Chapter 3

Random number

For a given finite $\{0, 1\}$-sequence x, let us consider a computer program which produces it. If x can be produced by a short program, it is considered to be regular, and otherwise, it is considered to be irregular. Consequently, it is a good idea to call x a random number if a very long program is needed to produce it ([5, 18, 19, 20, 26]). To make this idea precise and universal, we use the notion of partial recursive function ([5, 18, 19, 20, 21, 26], cf. [23, 54]). In this chapter, two fundamental theorems (Theorem 3.3 and Theorem 3.6, whose detailed proofs are not given here) are introduced as the basis of discussion, from which other theorems are derived. In particular, Theorem 3.15, which asserts that it is impossible to judge whether a given $\{0, 1\}$-sequence is a random number or not, and Theorem 3.20, which says about the relation between random numbers and statistical tests, are very important.

The notion of random number may not directly solve practical problems, but recognizing it brings us profound understanding of the Monte Carlo method as well as probability theory itself (§ 3.6).

3.1 Partial recursive function

Modern computers can deal with many kinds of data; as input, data from keyboard, mouse, scanner, and video camera, ..., as output, document, picture, sound, video, control sequence for electronic machine, ... But in the final analysis, all of them are binary strings, i.e., finite $\{0, 1\}$-sequences.[†1] Since each finite $\{0, 1\}$-sequence can be associated with a non-negative integer via dyadic expansion (§ 3.1.3), all data that computers deal with can be essentially regarded as non-negative integers. Namely, any action of computer can be regarded as a function $f : \mathbb{N} \to \mathbb{N}$.

Each action of computer is determined by a program, which, just like all input/output data, can be regarded as a finite $\{0, 1\}$-sequence, or a non-negative integer, too. The set of all programs is therefore a countable set. In other words, among uncountably many functions $f : \mathbb{N} \to \mathbb{N}$, only countably many ones can be realized by actions of computer.

The notion of partial recursive function is used to express the actions of computer mathematically. By this notion, every action of computer, including infinite loops that do

[†1] Here we do not assume data processing of infinite input or infinite output.

not stop, can be expressed. In this section, for the later use, we introduce some relevant concepts and theorems about partial recursive functions.

3.1.1 Primitive recursive function and partial recursive function

Definition 3.1 (*Primitive recursive function*, cf. [6, 32])

1. (Basic functions)

$$\begin{array}{rcll} \mathrm{zero} & : & \mathbb{N}^0 \to \mathbb{N}, & \mathrm{zero}(\,) := 0 \\ \mathrm{suc} & : & \mathbb{N} \to \mathbb{N}, & \mathrm{suc}(x) := x+1 \\ p_i^n & : & \mathbb{N}^n \to \mathbb{N}, & p_i^n(x_1,\dots,x_n) := x_i, \qquad i = 1,\dots,n, \end{array}$$

   are primitive recursive functions.

2. (Composition)
   If $g : \mathbb{N}^m \to \mathbb{N}$, $g_j : \mathbb{N}^n \to \mathbb{N}$, $j = 1,\dots,m$, are primitive recursive functions, then so is

$$f : \mathbb{N}^n \to \mathbb{N}, \qquad f(x_1,\dots,x_n) := g(g_1(x_1,\dots,x_n),\dots,g_m(x_1,\dots,x_n)).$$

3. (Recursion)
   If $g : \mathbb{N}^n \to \mathbb{N}$ and $h : \mathbb{N}^{n+2} \to \mathbb{N}$ are primitive recursive functions, then so is $f : \mathbb{N}^{n+1} \to \mathbb{N}$ defined by

$$\left\{ \begin{array}{lcl} f(x_1,\dots,x_n,0) & := & g(x_1,\dots,x_n), \\ f(x_1,\dots,x_n,y+1) & := & h(x_1,\dots,x_n,y,f(x_1,\dots,x_n,y)). \end{array} \right.$$

4. (Direct product)
   If $g_j : \mathbb{N}^{n_j} \to \mathbb{N}$, $j = 1,\dots,m$, are primitive recursive functions, then so is

$$g : \mathbb{N}^{n_1+\cdots+n_m} \to \mathbb{N}^m, \qquad g := (g_1,\dots,g_m).$$

5. A function $\mathbb{N}^m \to \mathbb{N}^n$ obtained from the basic functions by a finite combination of compositions, recursions, and direct products is a primitive recursive function, and any primitive recursive function can be obtained in this way.

Definition 3.2 (*Partial recursive function*, cf. [6, 32])

1. (μ-operation)
   For a partial function[†2] $p : \mathbb{N}^{n+1} \to \mathbb{N}$, we define $\mu_y(p(\bullet,\cdots,\bullet,y)) : \mathbb{N}^n \to \mathbb{N}$ by

$$\mu_y(p(x_1,\dots,x_n,y)) := \left\{ \begin{array}{ll} y_0 & (\,\exists y_0 \text{ s.t. } 0 \le \forall y < y_0,\ p(x_1,\dots,x_n,y) > 0, \\ & \ \text{and } p(x_1,\dots,x_n,y_0) = 0\,), \\ \text{undefined} & (\text{otherwise}). \end{array} \right.$$

[†2] $f : \mathbb{N}^m \to \mathbb{N}^n$ is called a *partial* function if f is defined on a subset of $\mathbb{N}^m$ and it is $\mathbb{N}^n$-valued. It is called a *total* function if it is defined on the whole domain $\mathbb{N}^m$.

2. A function $\mathbb{N}^m \to \mathbb{N}^n$ obtained from the basic functions by a finite combination of compositions, recursions, direct products, and μ-operations is a partial recursive function, and any partial recursive function can be obtained in this way.

Primitive recursive functions are partial recursive functions. If a partial recursive function happens to be defined on the whole domain, it is called a *total recursive function*. Any action of computer can be expressed as a partial recursive function, and any partial recursive function can be realized by a Turing machine.[†3]

3.1.2 Kleene's normal form

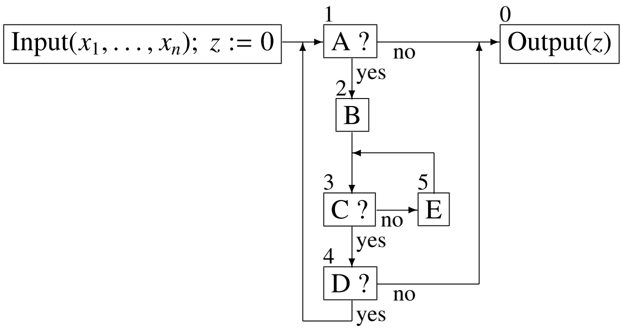
The following theorem describes the general structure of partial recursive functions.

Theorem 3.3 *(Kleene's normal form) For any partial recursive function $f : \mathbb{N}^n \to \mathbb{N}$, there exist two primitive recursive functions $g, p : \mathbb{N}^{n+1} \to \mathbb{N}$ such that*

$$f(x_1, x_2, \dots, x_n) = g(x_1, x_2, \dots, x_n, \mu_y(p(x_1, x_2, \dots, x_n, y))), \quad (x_1, x_2, \dots, x_n) \in \mathbb{N}^n. \tag{3.1}$$

See [6, 32] for a detailed proof. We here only give an idea of the proof by explaining an example. The point is that even if a given program has more than one loops, which correspond to μ-operations, we can rearrange them into a single loop.

Figure 3.1: Flow chart (A)



Let Figure 3.1 (Flow chart (A)[†4]) be a flow chart to compute the function f. $\boxed{\text{A ?}}$ $\boxed{\text{C ?}}$ $\boxed{\text{D ?}}$ denote conditions of branches, and $\boxed{\text{B}}$ $\boxed{\text{E}}$ are procedures without loops (i.e., calculation of primitive recursive functions) which set some values of z, respectively. This program includes the main loop $\boxed{\text{A ?}} \to \boxed{\text{B}} \to \boxed{\text{C ?}} \to \boxed{\text{D ?}} \to \boxed{\text{A ?}}$, a nested loop $\boxed{\text{C ?}} \to \boxed{\text{E}} \to \boxed{\text{C ?}}$, and an escape branch from the main loop at $\boxed{\text{D ?}}$. Let us show that these loops can be rearranged into a single loop by introducing a new variable u. To do this, we put numbers $0 \sim 5$ respectively at left-top of the boxes of all procedures $\boxed{\text{A ?}}$ $\boxed{\text{C ?}}$ $\boxed{\text{D ?}}$ $\boxed{\text{B}}$ $\boxed{\text{E}}$ and the output procedure in order for the variable u to refer (Figure 3.1).

[†3]A virtual computer, having infinite memory. See [6, 32]

[†4]Flow charts (A), (B) are slight modifications of Figure 3 (p.12), Figure 4 (p.13) of [45], respectively.

Figure 3.2: Flow chart (B)

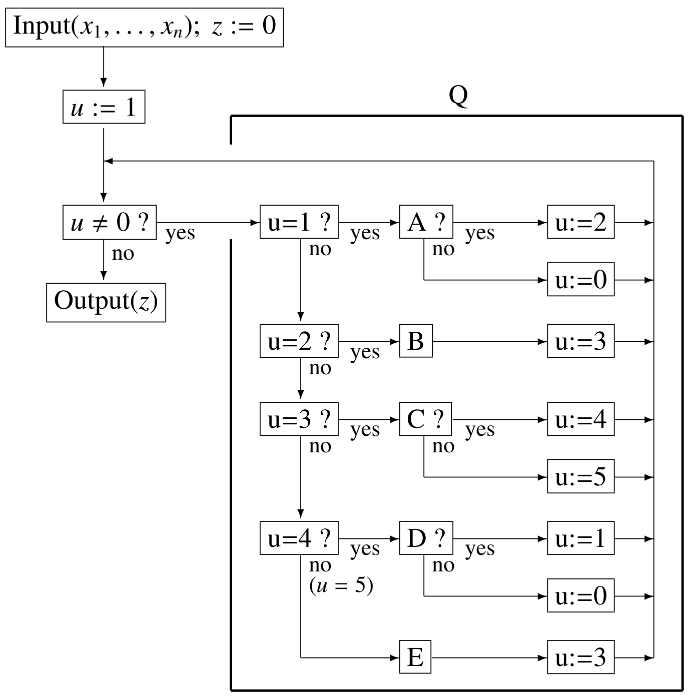


Figure 3.2 (Flow chart (B)) shows a rearrangement of Flow chart (A). It is easy to confirm that Flow chart (B) also computes the same function f. Let us show that f, which Flow chart (B) computes, can be expressed in the form (3.1). Let Q be a procedure consisting of all the procedures enclosed by the thick lines in Flow chart (B). Define $g(x_1, \ldots, x_n, y)$ as the value of the output variable z that is produced after Q being executed y times under the input $(x_1, \ldots, x_n)$, and define $p(x_1, \ldots, x_n, y)$ as the value of u after Q being executed y times under the input $(x_1, \ldots, x_n)$. Then we see (3.1) holds.

3.1.3 Canonical order

Each finite $\{0, 1\}$-sequence can be regarded as a non-negative integer by dyadic expansion. This is explicitly formulated as follows.

Let $\{0, 1\}^* := \bigcup_{n\in\mathbb{N}}\{0, 1\}^n$. Namely $\{0, 1\}^*$ is the set of all finite $\{0, 1\}$-sequences. In particular, the $\{0, 1\}$-sequence of length 0 is called the *empty word*. The *canonical order* of $\{0, 1\}^*$ is defined in the following way; for $x, y \in \{0, 1\}^*$, if x is longer than y then define $x > y$, if x and y have a same length then define the order regarding them as dyadic integer. From now on, we identify $\{0, 1\}^*$ with $\mathbb{N}$ by the canonical order. Namely, the empty word= 0, “0”= 1, “1”= 2, “00”= 3, “01”= 4, “10”=5”, “11”= 6, “000”= 7, ...

3.1.4 Enumeration theorem and halting problem

We introduce a function $G^n : (\{0,1\}^*)^n \to \{0,1\}^*$ to make a given function of n variables into a function of a single variable that is equivalent to the original one.

First, if $x_i \in \{0,1\}^{m_i}$, $i = 1, 2$, are given by

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{im_i}), \quad x_{ij} \in \{0,1\}, \quad i = 1, 2,$$

define

$$G^2(x_1, x_2) := (x_{11}, x_{11}, x_{12}, x_{12}, \ldots, x_{1m_1}, x_{1m_1}, 0, 1, x_{21}, x_{22}, \ldots, x_{2m_2}). \tag{3.2}$$

Next, define inductively as[†5]

$$G^n(x_1, x_2, \ldots, x_n) := G^2(x_1, G^{n-1}(x_2, \ldots, x_n)), \quad n = 3, 4, \ldots.$$

For simplicity, we use the following notation;

$$\langle x_1, \ldots, x_n \rangle := G^n(x_1, \ldots, x_n).$$

The inverse function is written for $u = \langle x_1, x_2, \ldots, x_n \rangle$ as

$$(u)_i := x_i, \quad i = 1, 2, \ldots, n.$$

Definition 3.4 A subset $U \subset \mathbb{N}^n$ is called a *recursively enumerable set* if there exists a partial recursive function $f : \mathbb{N}^n \to \mathbb{N}$ such that U is the domain of definition of f, i.e.,

$$U = \{(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n \mid \exists y \text{ s.t. } f(x_1, x_2, \ldots, x_n) = y\}.$$

The origin of the term "recursively enumerable" can be found in the following theorem (in particular (ii) and (iii) of it).

Theorem 3.5 *For $U \subset \mathbb{N}^n$, the followings are equivalent to each other.*
(i) U is a recursively enumerable set.
(ii) U is an empty set, or it is the image of a primitive recursive function $f : \mathbb{N} \to \mathbb{N}^n$, i.e., $U = f(\mathbb{N})$.
(iii) U is the image of a partial recursive function $f : \mathbb{N} \to \mathbb{N}^n$, i.e., $U = f(\mathbb{N})$.
(iv) There exists a primitive recursive function $p : \mathbb{N}^{n+1} \to \mathbb{N}$ such that

$$U = \{(x_1, x_2, \ldots, x_n) \mid \exists y \text{ s.t. } p(x_1, x_2, \ldots, x_n, y) = 0\}.$$

(v) There exists a partial recursive function $p : \mathbb{N}^{n+1} \to \mathbb{N}$ such that

$$U = \{(x_1, x_2, \ldots, x_n) \mid \exists y \text{ s.t. } p(x_1, x_2, \ldots, x_n, y) = 0\}.$$

[†5]These functions are simple analogues of so-called Gödel functions.

Proof. For simplicity, we prove the theorem for $n = 1$.

(i)$\Longrightarrow$(ii): Assume that $U \neq \emptyset$. By Theorem 3.3, there exist two primitive recursive functions $g, p : \mathbb{N}^2 \to \mathbb{N}$ such that

$$U = \{x \in \mathbb{N} \mid \exists y \text{ s.t. } g(x, \mu_z(p(x, z))) = y\}.$$

Fixing any $a \in U$, consider a function defined by

$$h(u) = \begin{cases} (u)_1 & (u \in G^2(\mathbb{N}^2) \text{ and } p((u)_1, (u)_2) = 0), \\ a & (\text{otherwise}). \end{cases}$$

Then h is a primitive recursive function such that $h(\mathbb{N}) = U$.

(ii)$\Longrightarrow$(iii): Obvious.

(iii)$\Longrightarrow$(iv): By Theorem 3.3, we can assume that $f(x) = g(x, \mu_y(p(x, y)))$, g, p being primitive recursive functions, and that $U = f(\mathbb{N})$. Namely,

$$U = \{z \in \mathbb{N} \mid \exists x, y \text{ s.t. } z = g(x, y),\ p(x, y) = 0,\ \forall w < y,\ p(x, w) \neq 0\}.$$

Now define

$$q(x, y, z) := \begin{cases} 0 & (z = g(x, y),\ p(x, y) = 0,\ \forall w < y,\ p(x, w) \neq 0), \\ 1 & (\text{otherwise}). \end{cases}$$

Then $q(x, y, z)$ is a primitive recursive function, and hence

$$q'(u, z) := \begin{cases} q((u)_1, (u)_2, z) & (u \in G^2(\mathbb{N}^2)), \\ 1 & (u \notin G^2(\mathbb{N}^2)). \end{cases}$$

is also a primitive recursive function, which satisfies that

$$U = \{u \in \mathbb{N} \mid \exists z \text{ s.t. } q'(u, z) = 0\}.$$

(iv)$\Longrightarrow$(v): Obvious.

(v)$\Longrightarrow$(i): Define $f(x) := \mu_y(p(x, y))$, then it is a partial recursive function, which satisfies

$$U = \{x \in \mathbb{N} \mid \exists y \text{ s.t. } y = f(x)\}.$$

□

Theorem 3.6 *(Enumeration theorem) There exists a partial recursive function $univ_n : \mathbb{N} \times \mathbb{N}^n \to \mathbb{N}$ which has the following property; for each partial recursive function $f : \mathbb{N}^n \to \mathbb{N}$, there exists an $e_f \in \mathbb{N}$ such that*

$$univ_n(e_f, x_1, \ldots, x_n) = f(x_1, \ldots, x_n), \quad (x_1, \ldots, x_n) \in \mathbb{N}^n.$$

The function $univ_n$ is called an *enumerating function* or a *universal function*, and e_f is called the *Gödel number* of f. The detailed proof of the theorem can be found in textbooks about computability (e.g., [6, 32, 45]). We here give only an idea of the proof. First, write a computer (say, universal Turing machine) program of a given partial recursive function f, and regard it as a non-negative integer. The Gödel number e_f is such an integer. The enumerating function $univ_n(e, x_1, \ldots, x_n)$ checks if e is a Gödel number of some partial recursive function f of n variables. If it is, $univ_n(e, x_1, \ldots, x_n)$ then reconstruct a partial recursive function f from $e = e_f$, and finally computes $f(x_1, \ldots, x_n)$.

In order for the enumeration theorem to hold, the notion of "partial function" is essential. The following is a theorem about this.

Theorem 3.7 *Any total function which is an extension of the enumerating function $univ_n$ is not recursive.*

Proof. By contradiction.[†6] Suppose that there exists a total recursive function g which is an extension of $univ_n$. Then

$$h(z, x_2, \ldots, x_n) \;:=\; g(z, z, x_2, \ldots, x_n) + 1 \tag{3.3}$$

is also a total recursive function. Therefore using the Gödel number e_h of h, we can write

$$h(z, x_2, \ldots, x_n) \;=\; univ_n(e_h, z, x_2, \ldots, x_n).$$

Since h is total, so is $univ_n(e_h, \cdot, \cdot, \ldots, \cdot)$. Since g is an extension of $univ_n$, we see

$$h(z, x_2, \ldots, x_n) \;=\; univ_n(e_h, z, x_2, \ldots, x_n) \;=\; g(e_h, z, x_2, \ldots, x_n).$$

Putting $z = e_h$ in the above equality, we get

$$h(e_h, x_2, \ldots, x_n) \;=\; g(e_h, e_h, x_2, \ldots, x_n).$$

But (3.3) implies that

$$h(e_h, x_2, \ldots, x_n) \;=\; g(e_h, e_h, x_2, \ldots, x_n) + 1,$$

which is a contradiction. □

As an application of Theorem 3.7, let us consider the following halting problem. Using the enumerating function $univ_n : \mathbb{N} \times \mathbb{N}^n \to \mathbb{N}$, define a total function $halt_n : \mathbb{N} \times \mathbb{N}^n \to \{0, 1\}$ by

$$halt_n(z, x_1, \ldots, x_n) \;:=\; \begin{cases} 1 & (univ_n(z, x_1, \ldots, x_n) \text{ is defined}), \\ 0 & (univ_n(z, x_1, \ldots, x_n) \text{ is not defined}). \end{cases}$$

Namely, $halt_n$ is an indicator function of the domain of $univ_n$. It judges whether a given partial recursive function is defined for a given input $(x_1, \ldots, x_n)$, or equivalently, whether a given program stops or fall into an infinite loop for a given input $(x_1, \ldots, x_n)$.

Theorem 3.8 *([48]) $halt_n$ is not a total recursive function.*

Proof. Consider a function $g : \mathbb{N} \times \mathbb{N}^n \to \mathbb{N}$ defined by

$$g(z, x_1, x_2, \ldots, x_n) \;:=\; \begin{cases} univ_n(z, x_1, x_2, \ldots, x_n) & (halt_n(z, x_1, \ldots, x_n) = 1), \\ 0 & (halt_n(z, x_1, \ldots, x_n) = 0). \end{cases}$$

If $halt_n$ is a total recursive function, so is g. But this is impossible by Theorem 3.7, because g is an extension of the enumerating function $univ_n$. □

Theorem 3.8 asserts that there is no program which computes $halt_n$. This means that there is no program which judges whether a given program stops or fall into an infinite loop for a given input $(x_1, \ldots, x_n)$. This situation is expressed in computer science as "the halting problem is not computable".

[†6]In proving impossibilities seen in such theorems of computer science as this theorem or Theorem 3.15 below, some self-referring versions of "diagonal method" are used. The proof below is self-referring in that we substitute the Gödel number e_h for the first argument of h defined by (3.3).

—— ◇ —— ◇ ——

It is hard to study the theory of partial recursive function rigorously. But even a little knowledge about computer will help you understand it. For example, Kleene's normal form is used to design a CPU architecture in practice. The new variable u is called the program counter of CPU, which indicates the program address where the computer is executing. The function $\langle x_1, \ldots, x_n \rangle$ is a mathematical expression of how to hand more than one parameters to a function. For example, as is seen in an expression like $f(1.5,\ 20.0,\ -2.1)$, a sequence of letters "1.5, 20.0, −2.1" is put into $f(\cdot)$, each parameter being divided by " , ". Here the sequence of letters is coded as a finite $\{0,1\}$-sequence. The delimiter " , " corresponds to "0, 1" seen in the definition (3.2) of $\langle x_1, x_2 \rangle$. The enumerating function of partial recursive functions is a mathematical model of multi-purpose computer, whose action depends on the installed program, or the Gödel number.

About the halting problem, a little knowledge of number theory will help you understand it. Let us consider a partial recursive function

$$f(x) \ := \ \inf\{\, y \geq x \,|\, y \in 2\mathbb{N}^+,\ \ y \text{ cannot be expressed as a sum of two prime numbers}\}.$$

Then if $halt_1$ were a total recursive function, there would exist a program which computes $halt_1(e_f, 4)$, i.e., we would be able to know whether the Goldbach conjecture is true or not. Just like this, the function $halt_1$would solve many other unsolved problems in number theory. This is quite unlikely.

3.2 Kolmogorov complexity and random number

Definition 3.9 For each $p \in \{0,1\}^*$, let $L(p) \in \mathbb{N}$ denote the n such that $p \in \{0,1\}^n$, i.e., $L(p)$ is the length of p. For $p \in \mathbb{N}$, $L(p)$ means the length of the corresponding $\{0,1\}$-sequence to p in the canonical order. For instance, $L(5) = L(\text{"10"}) = 2$. In general, $L(p) = \lfloor \log_2(p+1) \rfloor$ holds.

Definition 3.10 (Computational complexity depending on algorithm) Let $A : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a partial recursive function as a function $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We call A an algorithm. The computational complexity of $x \in \{0,1\}^*$ under the algorithm A with input $y \in \{0,1\}^*$ is defined by

$$K_A(x|y) \ := \ \min\{L(p) \,|\, p \in \{0,1\}^*,\ A(p,y) = x\,\}.$$

If there is no such p that $A(p,y) = x$, we set $K_A(x|y) := \infty$.

In Definition 3.10, the first argument p of A is called a program. So $A(p,y) = x$ means that the program p computes x under the algorithm A from input y. Thus $K_A(x|y)$ returns the length of the shortest program which computes x under A from input y.

Since K_A naturally depends on A, it is not a universal index for complexity. Then we introduce the following theorem.

Theorem 3.11 *There exists an algorithm* $A_0 : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *such that for any algorithm* $A : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$*, we can find such a constant* $c_{A_0A} \in \mathbb{N}$ *that*

$$\forall x,\ y \in \{0,1\}^*, \quad K_{A_0}(x|y) \ \leq \ K_A(x|y) + c_{A_0A}.$$

A_0 *is called a* universal algorithm *or an* asymptotically optimal algorithm.

Proof. Using $univ_2$, define an algorithm A_0 by

$$A_0(z, y) := univ_2((z)_1, (z)_2, y), \qquad z, y \in \{0,1\}^*.$$

If z is not of the form $z = \langle e, p\rangle$, we do not define $A_0(z, y)$. Since $A_0(\langle e_A, p\rangle, y) = A(p, y)$, it follows from (3.2) that

$$\forall x,\ y \in \{0,1\}^*, \quad K_{A_0}(x|y) \le K_A(x|y) + 2L(e_A) + 2.$$

Therefore with $c_{A_0A} := 2L(e_A) + 2$, the theorem holds. □

If A_0 and A_0' are two universal algorithms, there exists a constant $c > 0$ such that

$$\forall x,\ y \in \{0,1\}^*, \quad \left|K_{A_0}(x|y) - K_{A_0'}(x|y)\right| < c. \tag{3.4}$$

This means that when $K_{A_0}(x|y)$ and $K_{A_0'}(x|y)$ are much greater than c, their difference can be ignored.

Definition 3.12 We fix an universal algorithm A_0, and define

$$K(x|y) := K_{A_0}(x|y), \quad x,\ y \in \{0,1\}^*.$$

We call $K(x|y)$ the computational complexity of x given y. In particular, when y is the empty word, we write it as $K(x)$ and call it the *Kolmogorov complexity*[†7] *of* x.

Both $K(x|y)$ and $K(x)$ are defined for all $x, y \in \{0,1\}^*$ and they take finite values.

Theorem 3.13 *(i) There exists a constant* $c > 0$ *such that*

$$\forall n \in \mathbb{N}^+,\ \forall x \in \{0,1\}^n,\ \forall y \in \{0,1\}^*, \quad K(x|y) \le n + c.$$

(ii) If $n > c' > 0$*, then we have*

$$\forall y \in \{0,1\}^*, \quad \#\{x \in \{0,1\}^n \,|\, K(x|y) \ge n - c'\} > 2^n - 2^{n-c'}.$$

Proof. (i) For an algorithm $A(x, y) := p_1^2(x, y) = x$, we have $K_A(x|y) = n$ for $x \in \{0,1\}^n$. Consequently, Theorem 3.11 implies $K(x|y) \le n + c$. (ii) The number of p's such that $L(p) < n - c'$ is equal to $2^0 + 2^1 + \cdots + 2^{n-c'-1} = 2^{n-c'} - 1$, and hence the number of x's $\in \{0,1\}^*$ such that $K(x|y) < n - c'$ is at most $2^{n-c'} - 1$. Thus (ii) holds.[†8] □

Since $K(x)$ is a special case of $K(x|y)$, the assertions (i) and (ii) of Theorem 3.13 are valid for $K(x)$ as well. Therefore when n is so large that the constant c can be ignored, $K(x) \approx n$ holds for almost all x's $\in \{0,1\}^n$. If $x \in \{0,1\}^n$ satisfies $K(x) \approx n$, it is called a *random number*. [†9]

[†7]It is also called the Kolmogorov-Chaitin complexity, algorithmic complexity, description complexity, . . . etc.

[†8]These assertions (i) and (ii) have been essentially discussed in § 2.3.

[†9]Since the value of $K(x)$ has an ambiguity (3.4), the definition of random number should remain a little bit ambiguous like this.

Example 3.14 The world record of computation of π is 2,576,980,370,000 decimal digits or approximately 8,560,543,490,000 bits (as of August 2009). Since the program that produced the record is much shorter than this, the $\{0,1\}$-sequence of π in its dyadic expansion up to 8,560,543,490,000 digit is not a random number.

As is seen in Example 3.14, we know some x's $\in \{0,1\}^*$ which can be proved non-random. However we know no concrete example of random numbers. Indeed, the following theorem implies that there is no algorithm to judge whether a given $x \in \{0,1\}^n$, $n \gg 1$, is random or not.

Theorem 3.15 *For each $y \in \{0,1\}^*$, the function $K(\bullet|y)$ is not a total recursive function, in particular, neither is $K(x)$.*

Proof. Let us identify $\{0,1\}^*$ and $\mathbb{N}$. Fix $y \in \{0,1\}^*$. We show the theorem by contradiction. Suppose that $K(\bullet|y)$ is a total recursive function. Then a function $\psi(x) := \min\{z \in \mathbb{N} \mid K(z|y) \geq x\}$, $x \in \mathbb{N}$, is also a total recursive function. We see $x \leq K(\psi(x)|y)$ by definition. Define an algorithm A by $A(p,y) := \psi(\langle p,y\rangle)$. Then we have

$$K_A(\psi(x)|y) = \min\{L(p) \mid p \in \mathbb{N},\ \psi(\langle p,y\rangle) = \psi(x)\},$$

and consequently, for infinitely many x of the form $x = \langle p,y\rangle$, it holds that $K_A(\psi(x)|y) \leq L(p) \leq L(x)$. Therefore by Theorem 3.11, we know that there exists a constant $c > 0$ such that for infinitely many x,

$$x \leq K(\psi(x)|y) \leq L(x) + c. \tag{3.5}$$

But this is impossible for $x \gg 1$, because $L(x) = \lfloor \log_2(x+1) \rfloor$. □

Let us see a relation between Theorem 3.15 and Theorem 3.8. The following function *complexity* seems to compute $K(x|y)$.

```
function complexity ($x$ : $\{0,1\}^*$) : integer;
begin
    Let $l := 1$.
    Repeat what follows below, and increase $l$ by 1,
      Repeat what follows below for all $z \in \{0,1\}^l$,
        If $z$ is a Gödel number, then
          If $A_0(z,y) = x$, then output $l$, and stop.
end;
```

Starting from the shortest program z, the function *complexity* executes every program z to check whether it computes x from the input y or not, and stops if it does. But the program does not necessarily stop. Indeed, for sufficiently long x, it must fall into an infinite loop before $K(x|y)$ is computed, which cannot be avoided in advance because of Theorem 3.8.

Theorem 3.16 *There exists a primitive recursive function $K'(t,x,y)$ such that*
(i) for each $t,x,y \in \mathbb{N}$, we have $K'(t,x,y) \geq K(x|y)$,
(ii) for each $x,y \in \mathbb{N}$, $K'(t,x,y)$ converges decreasingly to $K(x|y)$ as $t \to \infty$.

Proof. Let $c > 0$ be a constant such that $K(x|y) = K_{A_0}(x|y) < L(x) + c$, and let

$$A_0(p,y) = g(p,y,\mu_z(q(p,y,z))),$$

be Kleene's normal form of the universal algorithm A_0, where $g,\ q : \mathbb{N}^3 \to \mathbb{N}$ are some primitive recursive functions. For each $t \in \mathbb{N}$, we define[†10]

$$\mu_{z<t}(q(p,y,z)) \ := \ \min\left(\{z < t \,|\, q(p,y,z) = 0\} \cup \{t\}\right), \quad p,y \in \mathbb{N}.$$

Note that this is a primitive recursive function as a function of $(t,p,y) \in \mathbb{N}^3$. Let

$$\begin{aligned} g'(t,p,y,z) \ &:= \ \begin{cases} g(p,y,z) & (z < t), \\ 0 & (z \geq t), \end{cases} \\ A(t,p,y) \ &:= \ g'(t,p,y,\mu_{z<t}(q(p,y,z))). \end{aligned}$$

Then $A(t,p,y)$ is also a primitive recursive function. Finally define

$$K'(t,x,y) \ := \ \min\left(\{\, L(p) \,|\, p \in \mathbb{N},\ L(p) < L(x) + c,\ A(t,p,y) = x \,\} \cup \{L(x) + c\}\right),$$

which is what we wish to get. □

3.3 Test and Martin-Löf's theorem

In mathematical statistics, whether an individual $\{0,1\}$-sequence is random or not, i.e., whether it can be regarded as a generic sample of coin tossing process or not, is judged by tests. But the totality of all tests is not consistent.

Example 3.17 (Inconsistency of tests) For elements of $\{0,1\}^n$, let us consider $n-9$ tests $U^{(1)},\ldots,U^{(n-9)}$ corresponding to rejection regions

$$R^{(j)} \ := \ \{\,(x_1,\ldots,x_n) \in \{0,1\}^n \,|\, (x_j, x_{j+1},\ldots,x_{j+9}) \ = \ (0,\ldots,0)\,\}, \quad j = 1,\ldots,n-9.$$

The significance levels of each $U^{(j)}$ are all 2^{-10}. Now, let us assume that an $x \in \{0,1\}^n$ is accepted by all of these tests. Then the test U corresponding to the following rejection region

$$R \ := \ \{\,(x_1,\ldots,x_n) \in \{0,1\}^n \,|\, 1 \leq \forall\, j \leq n-9, \quad (x_j, x_{j+1},\ldots,x_{j+9}) \ \neq \ (0,\ldots,0)\,\}$$

rejects x, although its significance level is arbitrarily small if we take n large. This means that every x is rejected by some $U^{(1)},\ldots,U^{(n-9)}$ or U.

To avoid this inconsistency, adjusting the significance levels of all tests, Martin-Löf constructed a so-called universal test (Theorem 3.19), and showed that for each $x \in \{0,1\}^n$, being a random number in the sense of the last section is equivalent to being accepted by the universal test (Theorem 3.20).

[†10] $\mu_{z<t}$ is called a bounded μ-operation.

3.3.1 Formulation of test and universal test

We first give a general definition of tests.

Definition 3.18 We call $U \subset \mathbb{N} \times \{0,1\}^*$ a *test*[†11] if
(i) U is a recursively enumerable set,
(ii) setting $U_m := \{x \in \{0,1\}^* \,|\, (m,x) \in U\}$, we have $U_m \supset U_{m+1}$, $m \in \mathbb{N}$,
(iii) $\#(U_m \cap \{0,1\}^n) \le 2^{n-m}$, $n > m \ge 0$,
(iv) $(0, 0\,(=\text{empty word})) \in U$.[†12]
For each test U, we define a function

$$m_U(x) := \max\{m \in \mathbb{N} \,|\, x \in U_m\}. \tag{3.6}$$

Here U_m is regarded as a rejection region of the test U of significance level $\le 2^{-m}$. Accordingly we can say; the smaller $m_U(x)$ is, the more easily x is accepted by U, i.e., the more random x is.

Theorem 3.19 *([26]) There exists a test V, which is called a* universal test*, such that for any test U, there exists such a constant $c = c_{VU} \in \mathbb{N}$ that*

$$\forall m \in \mathbb{N}, \qquad U_{m+c} \subset V_m, \tag{3.7}$$

i.e., those x's $\in \{0,1\}^*$ rejected by U with significance level 2^{-m-c} are rejected by V with significance level 2^{-m}.

Proof. We first show an idea of the proof. Let $\{U_e\}_{e\in\mathbb{N}}$ be some enumeration of the set of all tests. Define $V_m \subset \{0,1\}^*$, $m \in \mathbb{N}$, by

$$V_m := \bigcup_{e\in\mathbb{N}} (U_e)_{m+e+1}, \quad \text{where } (U_e)_{m+e+1} := \{x \,|\, (m+e+1, x) \in U_e\}.$$

Then $V_{m+1} \subset V_m$ is obvious, and for any test U, there exists an e such that $U = U_e$ and that $U_{m+e+1} = (U_e)_{m+e+1} \subset V_m$. Moreover, we see

$$\#(V_m \cap \{0,1\}^n) \le \sum_{e\in\mathbb{N}} \#((U_e)_{m+e+1} \cap \{0,1\}^n) \le \sum_{0\le e\le n-m-1} 2^{n-m-e-1} = 2^{n-m} - 1.$$

Thus

$$V := \bigcup_{m\in\mathbb{N}} \{(m,x) \,|\, x \in V_m\}$$

is a universal test, if it is recursively enumerable.

Let us realize the above idea. Let

$$univ_1(e,k) = g(e,k,\mu_z(q(e,k,z))), \quad e,k \in \mathbb{N},$$

[†11]Tests defined here are generalization or abstraction of usual statistical tests. But they are not practical ones.
[†12]The condition (iv) is not essential, and may be removed. But we here assume it to make the proof of Theorem 3.19 simpler.

be Kleene's canonical form of the universal function $univ_1$, where g and q are some primitive recursive functions. For each $t \in \mathbb{N}$, we define a primitive recursive function $\psi : \mathbb{N}^3 \to \mathbb{N}$ by

$$\psi(e,t,k) := g(e,k,\mu_{z<t}(q(e,k,z))).$$

We next define a primitive recursive function $\psi' : \mathbb{N}^3 \to \mathbb{N} \times \{0,1\}^*$ as follows; for $e,t,k \in \mathbb{N}$, let $y := \psi(e,t,k)$, and let

$$\psi'(e,t,k) := \begin{cases} ((y)_1,(y)_2) & (\mu_{z<t}(q(e,k,z)) < t \text{ and } y \in G^2(\mathbb{N}^2)), \\ (0,0) & (\text{otherwise}). \end{cases}$$

Now, for each $e,t \in \mathbb{N}$, let

$$\widetilde{U}_{e,t} := \{(m',x) \,|\, m' \leq m, \text{ where } \psi'(e,t,k) =: (m,x),\ k < t\} \subset \mathbb{N} \times \{0,1\}^*,$$

and let

$$\begin{aligned} U_{e,0} &:= \{(0,0)\}, \\ U_{e,t+1} &:= \begin{cases} U_{e,0} \cup \widetilde{U}_{e,t+1} & (U_{e,0} \cup \widetilde{U}_{e,t+1} \text{ is a test}), \\ U_{e,t} & (\text{otherwise}), \end{cases} \qquad t \in \mathbb{N}. \end{aligned}$$

It is easy to see that $\{U_{e,t}\}_{t\in\mathbb{N}}$ is an increasing sequence of tests, and that the limit $U_e := \bigcup_{t\in\mathbb{N}} U_{e,t}$ is also a test. On the other hand, for any test U, there exists an e such that $U = U_e$.

Now, set

$$V_m := \bigcup_{e\in\mathbb{N}} (U_e)_{m+e+1} = \bigcup_{t\in\mathbb{N}} \bigcup_{e\in\mathbb{N}} (U_{e,t})_{m+e+1}, \quad m \in \mathbb{N},$$

and

$$\begin{aligned} V &:= \bigcup_{m\in\mathbb{N}} \{(m,x) \,|\, x \in V_m\} = \bigcup_{m\in\mathbb{N}} \bigcup_{t\in\mathbb{N}} \bigcup_{e\in\mathbb{N}} \{(m,x) \,|\, x \in (U_{e,t})_{m+e+1}\} \\ &= \bigcup_{m\in\mathbb{N}} \bigcup_{t\in\mathbb{N}} \bigcup_{e\in\mathbb{N}} \{(m,x) \,|\, (m+e+1,x) \in U_{e,t}\}, \end{aligned}$$

which is a universal test, if it is recursively enumerable. Let us construct a partial recursive function $f : \mathbb{N} \to \mathbb{N} \times \{0,1\}^*$ that enumerates V. Let $n \in \mathbb{N}$. If $n \in G^4(\mathbb{N}^4)$, we set $m := (n)_1$, $e := (n)_2$, $t := (n)_3$, and $x := (n)_4$. Using ψ', we can enumerate each set of $\{\widetilde{U}_{e,s}\}_{s\leq t}$, accordingly the set $U_{e,t}$ as well, and we can check if $(m+e+1,x) \in U_{e,t}$ holds. If it does, we define $f(n) := (m,x)$. The function f thus defined is a partial recursive function,[†13] which enumerates V. Hence V is a recursively enumerable set. □

The inclusion relation (3.7) can be rewritten as

$$m_U(x) \leq m_V(x) + c. \tag{3.8}$$

In particular, if V and V' are universal tests, there exists a constant $c > 0$ depending only on V and V' such that

$$\forall x \in \{0,1\}^*, \qquad |m_V(x) - m_{V'}(x)| < c.$$

Namely, they can have only a constant difference. Now, we fix a universal test V and write $m_V(x)$ simply as $m(x)$.

[†13]The enumeration algorithm such as f is often called *dovetailing*.

3.3.2 Martin-Löf's theorem

Theorem 3.20 *([26]) There exists a constant $c > 0$ such that*

$$\forall x \in \{0,1\}^*, \qquad |L(x) - K(x|L(x)) - m(x)| \le c. \tag{3.9}$$

Proof. <u>*Step 1.*</u> The proof of $L(x) - K(x|L(x)) \le m(x) + c$: Set

$$U := \{(m,x) \,|\, K(x|L(x)) < L(x) - m\}.$$

If U is a test, we have

$$m_U(x) = L(x) - K(x|L(x)) - 1,$$

and hence (3.8) implies $m(x) + c > L(x) - K(x|L(x))$ for some constant $c > 0$.

Let us show that U is a test. It is obvious that U satisfies the condition (ii) of Definition 3.18. (iii) follows from Theorem 3.13(ii). Now we have only to show (i), i.e., that U is a recursively enumerable set.[†14] Using the primitive recursive function K' of Theorem 3.16, we set $K''(t,x) := K'(t,x,L(x))$. Then $K''(t,x)$ is a primitive recursive function converging decreasingly to $K(x|L(x))$ as $t \to \infty$, in particular, for each x, there exists a $t_x > 0$ such that $K''(t,x) = K(x|L(x))$ if $t \ge t_x$. Hence, by a primitive recursive function

$$x \mathbin{\dot{-}} y := \begin{cases} x - y & (x > y), \\ 0 & (x \le y), \end{cases}$$

U can be written as

$$U = \{(m,x) \,|\, \exists t \text{ s.t. } K''(t,x) \mathbin{\dot{-}} (L(x) - m - 1) = 0\}, \tag{3.10}$$

which is recursively enumerable by Theorem 3.5(iv).

<u>*Step 2.*</u> The proof of $K(x|L(x)) \le L(x) - m(x) + c$: Let ϕ be a primitive recursive function which enumerates the universal test V (Theorem 3.5(ii)), i.e., $\phi(\mathbb{N}) = V$. Here we may assume that ϕ is injective. By using ϕ, define an algorithm $A : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ as follows. Let $\phi(i) =: (m_i, x_i) \in V$. First, define

$$A(\underbrace{0\ldots00}_{L(x_1)-m_1}, L(x_1)) := x_1.$$

Next, if $(m_1, L(x_1)) = (m_2, L(x_2))$, then define

$$A(\underbrace{0\ldots01}_{L(x_2)-m_2}, L(x_2)) := x_2, \tag{3.11}$$

and if $(m_1, L(x_1)) \ne (m_2, L(x_2))$, define

$$A(\underbrace{0\ldots00}_{L(x_2)-m_2}, L(x_2)) := x_2.$$

[†14] If $K(x|L(x))$ were a total recursive function, by Theorem 3.5(v), we would see that U is recursively enumerable (cf. (3.10) below). But a similar argument as the proof of Theorem 3.15 shows that $K(x|L(x))$ is not recursive, and hence we need the argument below.

In this way, continue to define A. Since V is a test, the number of (m, x)'s having the same value $(m, L(x))$ is at most $2^{L(x)-m}$, they can be described by programs with length less than $L(x) - m$ bits. Thus A is certainly well-defined. Then obviously,

$$K_A(x|L(x)) = L(x) - m(x),$$

from which $K(x|L(x)) \le L(x) - m(x) + c$ follows. □

The inequality (3.9) reads "When $L(x)$ is so large that c is negligible, $K(x|L(x))$ and $L(x)$ are close to each other, if and only if $m(x)$ is small." In a word, x is random, if and only if $K(x|L(x)) \approx L(x)$.

3.4 Kolmogorov complexity and entropy

Suppose that a random variable X has a discrete distribution

$$\Pr(X = a_i) = p_i, \quad i = 1, 2, \ldots, M, \quad (a_i \neq a_j, \text{ if } i \neq j).$$

Then

$$H(X) := -\sum_{i=1}^{M} p_i \log_2 p_i \ge 0$$

is called the *entropy* of X. For example, let us consider general (including "unfair") coin tossing process; let $\{X_i^{(p)}\}_{i=1}^n$, $0 < p < 1$, be an i.i.d. sequence whose common distribution is given by $\Pr(X_1^{(p)} = 1) = p$, $\Pr(X_1^{(p)} = 0) = q := 1 - p$. Then we see

$$\begin{aligned} H\left(\{X_i^{(p)}\}_{i=1}^n\right) &= -\sum_{x\in\{0,1\}^n} \Pr\left(\{X_i^{(p)}\}_{i=1}^n = x\right) \log_2 \Pr\left(\{X_i^{(p)}\}_{i=1}^n = x\right) \\ &= -\sum_{r=0}^{n} \Pr(S_n = r) \log_2 p^r q^{n-r}, \quad S_n := \sum_{i=1}^{n} X_i, \\ &= -\sum_{r=0}^{n} \Pr(S_n = r)(r \log_2 p + (n-r) \log_2 q) \\ &= -\mathbf{E}[S_n \log_2 p + (n - S_n) \log_2 q] \\ &= n(-p \log_2 p - q \log_2 q), \end{aligned}$$

where the last equality follows from $\mathbf{E}[S_n] = np$.[†15] In what follows in this section, we define

$$h(p) := -p \log_2 p - q \log_2 q.$$

$\{X_i^{(1/2)}\}_{i=1}^n$ is a "fair" coin tossing process, which is distributed uniformly in $\{0, 1\}^n$, and has the maximum entropy; for any p, it holds that

$$H\left(\{X_i^{(p)}\}_{i=1}^n\right) \le H\left(\{X_i^{(1/2)}\}_{i=1}^n\right) = nh(1/2) = n.$$

[†15]Note that the entropy of a stationary process is usually defined by the entropy $H\left(\{X_i^{(p)}\}_{i=1}^n\right)$ divided by the length n.

Roughly speaking, $H(X)$ indicates approximate length of the fair coin tossing process from which X is constructed (Macmillan's theorem, cf. [33]). On the other hand, the Kolmogorov complexity $K(x)$, $x \in \{0,1\}^*$, is the length of the shortest program that produces x. Kolmogorov's insight revealed the profound relation between them, and he himself called $K(x)$ the "entropy of x". The following lemma and theorem make their relation clear.

Lemma 3.21 *(cf. [54]) Suppose that $x = (x_1, x_2, \dots, x_n) \in \{0,1\}^n$ satisfies $\sum_{i=1}^n x_i = np$. Then there exists a constant $c > 0$ independent of n and x such that*

$$K(x) \le nh(p) + \frac{7}{2}\log_2 n + c.$$

Proof. The number of elements of $\{0,1\}^n$ satisfying the condition of the lemma is $\binom{n}{np}$. Among them, let x be the n_1-th element in the canonical order. Let A be an algorithm which produces x from n, np and n_1, i.e.,

$$A(\langle n, np, n_1\rangle, 0) = x.$$

Then we have

$$\begin{aligned} K_A(x) &= L(\langle n, np, n_1\rangle) \\ &= 2L(n) + 2 + 2L(np) + 2 + L(n_1) \\ &\le 2\lfloor \log_2(n+1)\rfloor + 2\lfloor \log_2(np+1)\rfloor + \left\lfloor \log_2\left(\binom{n}{np} + 1\right)\right\rfloor + 4. \end{aligned}$$

By Stirling's formula $n! \sim n^n e^{-n}\sqrt{2\pi n}$,

$$\binom{n}{np} = \frac{n!}{(np)!(nq)!} \sim \frac{1}{\sqrt{2\pi npq}} \cdot \frac{1}{p^{np}q^{nq}}, \quad n \gg 1,$$

i.e.,

$$\log_2\binom{n}{np} \approx -\frac{1}{2}(\log_2 2\pi pq + \log_2 n) - np\log_2 p - nq\log_2 q, \quad n \gg 1.$$

From these, it follows that for some $c' > 0$,

$$K_A(x) \le nh(p) + \frac{7}{2}\log_2 n + c'.$$

By Theorem 3.11, taking a new $c > c'$,

$$K(x) \le nh(p) + \frac{7}{2}\log_2 n + c.$$

□

Let $n \gg 1$. As was mentioned above, the number of all elements x satisfying the condition of Lemma 3.21 is $\binom{n}{np}$, which is about

$$nh(p) - \frac{1}{2}\log_2 n - \frac{1}{2}\log_2 2\pi pq$$

bit number. Consequently, by a similar argument as Theorem 3.13(ii), almost all elements x satisfying the condition of Lemma 3.21 satisfy

$$K(x) \geq nh(p) - \frac{1}{2}\log_2 n - \frac{1}{2}\log_2 2\pi pq - c'', \quad c'' > 0.$$

From this fact, Lemma 3.21, and the weak law of large numbers

$$\forall\,\varepsilon > 0, \quad \lim_{n\to\infty} \Pr\left(\left| \sum_{i=1}^{n} X_i^{(p)} - np \right| \geq n\varepsilon \right) = 0,$$

we can derive the following theorem.

Theorem 3.22 *(cf. [54] Proposition 5.1) Let* $0 < p < 1$*. For any* $\varepsilon > 0$*, we have*

$$\lim_{n\to\infty} \Pr\left(\left| \frac{K\left(\{X_i^{(p)}\}_{i=1}^{n}\right)}{n} - h(p) \right| \geq \varepsilon \right) = 0.$$

Remark 3.23 Let $n \gg 1$. Kolmogorov called $x \in \{0,1\}^n$ random if $K(x) \approx n$. But, as a matter of fact, those $x \in \{0,1\}^n$ such that $K(x) \approx rn$ $(0 < r < 1)$ cannot be chosen by anyone's will, either, and hence they may be called random. Therefore generic (typical) large samples of unfair coin tosses are random in this sense.

3.5 Infinite random sequence

Unlike the case of finite $\{0,1\}$-sequences, the definition of randomness for infinite $\{0,1\}$-sequences has no ambiguity. It is natural to think that an infinite $\{0,1\}$-sequence belonging to an event about the coin tossing process of probability 0 which is prescribed by a computable procedure, such as the exceptional set of the strong law of large numbers, is not a random sequence. Since those computable procedures are countably many, the union N of the events of probability 0 prescribed by those procedures is of probability 0. We call N the *maximal recursive null set*, and each element of N^c a *random sequence*.

Now let us make it clear what "a computable procedure which prescribes an event of probability 0" means. For each $y \in \{0,1\}^*$, let $C(y)$ be the (cylinder) set of all infinite $\{0,1\}$-sequences that begin with y, and let P_∞ be the distribution of the fair coin tossing process on $\{0,1\}^\infty$. An event $A \subset \{0,1\}^\infty$ satisfies $P_\infty(A) = 0$, if and only if, for any $m \in \mathbb{N}$, there exists a sequence $y_k^{(m)} \in \{0,1\}^*$, $k = 1,2,\ldots$, such that

$$U_m := \bigcup_k C(y_k^{(m)}) \supset A \tag{3.12}$$

$$\sum_k P_\infty\left(C(y_k^{(m)})\right) = \sum_k 2^{-L(y_k^{(m)})} < 2^{-m}. \tag{3.13}$$

Now set

$$U := \{(m,x) \in \mathbb{N}\times\{0,1\}^* \mid x \in U_m\}, \tag{3.14}$$

where, without loss of generality, we may assume that

$$(m,x) \in U, \quad n \leq m, \quad \text{and} \quad C(y) \subset C(x) \quad \Longrightarrow \quad (n,y) \in U. \tag{3.15}$$

Then, A is said to be a *recursive null set* if the U satisfying the conditions (3.12)~(3.15) is a recursively enumerable set. Since U_m can be regarded as a rejection region of significance level 2^{-m} of a test about infinite $\{0,1\}$-sequences, we call U a *sequential test*.

Applying the enumeration theorem in a similar way as § 3.3.1, we can show that there exists a sequential test V such that for any sequential test U, it holds that

$$\forall m \in \mathbb{N}^+, \quad U_{m+c} \subset V_m. \tag{3.16}$$

Here $c > 0$ is a constant depending on only U and V. We call V a *universal sequential test* ([26]). A universal sequential test determines a recursive null set, which is nothing but the maximal recursive null set N. There are more than one universal sequential tests, but N is uniquely determined.

Example 3.24 The infinite $\{0,1\}$-sequence $\{d_i(\pi-3)\}_{i=1}^{\infty}$, the dyadic digits of $\pi-3$, is not a random sequence. Indeed, there exists an algorithm which computes $\{d_i(\pi-3)\}_{i=1}^{n}$ for each n. By using this algorithm, we can show that the one point set consisting of only $\{d_i(\pi-3)\}_{i=1}^{n}$ is a recursive null set.

Remark 3.25 Martin-Löf [26] deals with randomness of infinite sequences under general computable probability measures including the distributions of unfair coin tossing processes.

Remark 3.26 For each infinite $\{0,1\}$-sequence, we cannot characterize its randomness by Kolmogorov complexity K. But slightly modifying it, we can characterize the randomness in terms of complexity. See [23] for details.

3.6 Random number and probability theory

In Kolmogorov's modern probability theory, a random quantity is expressed as a random variable X, which is a function defined on a probability space, say $(\{0,1\}^L, 2^{\{0,1\}^L}, P_L)$, i.e., $X : \{0,1\}^L \to \mathbb{R}$. We think X random by the following interpretation; an $\omega \in \{0,1\}^L$ is randomly chosen and as a result $X(\omega)$ becomes random. But in probability theory, we always deal with X as just a function and we never mind how ω is chosen. Since randomness does lie in the process how ω is chosen, probability theory does not mind what randomness is.

Suffering from Parkinson's disease in his later years, Kolmogorov devoted himself to the question "What is randomness?", which his probability theory had been avoiding. Finally, he answered the question, as we mentioned in this chapter, by establishing the notion of random number. Theoretically, we do not need the notion of random number in studying probability theory, but recognizing it brings us deep understanding of not only the Monte Carlo method but also probability theory itself. Let us explain it below.

According to Kolmogorov, studying randomness is equivalent to studying random number. The existence of random numbers becomes prominent, only when the sample space $\{0,1\}^L$ in question is huge. We therefore know that it is important to study the case $L \gg 1$. So, let us suppose $L \gg 1$. Then, we cannot choose any one of random numbers in $\{0,1\}^L$ of our own will, although they account for nearly all sequences. This means that assuming the uniform probability measure P_L implies that $\omega \in \{0,1\}^L$ is not assumed to be chosen by anyone's will, but by some method beyond man's will, i.e., at random. Thus the probability space $(\{0,1\}^L, 2^{\{0,1\}^L}, P_L)$ provides a framework to study randomness.

As Theorem 3.15 implies, it is difficult to get any knowledge about individual random numbers. But since random numbers account for nearly all sequences, it is a good idea to study characteristic properties that nearly all sequences share. Such properties have been minutely studied in probability theory — properties described in various limit theorems, such as law of large numbers, central limit theorems, etc. Probably, limit theorems are the only mathematical formulation that enables us to investigate randomness concretely. This explains why limit theorems are so much studied in probability theory.

What is really amazing is that since a long time before the discovery of the notion of random number, the probabilists of great insight had recognized the importance of limit theorems and had made a lot of efforts to study them.