let $\psi(x) = \theta_x(x) + 1$ if $\theta_x(x)\downarrow$ and $\psi(x)\uparrow$ otherwise. Now we can have a x_0 such that $\theta_{x_0} = \psi$ without problems, since $\theta_{x_0}(x_0)$ can just be undefined.

Definition 1.2.3. When we say that a relation or a set has some property (for example "$R \subseteq \omega^n$ is primitive recursive" or "$A \subseteq \omega$ is recursive") then we mean that the characteristic function has this property (χ_R is primitive recursive, χ_A is total recursive).

By characteristic function, we refer to the standard definition where χ_A is the characteristic function of A iff

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

We can view a set of natural numbers as being a *decision problem*, where we can ask if a number belongs to the set or not. Thus, a set A is *decidable* iff it is recursive, as this means we for any number x in an effective manner can decide whether $x \in A$ by computing $\chi_A(x)$. If a set is not recursive, it is *undecidable*.

1.3 The Church-Turing Thesis

Definition 1.3.1. We define an *encoding* of finite sequences of numbers to be an injective function $E : \omega^{<\omega} \to \omega$, and we say that a number is a *sequence number* if it is in the range of E. We call E primitive recursive if $E(\omega^{<\omega})$ is primitive recursive, and if the following operations are primitive recursive:

- The restriction $E_n =_{\text{def}} E|\omega^n$ of E to sequences of length $n+1$.
- The length function

$$\text{lh}(x) = \begin{cases} z & \text{if } E^{-1}(x) \text{ exists and have length } z, \\ 0 & \text{otherwise.} \end{cases}$$

- The extractor function

$$(x)_y = \begin{cases} z & \text{if } E^{-1}(x) \text{ exists, with } z \text{ being its } y\text{'th number,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1.3.2. We define the *pairing encoding*, by using *Cantor's pairing function* $\langle \cdot, \cdot \rangle : \omega^2 \to \omega$,

$$\langle x, y \rangle = \frac{1}{2}(x+y)(x+y+1) + x,$$

which is a primitive recursive bijection. First we expand this to $n \geq 2$ by induction:

$$\langle a_1, \dots, a_n \rangle_n = \langle a_1, \langle a_2, \dots, a_n \rangle_{n-1} \rangle$$

where $\langle \cdot, \cdot \rangle_2 = \langle \cdot, \cdot \rangle$, and we let $\langle a_1 \rangle_1 = a_1$. By induction, these are also bijective and primitive recursive. Now define E by:

$$E() = 0$$
$$E(a_1, \dots, a_n) = \langle n, \langle a_1, \dots, a_n \rangle_n \rangle.$$

Theorem 1.3.3. *The pairing encoding is primitive recursive.*

Proof. E is bijective, so $x \in E(\omega^{<\omega}) \iff x \in \omega$, thus the image is primitive recursive.

E_n is clearly primitive recursive. Let $f : \omega \to \omega^2$ be the inverse of $\langle \cdot, \cdot \rangle$, and denote by f_1, f_2 the two coordinate functions of f, such that $\langle f_1(x), f_2(x) \rangle = x$ for all x. These are also primitive recursive, and so especially is $\text{lh}(x) = f_1(x)$.

To extract a number from a sequence, note that if $\langle a_1, \ldots, a_n \rangle = b$, then $a_1 = f_1(b)$, $a_2 = f_1 \circ f_2(b)$, and so $a_i = f_1 \circ f_2^{i-1}(b)$ for $i < n$ and $a_n = f_2^n(b)$. So

$$(x)_y = \begin{cases} f_1 \circ f_2^y(x) & \text{if } y < \mathrm{lh}(x), \\ f_2^y(x) & \text{if } y = \mathrm{lh}(x), \\ 0 & \text{otherwise,} \end{cases}$$

is primitive recursive. □

We will now use the pairing encoding to assign code numbers to the Turing programs and all possible Turing computations.

Recall the fact that every Turing program is fully determined by a finite set of quintuples.

Every sequence of numbers $a_1, \ldots, a_k$ can be identified with such a quintuple in the following way. If $k \geq 5$, then let

$$(q_{a_1}, s_{a_2}, q_{a_3}, s_{a_4}, r_{a_5}) \sim a_1, \ldots, a_k$$

where

$$s_i = \begin{cases} B & \text{if } i \equiv 0 \mod 2 \\ 1 & \text{otherwise,} \end{cases}$$

and

$$r_i = \begin{cases} R & \text{if } i \equiv 0 \mod 2 \\ L & \text{otherwise.} \end{cases}$$

If $k < 5$, let the missing indices be 0. So by identifying every quintuple of a Turing program, and encoding one of their corresponding sequences with the pairing encoding, a program now consists of some sequence $x_1, \ldots, x_n$ of numbers. Encode these numbers, and we have obtained a code number x for the Turing program.

This encoding was done in a bijective fashion—we can from any number x construct any of its $\mathrm{lh}(x)$ quintuples by obtaining $((x)_j)_i$ for $1 \leq j \leq 5$, and thus have a Turing program. From the primitive recursiveness of the pairing encoding it follows that this encoding is also primitive recursive.

Definition 1.3.4. The Turing program obtained from the code number e is denoted P_e, and the partial function computed by the Turing program P_e is denoted φ_e.

We will now create a similar primitive recursive encoding for the Turing computations. Recall that a computation according to P_e consists of configurations $c_1, \cdots, c_n$. Each configuration is on the form

$$s_{-m} \, \ldots \, s_{-1} \, q_i \, s_0 \, s_1 \, \ldots \, s_n.$$

and thus any sequence $a_1, \cdots, a_k$ can be identified with a configuration in the following way.

$$a_1, \cdots, a_k \sim \cdots s_{a_3}\, q_{a_1}\, s_{a_2} \cdots$$

where s_i is defined as above. We define the encoding of the computation to be the pairing encoding of the sequence $e, d_1, \cdots, d_n$, where d_i is the pairing encoding of a sequence corresponding to the configuration c_i.

Theorem 1.3.5 (Normal Form Theorem). *The predicate $T(e,x,y)$* (Kleene's T-predicate) *asserting that y is the code number of a computation according to Turing program P_e on input x is primitive recursive, and there exists a primitive recursive function $U(y)$ such that*

$$\varphi_e(x) = U(\mu y\ T(e,x,y))$$

for all e, x.

Proof. Let e,x,y be given. Obtain from e the Turing program P_e, and see if y is the code number of a computation. If not, then $T(e,x,y)$ does not hold. If it is, call this computation $c_1, \ldots, c_n$. Now check whether P_e given x yields exactly this computation. If it does, then $T(e,x,y)$ holds, and if not then $T(e,x,y)$ does not hold. Since the encodings are all primitive recursive, then so is T.

Define $U(y)$ as the number of 1's in the configuration recovered from $(y)_{\mathrm{ln}(y)}$—this is also primitive recursive. Thus $\varphi_e(x) = U(\mu y\ T(e,x,y))$. □

Remark. This can be extended to functions with multiple inputs, i.e. one can find predicates $T_n(e, x_1, \ldots, x_n, y)$ that are primitive recursive such that

$$\varphi_e^{(n)}(x_1, \ldots, x_n) = U(\mu y\, T_n(e, x_1, \ldots, x_n, y)).$$

It follows from the above Normal Form Theorem, that every partial function computable in the Turing machine sense is also a partial recursive function. By constructing Turing machines corresponding to each schema (i)-(vi) it can be shown that any p.r. function will also be Turing computable, see §68 in [3]. Hence these two classes of partial functions are the same.

The *Church-Turing Thesis* states that not only does our two definitions of computability give rise to the exact same class of functions, these are also exactly the functions which are computable *in the intuitive sense*, i.e., those which can be calculated with some algorithm. Given the informal nature of the thesis it is improvable, but it is widely accepted. All attempts to formalize the notion of computability has ended up with being equivalent to each other—another example being Church's λ-calculus. The thesis is discussed in pp. 317-332, 376-381 in [3], pp. 119-122 in [9] and in [11].

The Church-Turing Thesis is central to the motivation of recursion theory; it is what makes the subject concern the interesting epistemological notion that is computability, instead of concerning some meaningless formal system. Also, it is an invaluable tool for making our proofs easier to comprehend. In the following we will often rely on "loose" mathematical notation to define recursive functions, but in all cases this can be translated to one of the formal definitions.

1.4 Universal Turing machines and the S^m_n Theorem

In this section we will formulate some basic and important theorems about p.r. functions.

Lemma 1.4.1 (Padding Lemma). *Every p.r. function φ_e has infinitely many indices, and we can effectively find an infinite set containing such indices.*

Proof. Let P_e be any program, which we recall is a finite list of quintuples which tells the machine how to act. This will mention a finite number of states $\{q_0, \dots, q_n\}$. By adding the instruction (q_{n+1}, B, q_{n+1}, R) to P_e, we will not change the behaviour of the program, as there is no way for the machine to reach the state q_{n+1}. The coding of this Turing program however, will be higher than e. Repeat this for all numbers larger than n, and we will have an infinite set of indices. □

The following theorem postulates the existence of a *universal Turing machine.* Also introduced in 1936 in [12], this is a Turing machine which is able to simulate any other Turing machine, by taking its program as an input. Compare this to a modern computer operating system, which is a program that is able to execute any other program.

Theorem 1.4.2 (Enumeration Theorem). *Given $n \geq 1$, there exists a p.r. function $\varphi_{z_n}(e, x_1, \dots, x_n)$ of $n+1$ variables, such that*

$$\varphi_{z_n}(e, x_1, \dots, x_n) = \varphi_e(x_1, \dots, x_n)$$

for all e and $x_1, \dots, x_n$.

Proof. By Theorem 1.3.5 in its extended form, let

$$\varphi_{z_n}(e, x_1, \dots, x_n) = U(\mu y \, T(e, x_1, \dots, x_n, y)) = \varphi_e(x_1, \dots, x_n).$$

□

Our next theorem, the S^m_n Theorem, is in spirit the converse of the Enumeration Theorem. Visualize the p.r. functions ordered in a matrix where the n'th column contains all the n-ary functions. The Enumeration Theorem tells us that all the information in the n'th column is contained in a single cell in the $n+1$'th column, whereas the next theorem on the other hand will provide an effective way to pass from any cell in the $m+n$'th column to a range of cells in the m'th column.

Theorem 1.4.3 (Kleene's S^m_n Theorem)**.** *For every $m, n \geq 1$ there is an injective recursive function $S^m_n(e, y_1, \dots, y_m)$ of $m+1$ variables, such that for all $e, x_1, \dots, x_n, y_1, \dots, y_m$,*

$$\varphi_{S^m_n(e,y_1,\dots,y_m)}(x_1, \dots, x_n) = \varphi_e(y_1, \dots, y_m, x_1, \dots, x_n).$$

Proof. Call $\mathbf{x} = x_1, \dots, x_n$ and $\mathbf{y} = y_1, \dots, y_m$. Let the program $P_{S^m_n(e,\mathbf{y})}$ on input $\mathbf{x}$ first obtain P_e and then apply P_e to the input $(\mathbf{y}, \mathbf{x})$. This is an effective procedure in $\mathbf{x}$ and $\mathbf{y}$, so by the Church-Turing Thesis, S^m_n is a recursive function. If S^m_n is not injective, we can find an injective function S'^m_n such that $\varphi_{S^m_n(e,\mathbf{y})} = \varphi_{S'^m_n(e,\mathbf{y})}$ by using the Padding Lemma: Every time an index has been used before, just choose another index for the same program which is unused. □

A more formal proof can be found in §65 of [3]. This theorem proves useful in many of the basic computability results. Often it is used in the same way, namely as stated by the following corollary:

Corollary 1.4.4. *For every p.r. function $\psi(x, y)$ there is a recursive function f such that $\varphi_{f(x)}(y) = \psi(x, y)$.*

Proof. Find an index e of ψ. Let $f(x) = S^1_1(e, x)$ which exists and is recursive by the theorem. □

As an example of an application of the S^m_n Theorem, consider the following: We wish to find a recursive function f such that $\varphi_{f(x,y)} = \varphi_x \circ \varphi_y$. Therefore we define the p.r. function

$$\psi(x, y, z) = \begin{cases} \varphi_x(\varphi_y(z)) & \text{if } \varphi_y(z)\downarrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Fix an index e for ψ so $\psi = \varphi_e$, and use $S^2_1(e, x, y)$ to "hard code" the parameters x, y into e to obtain $f(x, y) = S^2_1(e, x, y)$.

1.5 Recursively Enumerable Sets

Definition 1.5.1. We call a set A *recursively enumerable* (since this name is rather unwieldy, we will refer to them as r.e. sets) if it is the domain of some p.r. function ψ, in symbols: $A = \{x : \psi(x)\downarrow\}$. We list the r.e. sets according to the numbering of p.r. functions, so we let the e'th r.e. set be the domain of φ_e, denoting it $W_e = \{x : \varphi_e(x)\downarrow\}$.

In the literature these are often known as *computably enumerable* (c.e.) sets, as their definition is independent on the choice of computability model, but we will use the original name. An r.e. set corresponds with a notion of semieffectiveness: Given a r.e. set W_e, one can check whether some number x lies in W_e by seeing if $\varphi_e(x)$ converges. But if it does not, we will have to wait forever to find out, and thus we can never get a negative answer.

Later we will show that if for some set A, both its complement and itself is r.e., then it must be recursive, as it can be effectively decided whether any element is a member of A.

Some r.e. sets are of special importance in the further study of the subject.

Definition 1.5.2. We define two important r.e. sets

$$K = \{x \ : \ \varphi_x(x)\downarrow\}$$
$$K_0 = \{\langle x, y\rangle \ : \ \varphi_y(x)\downarrow\}$$

Recall that $\langle\cdot,\cdot\rangle$ denotes Cantor's pairing function.

To see that K and K_0 are r.e., use φ_{z_1} from the Enumeration Theorem to obtain the p.r. function $\psi(x) = \varphi_{z_1}(x,x) = \varphi_x(x)$. Now $K = \{x \ : \ \psi(x)\downarrow\}$, and is thus r.e. Similarly, let $\psi_0(\langle x,y\rangle) = \varphi_{z_1}(y,x)$ so we can see that $K_0 = \{\langle x,y\rangle \ : \ \psi_0(\langle x,y\rangle)\downarrow\}$ is r.e.

The *halting problem* is to decide whether some program given an input will eventually terminate, or equivalently, whether some p.r. function φ_y is defined for input x. Therefore, this is equivalent to deciding whether $\langle x,y\rangle \in K_0$. K_0 is also called the *halting set*, and K the *diagonal halting set.*

Proposition 1.5.3. *Neither K nor K_0 is recursive, and thus the halting problem is undecidable.*

Proof. Firstly, we assume for contradiction that K is recursive, and thus χ_K is a total recursive function. But then we can define a new total recursive function f by

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{if } \chi_K(x) = 1, \\ 0 & \text{if } \chi_K(x) = 0. \end{cases}$$

Yet now, $f \neq \varphi_x$ for any x, and thus f cannot be recursive. Hence, K cannot be recursive.

Now assume that K_0 is recursive. Since $\langle x,x\rangle \in K_0$ iff $x \in K$, then K must also be recursive, which we already have established it is not. Hence, K_0 is not recursive either. □

1.6 Finite Approximations

As it is central to the concept of computability that a computation is done in a *finite* number of steps, we will introduce a way of writing how many steps are actually used to obtain a result. Here the Turing machine definition will be used, as this provides us directly with a method to count the amount of steps in a given computation.

Definition 1.6.1. We define $\varphi_{e,s}(x)$ inductively. Let $\varphi_{e,0}(x)$ be undefined for all x. Let $\varphi_{e,s+1}(x)\downarrow= y$ iff either $\varphi_{e,s}(x)\downarrow= y$, or $s = \langle e, x, y, t\rangle$ for some t and the Turing program P_e has halted with output y in less than t steps.

The above definition seems a bit overly complicated. Intuitively, s is just the amount of steps allowed to use in the calculation. A convenient consequence of this definition is, that for each s, there is at most one $\langle e, x\rangle$ such that $\varphi_{e,s}(x)\downarrow$ and $\varphi_{e,s-1}(x)\uparrow$.

Now $\varphi_{e,s}$ provides some notion of whether φ_e *converges*, because obviously

$$\varphi_e(x)\downarrow \iff \exists s\ \varphi_{e,s}(x)\downarrow .$$

Now, a natural decision problem to consider, is the problem of deciding whether $\varphi_{e,s}(x)$ is defined for some e, s, x. But since one only has to check for a finite number of steps, this must be computable. Hence we can formulate the following theorem, which will be useful in deciding the complexity of other sets.

Theorem 1.6.2. *The sets*

$$\{\langle e, x, s\rangle \ :\ \varphi_{e,s}(x)\downarrow\}$$

$$\{\langle e, x, y, s\rangle \ :\ \varphi_{e,s}(x) = y\}$$

are recursive.

Now we will define some subsets of the r.e. sets.

Definition 1.6.3. We define

$$W_{e,s} = \{x \ :\ \varphi_{e,s}(x)\downarrow\}.$$

Consider $W_{e,s}$. Any element $x \in W_{e,s}$ must then also fulfill $x \in W_{e,s+1}$, thus $W_{e,s} \subseteq W_{e,s+1}$. We can also easily see that $W_e = \bigcup_s W_{e,s}$. Therefore W_e is approximated by $\{W_{e,s}\}_{s\in\omega}$ – we call this an *enumeration* of W_e. So it is meaningful to think of an r.e. set as constructed in countably many steps. From the rather complicated Definition 1.6.1 it follows, that for any step s there is at most one W_e which receives a new element x, and specifically, at any step in the construction of W_e there is at most added one new element.

1.7 Kleene's Recursion Theorem

At this point we are able to formulate a very elegant result of recursion theory. It is the following theorem, known as *Kleene's Fixed Point Theorem* or the *Recursion Theorem*, the first version of which is stated very subtly in the last two lines of §2 in [2].

Theorem 1.7.1 (Recursion Theorem). *For every recursive function f, there is an n such that $\varphi_n = \varphi_{f(n)}$. We call this a* fixed point *of f. It follows that $W_n = W_{f(n)}$.*

Proof. The strategy for this proof is at first a bit odd. It looks like a diagonal argument, yet it yields the opposite result of what is expected from this. This is because of the strong properties of the S^m_n Theorem.

Use the S^m_n Theorem to define the recursive function $d(u)$ by

$$\varphi_{d(u)}(z) = \begin{cases} \varphi_{\varphi_u(u)}(z) & \text{if } \varphi_u(u)\downarrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We now have that d is total and injective. If we think about the matrix

$$\begin{array}{cccc} \varphi_{\varphi_1(1)} & \varphi_{\varphi_1(2)} & \varphi_{\varphi_1(3)} & \cdots \\ \varphi_{\varphi_2(1)} & \varphi_{\varphi_2(2)} & \varphi_{\varphi_2(3)} & \\ \varphi_{\varphi_3(1)} & \varphi_{\varphi_3(2)} & \varphi_{\varphi_3(3)} & \\ \vdots & & & \ddots \end{array}$$

where $\varphi_{\varphi_x(y)}$ is the totally undefined function if $\varphi_x(y)\uparrow$, then $\varphi_{d(u)}$ is exactly the diagonal of this matrix. But the S^m_n Theorem also ensures that $d(u)$ is recursive so it has an index e, and therefore the row $\varphi_{\varphi_e(x)}$ is identical to the diagonal.

Now let some recursive function f be given. We can find an index v such that

$$\varphi_v = f \circ d.$$

In some sense, we can think of this as mapping the row $\varphi_{d(u)}$ to the row $\varphi_{f\circ d(u)}$. But using the diagonal properties of d, they must overlap at the v'th element, namely:

$$\varphi_{d(v)} = \varphi_{\varphi_v(v)} = \varphi_{f(d(v))}.$$

□

Corollary 1.7.2. *For every p.r. function $\psi(x,y)$ there is an index e such that $\varphi_e(y) = \psi(e,y)$.*

Proof. Using the S^m_n Theorem, find a recursive function f such that $\varphi_{f(x)}(y) = \psi(x,y)$. Now take a fixed point e of this f, so

$$\varphi_e(y) = \varphi_{f(e)}(y) = \psi(e,y).$$

□

Using this knowledge, we can now prove another elegant result first shown by H.G. Rice in [8].

Theorem 1.7.3 (Rice's Theorem). *For any class $\mathcal{C}$ of p.r. function which is non-trivial, i.e. neither empty nor containing all p.r. functions, the set $\{x : \varphi_x \in \mathcal{C}\}$ of indices is non-recursive.*

Proof. Assume that $\mathcal{C}$ is a non-trivial class of p.r. functions, and its set of indices is recursive. Fix two p.r. functions ψ_1, ψ_2 with $\psi_1 \in \mathcal{C}$ and $\psi_2 \notin \mathcal{C}$. The following function is then recursive

$$f(x,y) = \begin{cases} \psi_2(y) & \text{if } \varphi_x \in \mathcal{C}, \\ \psi_1(y) & \text{otherwise.} \end{cases}$$

By Corollary 1.7.2 we can find an e such that $\varphi_e(y) = f(e,y)$. We now ask: Is $\varphi_e \in \mathcal{C}$? If it is, then $\varphi_e(y) = f(e,y) = \psi_2(y)$, but $\psi_2 \notin \mathcal{C}$. It it is not, then $\varphi_e(y) = f(e,y) = \psi_1(y)$, but $\psi_1 \in \mathcal{C}$. Thus both answers leads to contradiction, and hence the index set cannot be recursive. □

It follows from Rice's Theorem, that there is no effective way to decide if some algorithm has a certain property.

1.8 A Normal Form for r.e. Sets

In the following, we will introduce a method which can be used to easily state whether a set is r.e. or not. To do this, we must introduce the notion of a Σ_1-set. Later we will see this concept generalized.

Definition 1.8.1. A set A is Σ_1 if

$$x \in A \iff \exists y\ R(x,y)$$

where R is some recursive relation.

It does not matter if there are more than one $\exists$-quantifier in the definition. If a set A is defined by

$$x \in A \iff \exists y_1\, \exists y_2 \dots \exists y_n\ R(x, y_1, \dots, y_n)$$

where $R(x, y_1, \dots, y_n)$ is a recursive relation, we can encode $y_1, \dots, y_n$ into a single integer y. Define the recursive relation $R'(x,y)$ by

$$R'(x,y) \iff R(x, (y)_1, \dots, (y)_n),$$

for then

$$x \in A \iff \exists y\ R'(x,y),$$

and so $A \in \Sigma_1$.

Theorem 1.8.2 (Normal Form Theorem for r.e. sets). *A set A is r.e. iff A is Σ_1.*

Proof. If A is r.e., then it is the domain of some p.r. function φ_e. Recall from the above Normal Form Theorem the primitive recursive predicate T. With this, we have that

$$x \in A \iff \exists y\ T(e, x, y),$$

and thus A must be Σ_1.

If on the other hand we assume that $A = \{x : \exists y\ R(x, y)\}$ where R is recursive, then define the function $\psi(x) = \mu y\ R(x, y)$ which must be p.r. Now A is the domain of ψ, and thus it is r.e. □

We can demonstrate this characterization with K and K_0. Recall the definition of K,

$$K = \{x : \varphi_x(x) \downarrow\} = \{x : \exists s\ [\varphi_{x,s}(x) \downarrow]\}$$

and since by Theorem 1.6.2 the relation $R(x, s) = \{(x, s) : \varphi_{x,s}(x) \downarrow\}$ is recursive, K must be Σ_1 and hence r.e. Likewise we can see that K_0 is r.e. from

$$K_0 = \{\langle x, y\rangle : \varphi_y(x) \downarrow\} = \{\langle x, y\rangle : \exists s\ [\varphi_{y,s}(x) \downarrow]\}.$$

We will now examine an interesting relation between graphs and partial recursiveness. We recall the standard notion of a graph.

Definition 1.8.3. The *graph* of a partial function ψ is the relation

$$(x, y) \in \operatorname{graph} \psi \iff \psi(x) = y.$$

Theorem 1.8.4 (Graph Theorem). *A partial function ψ is p.r. iff its graph is r.e.*

Proof. Assume first that $\psi = \varphi_e$ is p.r. Now the graph is given by

$$\operatorname{graph} \varphi_e = \{(x, y) : \exists s\ [\varphi_{e,s}(x) = y]\},$$

which is r.e. by the Normal Form Theorem 1.8.2 and Theorem 1.6.2.

Now assume that for some ψ, $\operatorname{graph} \psi$ is r.e. Thus there is a recursive relation R so

$$(x, y) \in \operatorname{graph} \psi \iff \exists z\ R(x, y, z).$$

Since R is recursive, we can define the p.r. function $\theta(x)$ as follows: Given x, find the smallest pair $\langle y, z\rangle$ such that $(x, y, z) \in R$, and output y. Now $\theta(x)$ is defined iff $(x, \theta(x)) \in \operatorname{graph} \psi$, and so by Definition 1.8.3 $\theta = \psi$. □

A good way to think of an r.e. set, is to think of it as one whose members can be effectively listed (*recursively enumerated*), as $A = \{a_0, a_1, \dots\}$ with some algorithm. This, though, is not clear from the definition, but it follows from the following basic result which explains the name *r.e.*

Theorem 1.8.5 (Listing Theorem)**.** *A set A is r.e. iff it is the range of some total recursive function f, or if it is empty.*

Proof. Let A be a non-empty r.e. set, thus $A = W_e$ for some e. We will create a recursive functions f such that A is exactly the range of f. First we will fix the smallest integer $\langle s, a\rangle$ such that $a \in W_{e,s}$. This a will be the 'default' element in our listing. Then we define f by

$$f(\langle s, x\rangle) = \begin{cases} x & \text{if } x \in W_{e,s+1} \text{ but } x \notin W_{e,s}, \\ a & \text{otherwise.} \end{cases}$$

Now for any element $x \in A$, there will be some s such that $f(\langle s, x\rangle) = x$, and every output of f will be an element of A, thus A is the range of f. So $f(0), f(1), \dots$ is a listing of all elements of A, where any element except the default a occurs only once.

Conversely, let A be the range of the recursive function φ_e. Then A is exactly the set

$$\{y \ : \ \exists x \ [\varphi_e(x) = y]\} = \{y \ : \ \exists s \ \exists x \ [\varphi_{e,s}(x) = y]\},$$

which is r.e. If $A = \varnothing$ then it is trivially r.e. □

The following theorem first published by Post in [7] binds the two concepts of r.e. sets and recursive sets together, in the way intuitively explained above.

Theorem 1.8.6 (Complementation Theorem)**.** *A set is recursive iff both the set itself and its complement are r.e.*

Proof. If A is recursive, then also is $\overline{A}$, since we can define its characteristic function just by flipping the values of χ_A. A recursive set is also r.e., A is for instance the domain of

$$\psi(x) = \begin{cases} 1 & \text{if } \chi_A(x) = 1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Thus A and $\overline{A}$ are r.e.

If we assume A and $\overline{A}$ are r.e., then there are indices e, i so $A = W_e$ and $\overline{A} = W_i$. Now define

$$f(x) = \mu s \ [x \in W_{e,s} \text{ or } x \in W_{i,s}],$$

which is total recursive. Now $x \in A$ iff $x \in W_{e,f(x)}$, so A is recursive. □

Chapter 2

Relative Computability

In order to study more complex concepts of recursion theory, we will need to expand our idea of computability. Introduced in 1939 by Turing in [13] is the concept of an *oracle machine*, which in essence is a Turing machine with the ability to consult some 'oracle' which has all the answers to some number theoretic problem. Later, an analogous technique is introduced for recursive functions as seen in page 275 of [3].

2.1 Relative Turing Computability

We will relativize the above definition of a Turing machine by defining the oracle machine as a normal Turing machine, but with an extra tape which is 'read-only'. Call this the *oracle tape*, and call the old tape the *work tape*. The alphabet for the oracle tape is $S_o = \{B, 0, 1\}$, and the contents of the tape is to be the characteristic function of some set A, with the starting position denoting $\chi_A(0)$, thus all cells left of this contains only B, and all cells right of this contains either 0 or 1. The reading head will move in the same direction on both tapes simultaneously. Recalling how we defined a Turing program, we now use the same symbols to define an *oracle Turing program* to be the partial function

$$\delta : Q \times S \times S_o \to Q \times S \times \{R, L\},$$

where $\delta(q, a, b) = (p, c, X)$ indicates that the machine in state q reading the symbol a on the work tape and the symbol b on the oracle tape, passes to the state p, print the symbol c over a on the work tape, and moves one step to X on both tapes. We define the input and output in the same way as above for Turing machines. A thing that will be interesting is, how many cells of the oracle tape which are scanned. If y is the number of non-blank cells scanned on the oracle tape, then the maximum integer which is tested for membership of A in the program is $y-1$. We say that all $z < y$ are *used* in the computation. It is clear that this new definition is stronger than the previous, as every Turing program can be written as an oracle Turing program, by defining δ as a partial function of Q and S, and not S_o.

Definition 2.1.1. We say that a partial function ψ of n variables is *Turing computable in A* if there is some oracle Turing program P such that on a machine with χ_A written on the oracle tape, then for all $x_1, \ldots, x_n, y$, P on input $x_1, \ldots, x_n$ halts with output y iff $\psi(x_1, \ldots, x_n) \downarrow = y$.

In the same way as for the usual Turing programs, we are able to make an effective primitive recursive encoding of the oracle Turing programs. We again call the e'th program P_e.

2.2 Relative Recursiveness

We will now define the analogous concept for recursive functions.

Definition 2.2.1. We say that a partial function ψ is *partial recursive in A* if it can be derived from the characteristic function of A and the schemata (i)-(vi) from Definitions 1.2.1 and 1.2.2.

Once again, these definitions give rise to the same functions, so we can formulate the following theorem, proved in §68 and §69 of [3].

Theorem 2.2.2. *The partial function ψ is Turing computable in A iff ψ is partial recursive in A.*

Also, we will accept the Church-Turing Thesis with respect to relative computability, which say that the functions Turing computable in A are exactly the functions computable in A in the intuitive sense. Thus we still allow ourselves to use the most easily comprehended notation when defining relative recursive functions, as long as it is clear that it can be translated to one of the formal definitions.

When writing these relative computable functions we will introduce a new notation.

Definition 2.2.3. If ψ is computable in A by some program P_e, we write $\psi \leq_T A$, and we call this function $\psi = [\![e]\!]^A$. If $A = \varnothing$ we write $[\![e]\!]$ instead of $[\![e]\!]^\varnothing$ (note that then $[\![e]\!] = \varphi_e$).

The reason for introducing a new notation and not just use φ_e^A is partly to ease reading by having fewer subscripts, but is also to put emphasis on the code number of a function, as this is a complete description of the function. We will speak of sets and relations as being A-recursive, or recursive in A, if their characteristic functions are recursive in A.

2.3 Use

As mentioned, it will be important to have a notion of how large numbers are used in a computation. Compare the following definition with Definition 1.6.1.

Definition 2.3.1. We write $[\![e]\!]^A_s(x) = y$ if $x, y, e < s$ and $[\![e]\!]^A(x) = y$ in less than s steps according to P_e, and if only numbers smaller than s are *used* in the computation.

We denote *binary strings* of 0's and 1's with the greek letters $\sigma, \tau, \dots$, and use them to denote an initial segment of a characteristic function.

Definition 2.3.2. A *binary string* σ of length n is a function $\sigma : \{0, \dots n\} \to \{0, 1\}$. We denote the length of a string with $\mathrm{lh}(\sigma)$. We will write $\sigma \subset A$ if $\sigma(x) = \chi_A(x)$ for all $x \leq \mathrm{lh}(\sigma)$. Define the *restriction of A to x*, written $A \upharpoonright x$, as the string of length x with $A \upharpoonright x \subset A$. Similarly define $\sigma \upharpoonright x$ as σ restricted to arguments less than x.

Furthermore we define string *concatenation* as follows. If α is a binary string of length n_1 and β is a binary string of length n_2, then $\alpha\hat{}\beta$ is a binary string of length $n_1 + n_2$ with

$$(\alpha\hat{}\beta)(i) = \begin{cases} \alpha(i) & \text{if } i \leq n_1, \\ \beta(i - n_1) & \text{if } n_1 < i \leq n_1 + n_2. \end{cases}$$

We can easily find an effective coding of these strings, so when referring to a string, we can handle it like a number, which will be practical in some cases. We can use these strings to introduce another notion of finite approximation.

Definition 2.3.3. Let σ be a string. We write $[\![e]\!]^\sigma_s(x) = y$ if σ is the initial segment of some set A with $[\![e]\!]^A_s(x) = y$ with only numbers smaller than $\mathrm{lh}(\sigma)$ being used in the computation. We write $[\![e]\!]^\sigma(x) = y$ if $\exists s\ [[\![e]\!]^\sigma_s(x) = y]$.

Theorem 2.3.4 (Master Enumeration Theorem)**.**

(i) $\{\langle e, \sigma, x, s\rangle : [\![e]\!]^\sigma_s(x)\downarrow\}$ *is recursive,*

(ii) $\{\langle e, \sigma, x\rangle : [\![e]\!]^\sigma(x)\downarrow\}$ *is r.e.*

Proof. (i) To see whether $[\![e]\!]^\sigma_s(x)$ converges for given e, σ, x, s, one must check whether the program P_e with σ as oracle halts with x as input in less than s steps. As this is a finite procedure, it must be recursive.

(ii) $\{\langle e, \sigma, x\rangle : [\![e]\!]^\sigma(x)\downarrow\} = \{\langle e, \sigma, x\rangle : \exists s\ [[\![e]\!]^\sigma_s(x)\downarrow]\}$, so by (i) it is Σ_1 and hence r.e.

□

Definition 2.3.5. We define the *use function* to be $u(A;e,x,s) = 1+$ the maximum number used in the computation of $[\![e]\!]_s^A(x)$ if $[\![e]\!]_s^A(x)\downarrow$, and $u(A;e,x,s) = 0$ otherwise. We say that $u(A;e,x) = u(A;e,x,s)$ if there is an s such that $[\![e]\!]_s^A(x)\downarrow$, and write $u(A;e,x)\uparrow$ otherwise.

Theorem 2.3.6 (Use Principle)**.**

(i) $[\![e]\!]^A(x) = y \implies \exists s\ \exists\sigma\subset A\ [[\![e]\!]_s^\sigma(x) = y]$

(ii) $[\![e]\!]_s^\sigma(x) = y \implies \forall t\geq s\ \forall\tau\supseteq\sigma\ [[\![e]\!]_t^\tau(x) = y]$

(iii) $[\![e]\!]^\sigma(x) = y \implies \forall A\supset\sigma\ [[\![e]\!]^A(x) = y]$.

The proof of this principle follows largely from the Definitions 2.2.3 and 2.3.1 and the fact that any computation that halts must do so after a finite amount of steps. The Use Principle is important because it allows us to make for example the following steps: If for some A,s,x,y we have that $[\![e]\!]_s^{A\restriction y}(x)\downarrow$, then by (iii)

$$[\![e]\!]^A(x) = [\![e]\!]^{A\restriction y}(x) = [\![e]\!]_s^{A\restriction y}(x).$$

Or, if we have the use $u = u(A;e,x)$ of some computation, then

$$[\![e]\!]^{A\restriction u}(x) = [\![e]\!]^A(x).$$

Definition 2.3.7. A set A is *r.e. in* B if A is the domain of $[\![e]\!]^B$ for some e. A set A is Σ_1^B if $x\in A \iff \exists y\ R(x,y)$ where R is a B-recursive relation.

The theorems concerning p.r. functions and r.e. sets can be relativized to work with these new notions. As an example, here the S_n^m Theorem and the Normal Form Theorem for r.e. sets:

Theorem 2.3.8 (Relativized S_n^m Theorem)**.** *For every $m,n\geq 1$ there is an injective recursive function $S_n^m(e,y_1,\dots,y_m)$ such that for all $e,y_1,\dots,y_m$, $x_1\dots,x_n$ and all sets A*

$$[\![S_n^m(e,y_1,\dots,y_m)]\!]^A = [\![e]\!]^A(y_1,\dots,y_m,x_1,\dots,x_n).$$

Theorem 2.3.9 (Relativized Normal Form Theorem for r.e. sets)**.** *A set A is r.e. in B iff $A\in\Sigma_1^A$.*

The proofs of these relativized theorems can be obtained from their original proofs by replacing *recursive* with *A-recursive*, etc.

2.4 Turing Degrees and the Jump Operator

The intention behind the notation $\psi\leq_T A$ is to have $\leq_T$ as a notion of relative complexity between sets.

Definition 2.4.1. For sets A, B we write $B \leq_T A$ if $\chi_B \leq_T A$. We say that the two sets are *Turing equivalent* if $B \leq_T A$ and $A \leq_T B$, and write $A \equiv_T B$.

Theorem 2.4.2. *The relation $\leq_T$ is reflexive and transitive, so $\equiv_T$ is an equivalence relation.*

Proof. Any set A can trivially be computed by a machine with A as an oracle, so $A \leq_T A$, and $\leq_T$ is reflexive. If A can be computed from B and B can be computed from C, then A can be computed from C by first computing B and then computing A—thus $\leq_T$ is transitive. Therefore, the relation $\equiv_T$ is also reflexive and transitive and it is trivially symmetric, hence it is an equivalence relation. □

If $A \equiv_T B$ then we say that A and B *codes the same information.*

Theorem 2.4.3. $K \equiv_T K_0$.

Proof. $K \leq_T K_0$: Obvious, since $x \in K \iff \varphi_x(x)\downarrow \iff \langle x, x\rangle \in K_0$.

$K_0 \leq_T K$: Since K_0 is r.e., there is an e such that $K_0 = W_e$. Now define the p.r. function $\psi(x, y) = \varphi_e(x)$, and obtain the total recursive function g from the S^m_n Theorem which fulfills

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

Then

$$x \in K_0 \iff \varphi_e(x)\downarrow \iff \varphi_{g(x)}(g(x))\downarrow \iff g(x) \in K.$$

□

Actually, the second part of the above proof shows that K is *r.e.-complete*, by which we mean that $W \leq_T K$ for any r.e. set W, and thus these sets are the largest r.e. sets with respect to $\leq_T$.

Definition 2.4.4. The *Turing degree* of a set A is the equivalence class

$$\deg(A) = \{B \ : \ B \equiv_T A\}.$$

Turing degrees will be written as lowercase boldface letters $\mathbf{a}, \mathbf{b}, \mathbf{c}$. If $A \leq_T B$ for some $A \in \mathbf{a}$ and $B \in \mathbf{b}$, then we say $\mathbf{a} \leq \mathbf{b}$. We say $\mathbf{a} < \mathbf{b}$ if $\mathbf{a} \leq \mathbf{b}$ and $\mathbf{a} \neq \mathbf{b}$.

A degree $\mathbf{a}$ is r.e. if it contains an r.e. set, and it is r.e. in $\mathbf{b}$ if it contains some set A which is r.e. in some set $B \in \mathbf{b}$.

Intuitively, $\mathbf{a} < \mathbf{b}$ means that membership of sets of degree $\mathbf{b}$ is harder to compute than membership of sets of degree $\mathbf{a}$.

Since there for any recursive function f must be an e such that $f = [\![e]\!] = [\![e]\!]^{\varnothing}$, then all recursive sets are of degree $\mathbf{0} =_{\text{def}} \deg(\varnothing)$—this is the smallest Turing degree.

The Turing degree $\deg(K)$ is r.e., and since no r.e. set codes more information than K, it is the largest r.e. degree. By re-using the method used to define K as a non-recursive set, we will define the relativized version: The *jump set.*

Definition 2.4.5. The *jump* of a set A is defined as

$$A' = K^A = \{x \ : \ [\![x]\!]^A(x)\downarrow\},$$

and the n'th jump of A is obtained by iterating the jump n times, i.e., $A^{(0)} = A, A^{(n+1)} = (A^{(n)})'$.

This jump operator has some crucial properties, which are summarized in the following theorem.

Theorem 2.4.6 (The Jump Theorem). *(i) A' is r.e. in A.*

(ii) $A' \not\leq_T A$.

(iii) If A is r.e. in B and $B \leq_T C$ then A is r.e. in C.

(iv) If $A \equiv_T B$ then $A' \equiv_T B'$.

(v) A is r.e. in B iff A is r.e. in $\overline{B}$.

The proof can be found in p. 53 of [10]. By (iv), we can speak of jumps on degrees, and we will write $\mathbf{a}' = \deg(A')$ for $A \in \mathbf{a}$.

We define an infinite hierarchy of degrees,

$$\mathbf{0} < \mathbf{0}' < \mathbf{0}'' < \cdots < \mathbf{0}^{(n)} < \dots,$$

by $\mathbf{0}^{(n)} = \deg(\varnothing^{(n)})$. We call it the *Turing hierarchy*.

In p. 314 of [7] Post raises the question of whether there are any unsolvable (i.e., greater than $\mathbf{0}$) r.e. degrees strictly below $\mathbf{0}'$, a question he himself were unable to answer. This question is known as *Post's problem*. In [4] Kleene and Post constructs a pair of incomparable degrees below $\mathbf{0}'$, but they cannot prove that they are r.e. In Chapter 3 we will solve Post's problem with two different proofs.

2.5 The Arithmetical Hierarchy

In the last section we defined a hierarchy of degrees based on Turing reducibility and the jump operator. We will now define a hierarchy of sets, based on the quantifier complexity of the syntactic definition of the sets. It is a generalization of the earlier definition of Σ_1 sets.

Definition 2.5.1. We define the *arithmetical hierarchy* to be the classes $\Sigma_n, \Pi_n, \Delta_n$ for all n given by the following.

- The set A is in Σ_0 and Π_0 iff A is recursive.

- For $n \geq 1$, $A \in \Sigma_n$ iff there is a recursive relation $R(x, y_1, \ldots, y_n)$ such that
$$x \in A \iff \exists y_1\ \forall y_2\ \exists y_3 \ldots Q y_n\ R(x, y_1, \ldots, y_n),$$
where Q represents $\forall$ if n is even, and $\exists$ otherwise.

- For $n \geq 1$, $A \in \Pi_n$ iff there is a recursive relation $R(x, y_1, \ldots, y_n)$ such that
$$x \in A \iff \forall y_1\ \exists y_2\ \forall y_3 \ldots Q y_n\ R(x, y_1, \ldots, y_n),$$
where Q represents $\exists$ if n is even, and $\forall$ otherwise.

- The set A is in Δ_n iff $A \in \Sigma_n \cap \Pi_n$.

- We call a set A *arithmetical* if $A \in \bigcup_n (\Sigma_n \cup \Pi_n)$.

If we are given a definition of a set including more than one adjacent quantifiers of the same kind, they can be collapsed into a single quantifier, in the same way as remarked after Definition 1.8.1—which is only a special case of this definition.

We use the following abbreviations

$$\forall xRy\ \Phi =_{\text{def}} \forall x\ [xRy \implies \Phi]$$
$$\exists xRy\ \Phi =_{\text{def}} \exists x\ [xRy \wedge \Phi],$$

where R is one of $<, \leq, >, \geq$. If R is $<$ or $\leq$, we call the quantifier a *bounded quantifier*, and as we will see, these can be ignored when determining the quantifier complexity of a set. By applying the usual rules of quantifier manipulation, we can prove some facts about arithmetical sets, which are useful when determining if a concrete set is Σ_n or Π_n.

Theorem 2.5.2. *(i)* $A \in \Sigma_n \iff \overline{A} \in \Pi_n$,

(ii) $A \in \Sigma_n \cup \Pi_n \implies \forall m > n\ [A \in \Delta_m]$,

(iii) $A, B \in \Sigma_n \implies A \cup B, A \cap B \in \Sigma_n$,

(iv) $A, B \in \Pi_n \implies A \cup B, A \cap B \in \Pi_n$,

(v) If $R \in \Sigma_n$*, then the sets* $A, B \in \Sigma_n$ *where*
$$\langle x, y\rangle \in A \iff \forall z < y\ R(x, y, z)$$
and
$$\langle x, y\rangle \in B \iff \exists z < y\ R(x, y, z).$$
If instead $R \in \Pi_n$*, then* $A, B \in \Pi_n$ *would hold.*

Proof. (i) If

$$x \in A \iff \exists y_1\ \forall y_2 \dots Qy_n\ R(x, y_1, \dots, y_n)$$

then

$$x \in \overline{A} \iff x \notin A \iff \forall y_1\ \exists y_2 \dots \overline{Q}y_n\ \neg R(x, y_1, \dots, y_n),$$

where $\overline{Q}$ denotes $\forall$ if Q is $\exists$, and $\exists$ otherwise. Since the negation of a recursive relation is clearly also recursive, then $\overline{A} \in \Pi_n$.

(ii) By induction. Consider the case where $A \in \Sigma_n$, for n even, so

$$x \in A \iff \exists y_1\ \forall y_2 \dots \forall y_n\ R(x, y_1, \dots, y_n).$$

then

$$x \in A \iff \exists y_1\ \forall y_2 \dots \forall y_n\ \exists y_{n+1}\ [R(x, y_1, \dots, y_n) \wedge y_{n+1} = y_{n+1}]$$

and

$$x \in A \iff \forall y_1\ \exists y_2 \dots \exists y_n\ \forall y_{n+1}\ [y_1 = y_1 \wedge R(x, y_2, \dots, y_{n+1})],$$

so $A \in \Sigma_{n+1} \cap \Pi_{n+1}$.

(iii) Let $A, B \in \Sigma_n$, so

$$\begin{aligned} x \in A &\iff \exists y_1\ \forall y_2 \dots Qy_n\ R(x, y_1, \dots, y_n) \\ x \in B &\iff \exists z_1\ \forall z_2 \dots Qz_n\ S(x, z_1, \dots, z_n) \end{aligned}$$

and thus

$$\begin{aligned} x \in A \cup B &\iff [\exists y_1\ \forall y_2 \dots Qy_n\ R(x, y_1, \dots, y_n)] \\ &\qquad \vee [\exists z_1\ \forall z_2 \dots Qz_n\ S(x, z_1, \dots, z_n)] \\ &\iff \exists y_1\ \exists z_1\ \forall y_2 \forall z_2 \dots Qy_n\ Qz_n\ [R(x, y_1, \dots, y_n) \\ &\qquad\qquad \vee S(x, z_1, \dots, z_n)] \\ &\iff \exists u_1\ \forall u_2 \dots Qu_n\ [R(x, (u_1)_1, \dots, (u_n)_1) \\ &\qquad\qquad \vee S(x, (u_1)_2, \dots, (u_n)_2)], \end{aligned}$$

where u_i is an encoding of y_i, z_i, so $A \cup B \in \Sigma_n$. It is shown similarly for $A \cap B$.

(iv) The same as the proof of (iii) but with all quantifiers flipped.

(v) If $R \in \Sigma_n$, then $B \in \Sigma_n$, since the $\exists$-quantifier can be collapsed with the first $\exists$ in the definition of R. We will prove $A \in \Sigma_n$ by induction over n.

If $n = 0$ then A is clearly recursive. Fix $n > 0$ and assume it holds for $m < n$. Now there exists $S \in \Pi_{n-1}$ such that

$$\begin{aligned} \langle x, y\rangle \in A &\iff \forall z < y\ R(x,y,z) \\ &\iff \forall z < y\ \exists u\ S(x,y,z,u). \end{aligned}$$

If $\langle x, y\rangle \in A$, then let for each $z < y$, u_z denote the number which fulfills $S(x,y,z,u_z)$, and encode $u_0, \ldots, u_z$ into a single integer v. Thus

$$\forall z < y\ \exists u\ S(x,y,z,u) \iff \exists v\ \forall z < y\ S(x,y,z,(v)_z),$$

and by the induction hypothesis the relation $\forall z < y\ S$ is Π_{n-1}, so A must be Σ_n. The case where $R \in \Pi_n$ follows from (i).

□

We now have two different notions of classifications of sets, the Turing degrees, and the arithmetical sets. The following theorem establishes the connection between these two.

Theorem 2.5.3 (Post's Theorem). *For every $n \geq 0$*

(i) $A \in \Sigma_{n+1} \iff A$ *is r.e. in some* Π_n *set* $\iff A$ *is r.e. in some* Σ_n *set,*

(ii) $A \in \Sigma_n \implies A \leq_T \varnothing^{(n)}$,

(iii) $A \in \Sigma_{n+1} \iff A$ *is r.e. in* $\varnothing^{(n)}$,

(iv) $A \in \Delta_{n+1} \iff A \leq_T \varnothing^{(n)}$,

(v) $\varnothing^{(n)} \in \Sigma_n \setminus \Pi_n$, *for* $n > 0$.

Proof. (i) Let $A \in \Sigma_{n+1}$. Then $x \in A \iff \exists y\ R(x,y)$ for some $R \in \Pi_n$, so $A \in \Sigma_1^R$ and hence by the relativized Normal Form Theorem for r.e. sets, A is r.e. in R.

Next, let A be r.e. in some Π_n set B. Thus there is an e such that

$$\begin{aligned} x \in A &\iff [\![e]\!]^B(x)\downarrow \\ &\iff \exists s\ \exists \sigma\ [\sigma \subset B \wedge [\![e]\!]^\sigma_s(x)\downarrow]. \end{aligned}$$

From the Master Enumeration Theorem it follows, that $[\![e]\!]^\sigma_s\downarrow$ is recursive, so if we can show that $\sigma \subset B$ is Σ_{n+1}, then by quantifier contraction B is also Σ_{n+1}. But

$$\begin{aligned} \sigma \subset B &\iff \forall y < \mathrm{lh}(\sigma)\ [\sigma(y) = \chi_B(y)] \\ &\iff \forall y < \mathrm{lh}(\sigma)\ [[\sigma(y) = 1 \wedge y \in B] \vee [\sigma(y) = 0 \wedge y \notin B]], \end{aligned}$$

and the first part of this disjunction is Π_n, the second is Σ_n because $B \in \Pi_n$, thus $\sigma \subset B$ must be Σ_{n+1}. This proves the first bi-implication—the second follows from (v) in the Jump Theorem.

(ii) By induction on n. For $n = 0$ it is clear. Now fix $n > 0$, and assume it to hold for n. Let $A \in \Sigma_{n+1}$. Then by (i), A is r.e. in some $B \in \Sigma_n$. By the induction hypothesis $B \leq_T \varnothing^{(n)}$, so B is r.e. in $\varnothing^{(n)}$ by (iii) of the Jump Theorem. Thus there is an e such that $x \in A \iff [\![e]\!]^{\varnothing^{(n)}}(x)\downarrow$. Define the $\varnothing^{(n)}$-recursive function $\psi(x,y) = [\![e]\!]^{\varnothing^{(n)}}(x)$, and obtain with the relativized S^m_n Theorem an injective and recursive g such that

$$[\![g(x)]\!]^{\varnothing^{(n)}}(y) = \psi(x,y),$$

for then

$$\begin{aligned} x \in A &\iff [\![e]\!]^{\varnothing^{(n)}}(x)\downarrow \\ &\iff [\![g(x)]\!]^{\varnothing^{(n)}}(g(x))\downarrow \\ &\iff g(x) \in K^{\varnothing^{(n)}} = \varnothing^{(n+1)} \end{aligned}$$

and hence the characteristic function of A can be computed with an $\varnothing^{(n+1)}$ oracle machine. Notice the similarity to the proof of Theorem 2.4.3.

(iii) Follows from (iii) of Jump Theorem and (ii).

(iv)

$$\begin{aligned} A \in \Delta_{n+1} &\iff A, \overline{A} \in \Sigma_{n+1} \\ &\iff A, \overline{A} \text{ are r.e. in } \varnothing^{(n)} \text{ (by (iii))} \\ &\iff A \leq_T \varnothing^{(n)}, \end{aligned}$$

by the relativized version of the Complementation Theorem.

(v) We show $\varnothing^{(n)} \in \Sigma_n$ by induction. If $n = 1$ then $\varnothing' = K \in \Sigma_1$. Assume $\varnothing^{(n)} \in \Sigma_n$ for $n \geq 1$. By (i) of the Jump Theorem, we know that $\varnothing^{(n+1)}$ is r.e. in $\varnothing^{(n)}$, so by (i) $\varnothing^{(n+1)} \in \Sigma_{n+1}$. Hence $\varnothing^{(n)} \in \Sigma_n$ for all $n \geq 1$. Assume that for some n, $\varnothing^{(n)} \in \Pi_n$, thus $\varnothing^{(n)} \in \Delta_n$. By (iv) we get the contradiction $\varnothing^{(n)} \leq_T \varnothing^{(n-1)}$, and therefore $\varnothing^{(n)} \in \Sigma_n \setminus \Pi_n$.

□

From this theorem it follows that the hierarchy is non-trivial—it does not collapse, as stated by the following corollary which follows directly from (v):

Corollary 2.5.4 (Hierarchy Theorem). *For any $n > 0$, $\Delta_n \subset \Sigma_n$ and $\Delta_n \subset \Pi_n$.*

We will consider some examples of naturally occurring sets, and try to place them in our hierarchies.

Theorem 2.5.5. *(i)* $\mathrm{Fin} = \{x : W_x \textit{ is finite}\} \in \Sigma_2$, *(ii)* $\mathrm{Tot} = \{x : \varphi_x$ *is total*$\} \in \Pi_2$,

(iii) $\mathrm{Rec} = \{x : W_x$ *is recursive*$\} \in \Sigma_3$.

Proof. (i)

$$W_x \text{ is finite} \iff \exists s\, \forall t\, [t \le s \vee W_{x,s} = W_{x,t}].$$

The relation in the brackets is recursive, so $\mathrm{Fin} \in \Sigma_2$.

(ii)

$$\varphi_x \text{ is total} \iff W_x = \omega \iff \forall y\, \exists s\, [y \in W_{x,s}].$$

(iii)

$$\begin{aligned} W_x \text{ is recursive} &\iff \exists y\, [W_x = \overline{W_y}] \\ &\iff \exists y\, [W_x \cap W_y = \varnothing \wedge W_x \cup W_y = \omega] \\ &\iff \exists y\, [\forall z\, \forall s\, (z \notin W_{x,s} \wedge z \notin W_{y,s}) \\ &\qquad \wedge \forall q\, \exists t\, (q \in W_{x,t} \vee q \in W_{y,t})] \\ &\iff \exists y\, \forall z\, \forall s\, \forall q\, \exists t\, [(z \notin W_{x,s} \\ &\qquad \wedge z \notin W_{y,s}) \wedge (q \in W_{x,t} \vee q \in W_{y,t})]. \end{aligned}$$

□

These classifications are actually as sharp as possible, and

$$\deg(\mathrm{Fin}) = \deg(\mathrm{Tot}) = \mathbf{0}''$$
$$\deg(\mathrm{Rec}) = \mathbf{0}'''.$$

This is proved on page 66 of [10].

2.6 The Limit Lemma

In this section we will provide a characterization of the degrees below $\mathbf{0}'$. We will later use this characterization in one of our solutions to Post's problem.

Definition 2.6.1. A sequence of functions $\{f_s\}_{s\in\omega}$ *converges* to f if for all x there exists a t such that $f_s(x) = f(x)$ for all $s \ge t$. We write this as $\lim_s f_s = f$. We say the sequence is *recursive* if there is some recursive function $\hat{f}(x,s)$ such that $\hat{f}(x,s) = f_s(x)$ for all x, s. Likewise we define it to be A-*recursive* if $\hat{f}(x,s) = f_s(x)$ is A-recursive.

A *modulus* for a converging sequence $\{f_s\}_{s\in\omega}$ is a function $m(x)$ such that $s \ge m(x) \implies f_s(x) = f(x)$ for all s, x.

These moduli and how they relate to f are of interest when we are examining r.e. sets.

Lemma 2.6.2 (Modulus Lemma). *Let A be r.e., and let f be a function such that $f \leq_T A$. There is a recursive sequence $\{f_s\}_{s\in\omega}$ converging to f, and a modulus m of $\{f_s\}_{s\in\omega}$ such that $m \leq_T A$.*

Proof. Find an index e for f such that $f = [\![e]\!]^A$, and an index i for A such that $A = W_i$. Let $A_s = W_{i,s}$. Define

$$f_s(x) = \begin{cases} [\![e]\!]_s^{A_s}(x) & \text{if } [\![e]\!]_s^{A_s}(x)\downarrow, \\ 0 & \text{otherwise.} \end{cases}$$

and

$$m(x) = \mu s\ \exists z \leq s\ [[\![e]\!]_s^{A_s \upharpoonright z}(x)\downarrow\ \wedge\ A_s \upharpoonright z = A \upharpoonright z].$$

The function $\hat{f}(x,s) = f_s(x)$ must then be recursive, so it is a recursive sequence.

We now examine the definition of m; the relation $A_s \upharpoonright z = A \upharpoonright z$ must be recursive in A, the relation $[\![e]\!]_s^{A_s\upharpoonright z}(x)\downarrow$ is recursive, and the quantifier is bounded, hence $m \leq_T A$. Let now x, s be given such that $s \geq m(x)$. Then we have a z such that $A_s \upharpoonright z = A \upharpoonright z$ and $[\![e]\!]_s^{A_s\upharpoonright z}(x)\downarrow$, so

$$[\![e]\!]_s^{A_s\upharpoonright z}(x) = [\![e]\!]_s^{A\upharpoonright z}(x) = [\![e]\!]^A(x) = f(x)$$

by the Use Principle, and therefore m is a modulus. □

Now we can prove the Limit Lemma, which provides a characterization of the degrees below $\mathbf{0}'$.

Lemma 2.6.3 (Limit Lemma). *For a function f, $f \leq_T \varnothing'$ iff there is a recursive sequence $\{f_s\}_{s\in\omega}$ converging to f.*

Proof. Assume $f \leq_T \varnothing'$. Since $\varnothing'$ is r.e., the Modulus Lemma ensures the existence of a recursive sequence $\{f_s\}_{s\in\omega}$ converging to f.

Assume $f = \lim_s f_s$, where $\{f_s\}_{s\in\omega}$ is a recursive sequence with $\hat{f}(x,s) = f_s(x)$. Define the set A by

$$\langle s,x\rangle \in A \iff \exists t \geq s\ [\hat{f}(x,t) \neq \hat{f}(x,t+1)].$$

Since $\hat{f}$ is recursive, then A must be Σ_1, and thus $A \leq_T \varnothing'$. Next, define

$$m(x) = \mu s\ [\langle s,x\rangle \notin A],$$

which is A-recursive, and the least modulus for $\{f_s\}_{s\in\omega}$, since

$$\begin{aligned} \langle s,x\rangle \notin A &\iff \forall t \geq s\ [\hat{f}(x,t) = \hat{f}(x,t+1)] \\ &\iff \forall t \geq s\ [f_t(x) = f_{t+1}(x)] \\ &\iff \forall t \geq s\ [f_t(x) = f(x)]. \end{aligned}$$

Since $f(x) = f_{m(x)}(x)$ can be computed with m and $m \leq_T A$, then

$$f \leq_T A \leq_T \varnothing'.$$

□

By combining these lemmas, we can characterize the r.e. degrees below $\mathbf{0}'$.

Corollary 2.6.4. *A set A has r.e. degree iff there is a recursive sequence $\{f_s\}_{s\in\omega}$ converging to χ_A with a modulus $m \leq_T A$.*

Proof. If A has r.e. degree, apply the Modulus Lemma to obtain $\{f_s\}_{s\in\omega}$ and $m \leq_T A$.

Assume $\chi_A = \lim_s f_s$ with modulus $m \leq_T A$. As in the proof of the Limit Lemma, define B by

$$\langle s, x\rangle \in B \iff \exists t \geq s\ [f_t(x) \neq f_{t+1}(x)],$$

and recall that $A \leq_T B$ and that B is r.e. Observe that we can compute B by using m, so since $m \leq_T A$, then $B \leq_T A$. Thus $\deg(A) = \deg(B)$. □

Chapter 3

Finite Injury Priority Method

In this chapter, Post's problem will be solved. It was originally solved by R.M. Friedberg in 1957 [1], and independently by A.A. Muchnik (or Mučnik) in 1956 [6], and both invented the same new technique which is today known as the *finite injury priority method.* It is a sophisticated construction method, where some sets are built, step by step, in a way where they are highly dependent on each other. As the construction advances, the sets will eventually *injure* each other—but as we will see (and as the title suggests), it will only happen a finite amount of times at each step, and so our construction will eventually succeed.

Since its discovery, this technique has been refined, and there exists a much more powerful variant of it called *infinite injury*, which will not be treated here.

We will provide two different solutions to Post's problem. Firstly the original Friedberg-Muchnik solution, and then a newer solution which is a bit easier. Friedberg-Muchniks theorem states the existence of two r.e. sets A and B, which are $\leq_T$-incomparable, and thus $\mathbf{0} < \deg(A) < \mathbf{0}'$ and $\mathbf{0} < \deg(B) < \mathbf{0}'$, since if, say, $A \equiv_T \varnothing'$ then $B \leq_T A$ and if $B \equiv_T \varnothing$ then $A \leq_T B$. The other solution finds only one r.e. set A, but this is low, i.e., $A' \equiv_T K$, and non-recursive, so $\mathbf{0} < \deg(A) < \mathbf{0}'$.

These versions of the proofs are primarily based on S. Lempp's lecture notes [5], and his approach with strategy trees, but they should be somewhat more rigorous.

3.1 The Friedberg-Muchnik Theorem

Theorem 3.1.1 (The Friedberg-Muchnik Theorem). *There exists r.e. sets A and B such that $A \not\leq_T B$ and $B \not\leq_T A$.*

Proof. We will construct sets A and B that are r.e. and satisfy the requirements

$$\begin{aligned} R_e^A : &\quad \chi_A \neq [\![e]\!]^B \\ R_e^B : &\quad \chi_B \neq [\![e]\!]^A \end{aligned}$$

for all e, for this will ensure that neither $A \leq_T B$ nor $B \leq_T A$.

We will build the sets recursively in ω stages, and at each stage s create approximating finite sets A_s and B_s, such that

$$\begin{aligned} &A_0 \subseteq A_1 \subseteq A_2 \subseteq \ldots, \\ &B_0 \subseteq B_1 \subseteq B_2 \subseteq \ldots. \end{aligned}$$

Since this construction will be an effective procedure, $A = \bigcup_s A_s$ and $B = \bigcup_s B_s$ will be r.e. by the Listing Theorem.

For technical reasons we will order the requirements linearly, so we assign each even number to a requirement of the first type, and each odd number to a requirement of the second type, namely we say $R_{2e} = R_e^A$ and $R_{2e+1} = R_e^B$. We say R_i is of *higher priority* than R_e if $i < e$.

We will fulfill the requirements by finding *witnesses* to them. A witness x for, say, R_{2e} shall fulfill that either $\chi_A(x) = 0$ and $[\![e]\!]^B(x) \neq 0$ or $\chi_A(x) = 1$ and $[\![e]\!]^B(x) = 0$.

The construction will then roughly proceed in the following way. At stage $s+1$ of the construction, we find the highest priority requirement which *requires attention*, i.e., the least i such that (assume that $i = 2e$ and x_{2e} is the *potential witness* for R_{2e} chosen at a previous stage)

$$\chi_{A_s}(x_{2e}) = [\![e]\!]_{s+1}^{B_s}(x_{2e}).$$

Action is then taken for this requirement. This consists of finding a new unused potential witness x_{2e}, not restrained by any higher priority requirement, on which it is desired to obtain

$$\chi_A(x_{2e}) \neq [\![e]\!]^B(x_{2e}),$$

and keep it out of A. Then we wait for a stage s_0 where $[\![e]\!]_{s_0}^{B_{s_0}}(x_{2e}) = 0$, and if such a stage is found, we enumerate x_{2e} into A, and restrain any number smaller than $u(B_{s_0}; e, x_{2e}, s_0)$ from entering B at a later stage.

A higher priority requirement R_{2j+1} can later *injure* R_{2e} by enumerating numbers into B which are smaller than this restraint, and thus can effect the computation. But each requirement will only act finitely often after it is not injured anymore, and thus by induction all the requirements will eventually be met.

We systematize this construction in the following way. We will make *strategies* to fulfill each requirement, and each strategy will depend on the outcomes of all strategies for higher priority requirements. The strategies can be seen as effective procedures executed step by step, which can always remember where they left of. There will always be one strategy which is *eligible to act*, and it will act according to where it left off in the following description.

Strategy for fulfilling R_{2e}:

(1) Let $r = \max_{2i+1 \leq 2e}\{r_{2i+1}\}$ be the maximum restraint for A given by the higher order priorities. Choose a witness x_{2e} to be the least element in $\{\langle y, 2e\rangle : y \in \omega\}$ which is greater than r, and which is not already in A_s. Let $r_{2e} = -1$.

(2) Wait until
$$[\![e]\!]_s^{B_s}(x) = 0.$$

(3) Let $r_{2e} = u(B_s; e, x, s)$, and stop.

Strategy for fulfilling R_{2e+1}: The same as the above, switching A and B and replacing $2e$ with $2e+1$.

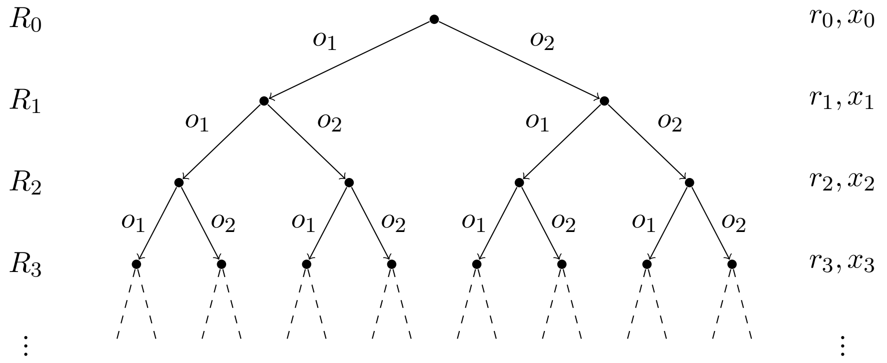
When a strategy at some stage has *acted*, then it will always have one of the following *current outcomes*:

o_1: We are waiting at (2), so we wish that the potential witness should not be in the set.

o_2: We have stopped at (3), so we wish that the potential witness shall be enumerated into the set.

We define the set of strings $T = \{o_1, o_2\}^{<\omega}$ and call it the *Tree of Strategies.* A string $\alpha \in T$ with the length i represents a strategy for fulfilling R_i, and if this strategy is eligible to act and has current outcome o, then the next strategy eligible to act will be $\alpha\hat{}\langle o\rangle$.

The reason for calling T a tree, is that it can be visualized as the following graph:



Each level of nodes represents all the strategies for fulfilling one requirement, and each of these levels have a restraint and a potential witness, which the strategies will alter.

Construction of A and B: At stage $s = 0$, we let the current path of the construction be $\gamma_0 = \varnothing \in T$ and define $\chi_{A_0}(x) = \chi_{B_0}(x) = 0$ for all x.

At stage $s+1$ we will go through substages i from 0 to $s+1$. At substage i, a strategy $\alpha \in T$ of length i is eligible to act. If α has not acted before, then it starts at (1), and if it has acted before, then either it continues from (2) or it has stopped at (3). Either way it will have an outcome, call this o. The next strategy eligible to act will be $\alpha\hat{}\langle o\rangle$.

Denote by $\gamma_{s+1} \in T$ the last strategy eligible to act at this stage, i.e., the current path of the construction. We define A_{s+1} and B_{s+1} in the following way:

Let $\chi_{A_{s+1}}(x) = 1$ if there exists a $2e = i \leq s+1$ such that $x = x_{2e}$ and $\gamma_{s+1}(2e) = o_2$, and $= \chi_{A_s}(x)$ otherwise; let $\chi_{B_{s+1}}(x) = 1$ if there exists an $2e+1 = i \leq s+1$ such that $x = x_{2e+1}$ and $\gamma_{s+1}(2e+1) = o_2$, and $= \chi_{B_s}(x)$ otherwise. Thus A_{s+1} and B_{s+1} have been constructed recursively. We define the sets $A = \bigcup_s A_s$ and $B = \bigcup_s B_s$, which are r.e. since each A_s and B_s is finite.

We see that the paths of construction γ_s will converge to an infinite path $\gamma \in \{o_1, o_2\}^\omega$, because the outcome of each strategy $\alpha \in \gamma_s$ will eventually reach a limit—for either the outcome is o_1 for all stages, or it becomes o_2 at some stage, at which point it will not be able to change back to o_1 again (in the visualization of the tree, the current path of construction will only be able to move right, so every requirement R_i can at most be injured $2^i - 1$ times). Hence, we can for every requirement R_i find a strategy $\alpha \in T$ with length i such that $\alpha \subset \gamma$, i.e., there is an s_0 such that $\alpha \subseteq \gamma_s$ for all $s \geq s_0$.

Lemma. *Every strategy $\alpha \subset \gamma$ ensures the satisfaction of its requirement, and thus $A \not\leq_T B$ and $B \not\leq_T A$.*

Fix $s_0 \geq \mathrm{lh}(\alpha)$ least such that $\alpha \subseteq \gamma_s$ for all $s \geq s_0$. Thus at stage s_0, α picks $x_{\mathrm{lh}(\alpha)}$ as a witness. Assume that $\mathrm{lh}(\alpha) = 2e$, the proof will be analogous for $\mathrm{lh}(\alpha) = 2e+1$.

Case 1: The outcome of α is o_1 for all stages $s \geq s_0$. Then x_{2e} will never be put into A_s (it can never be chosen as potential witness for any other requirement, since it was specifically chosen from the set $\{\langle y, 2e\rangle : y \in \omega\}$), and $[\![e]\!]_s^{B_s}(x_{2e}) \neq 0$ for all s, so $[\![e]\!]^B(x_{2e}) \neq 0$. Thus R_{2e} is satisfied.

Case 2: For some stage s_1, the outcome of α is o_2 for all stages $s \geq s_1$. Then x_{2e} is enumerated into A_{s_1}, and so

$$[\![e]\!]_{s_1}^{B_{s_1}}(x_{2e}) = 0 \neq 1 = \chi_{A_{s_1}}(x).$$

It remains to be shown, that no other strategy β will ever enumerate any number smaller than $u(B_{s_1}; e, x_{2e}), s_1) = r_{2e}$ into B after substage $t = \mathrm{lh}(\alpha)$ of stage s_1, for then $B_{s_1} \upharpoonright r_{2e} = B \upharpoonright r_{2e}$, and thus by the Use

Principle

$$[\![e]\!]_{s_1}^{B_{s_1}}(x_{2e}) = [\![e]\!]^B(x_{2e}) = 0.$$

If $\beta \subset \alpha$, it holds, for β cannot change its outcome after substage t of s_1 as $\beta \subset \alpha\hat{}\langle o_2\rangle \subseteq \gamma_s$ for all $s \geq s_1$.

If $\beta = \alpha$ it obviously holds; α can only enumerate into A, not B.

If $\beta \supseteq \alpha\hat{}\langle o_2\rangle$ it holds, because β must choose its witness after α, and thus it must be greater than r_{2e}.

For $\beta \nsubseteq \alpha$ and $\beta \nsupseteq \alpha$ it also holds, because β will never be eligible to act after substage t of s_1.

□

3.2 A Low, Non-Recursive r.e. Degree

Definition 3.2.1. A set A is *low* if $A \leq_T \varnothing'$ and $\deg(A)' = \mathbf{0}'$. The recursive degree $\mathbf{0}$ contains the trivial low sets.

Theorem 3.2.2. *There exists a non-trivial low r.e. set.*

Proof. We will construct A and a recursive function Γ such that for all e the following requirements are fulfilled:

$$\begin{aligned} R_{2e} = N_e :&\quad \chi_{A'}(e) = \lim_s \Gamma(e,s) \\ R_{2e+1} = P_e :&\quad \chi_A \neq [\![e]\!]. \end{aligned}$$

The *negative* requirements N_e ensures by the Limit Lemma that $A' \leq_T \varnothing'$ (lowness), and the *positive* requirements P_e ensures that A is non-recursive.

Strategy for fulfilling N_e:

(1) Let $r_e = -1$, and set $\Gamma(e,t) = 0$ for all t.

(2) Wait until
$$[\![e]\!]_s^{A_s}(e)\downarrow.$$

(3) Let $\Gamma(e,t) = 1$ for each $t \geq s$, and set $r_e = u(A_s; e, x, s)$.

Strategy for fulfilling P_e:

(1) Let $r = \max_{i\leq e}\{r_i\}$ be the maximum restraint. Choose a witness x_e to be the least element in $\{\langle y,e\rangle : y \in \omega\}$ which is greater than r, and which is not already in A_s.

(2) Wait until
$$[\![e]\!]_s(x) = 0.$$

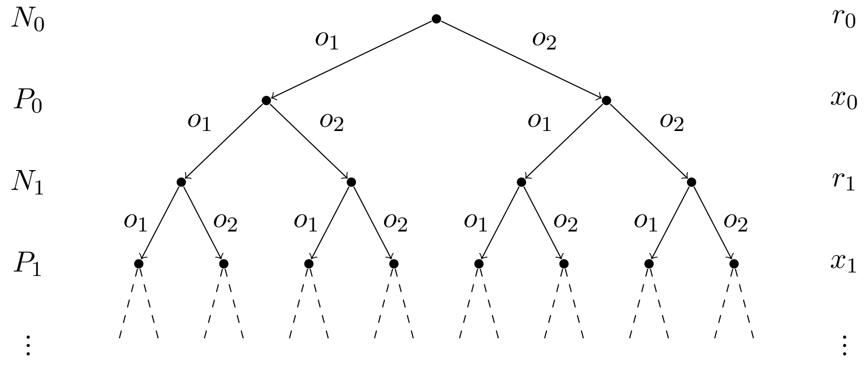
(3) Stop.

Each strategy will have one of the following current outcomes:

o_1: We wait at (2).

o_2: We have stopped at (3).

Let $T = \{o_1, o_2\}^{<\omega}$ be the Tree of Strategies, where a string $\alpha \in T$ of length e represents a strategy for fulfilling requirement R_e.



Construction of A: At stage $s = 0$, we let the current path of construction $\gamma_0 = \varnothing$, and define $\chi_{A_s}(x) = 0$ for all x.

At stage $s + 1$, we will go through substages i from 0 to $s + 1$. At each i, a strategy $\alpha \in T$ of length i is eligible to act. If α has not acted before, then it will start from (1), and if it has acted before it will either check if it is still waiting at (2) or simply be stopped at stage (3). Either way it will have an outcome, call this o. The next strategy eligible to act is then $\alpha\hat{}\langle o\rangle$. Denote by $\gamma_{s+1} \in T$ the last strategy eligible to act at this stage, i.e., the current path of the construction. Define $\chi_{A_{s+1}}(x) = 1$ if there exists an e such that $x = x_e$ and $\gamma_s(2e) = o_2$, and $= \chi_{A_s}(x)$ otherwise, and thus we have constructed A_{s+1} recursively. We define our set to be $A = \bigcup_s A_s$, so since all A_s are finite, then A will be r.e.

Now we see that γ_s converges to some $\gamma \in \{o_1, o_2\}^{\omega}$, since each strategy will either have outcome o_1 forever, or outcome o_2 forever from some point on (the path can only move to the right in the tree). So we can for each requirement R_i find a strategy $\alpha \in T$ of length i such that $\alpha \subset \gamma$, i.e., so there is an s_0 such that $\alpha \subseteq \gamma_s$ for all $s \geq s_0$.

Note that it is when we try to fulfill P_e that we can find new elements to put into A, and it is when we try to fulfill N_e that we block certain elements from entering A, hence they are called positive and negative requirements, respectively.

Lemma. *Every strategy $\alpha \subset \gamma$ ensures the satisfaction of its requirement, thus A is low and non-recursive.*

Fix $s_0 \geq \mathrm{lh}(\alpha)$ least such that $\alpha \subseteq \gamma_s$ for all $s \geq s_0$, i.e., the first stage at which α is eligible to act.

Case 1: The length of α is $2e$, so it is an N_e-strategy: After step s_0, α will define $\Gamma(e,t)$ for all t. If α has outcome o_1 for all stages $s \geq s_0$, then it waits at (2) and so $[\![e]\!]^A(e) \uparrow$ and thus $e \notin A'$, but $\Gamma(e,s) = 0$. If α has outcome o_2 from some stage $s_1 \geq s_0$, then $\Gamma(e,t) = 1$ for all $t \geq s_1$, and since r_e ensures that $A_{s_1} \upharpoonright r_e = A \upharpoonright r_e$, then by the Use Principle

$$[\![e]\!]_s^{A_s}(e) = [\![e]\!]_s^{A_s \upharpoonright r_e}(e) = [\![e]\!]^{A \upharpoonright r_e}(e) = [\![e]\!]^A(e)$$

and so $e \in A'$.

In either case, $A'(e) = \lim_s \Gamma(e,s)$.

Case 2: The length of α is $2e+1$, so it is a P_e-strategy: At stage s_0, α picks x_e as a witness. If it has outcome o_1 for all stages $s \geq s_0$ then it will never be put into A (it can never be chosen by a strategy for any other requirement, since it is specifically chosen from the set $\{\langle y,e\rangle : y \in \omega\}$), but then $\chi_{A_s}(x_e) = 0$ and $[\![e]\!]_s(x_e) \neq 0$ for all $s \geq s_0$, and thus $\chi_A(x_e) \neq [\![e]\!](x_e)$. If α has outcome o_2 some stage $s_1 \geq s_0$, then it will have this outcome for all $s \geq s_1$. Then $\chi_{A_s}(x_e) = 1 \neq 0 = [\![e]\!]_s(x_e)$ for all $s \geq s_1$, hence $\chi_A(x_e) \neq [\![e]\!](x_e)$.

□

Bibliography

[1] R.M. Friedberg, *Two Recursively Enumerable Sets of Incomparable Degrees of Unsolvability (Solution of Post's Problem, 1944)*, Proceedings of the National Academy of Sciences of the United States of America **43** (1957), no. 2, pp. 236–238.

[2] S. C. Kleene, *On notation for ordinal numbers*, The Journal of Symbolic Logic **3** (1938), no. 4, pp. 150–155.

[3] ———, *Introduction to Metamathematics*, North-Holland, 1952.

[4] S. C. Kleene and Emil L. Post, *The upper semi-lattice of degrees of recursive unsolvability*, The Annals of Mathematics **59** (1954), no. 3, pp. 379–407.

[5] S. Lempp, *Priority Arguments in Computability Theory, Model Theory, and Complexity Theory*, manuscript, available at `http://www.math.wisc.edu/~lempp/papers/prio.pdf` (accessed 9. May 2011).

[6] A.A. Muchnik, *On the unsolvability of the problem of reducibility in the theory of algorithms*, Dokl. Akad. Nauk SSSR, NS (Russian) **108** (1956), 194–197.

[7] E.L. Post, *Recursively enumerable sets of positive integers and their decision problems*, Bull. Am. Math. Soc. **50** (1944), 284–316.

[8] H.G. Rice, *Classes of recursively enumerable sets and their decision problems*, Transactions of the American Mathematical Society **74** (1953), no. 2, pp. 358–366.

[9] J.R. Shoenfield, *Mathematical Logic*, Addison Wesley, 1967.

[10] R.I. Soare, *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*, Springer-Verlag, Berlin, 1987.

[11] ———, *Computability and recursion*, The Bulletin of Symbolic Logic **2** (1996), no. 3, 284–321.

[12] A.M. Turing, *On computable numbers, with an application to the Entscheidungsproblem.*, Proceedings of the London Mathematical Society **42** (1936), no. 2, 230–65.

[13] ______, *Systems of logic based on ordinals*, Proceedings of the London Mathematical Society **2** (1939), no. 1, 161.