23 Lecture 23. Randomness

23.1 Summary so far

Chaos, according to my definition, is characterized (defined) by its 'close relation' to randomness. This is not so illogical or unnatural, if we recall the relation between a Bernoulli system $B(1/2, 1/2)$ (one-sided) and the coin-tossing process. Indeed, this was the motivation for my characterization of chaos.

How 'good' is this characterization? Consistency with intuition, close connection to fundamental concepts, equivalence to definitions based on very different points of view, etc., may be signs of goodness. As long as we assume that "(apparent) random behavior is fundamentally important characteristic of chaos," consistency with intuition is captured. Randomness must be a fundamental concept, so the characterization has a close connection with a fundamental concept.[245]

What is then 'randomness'?

23.2 Random number table that is not so random

To have some 'feeling' about 'randomness', let us look at the random number table.

What is a random number table? Intuitively, it is a table on which numbers are arranged without any regularity. Whether there is regularity or not is checked actually by various statistical tests, so a random number table is a table tabulating a number sequence that is recognized to have no regularity, passing all the statistical tests.[246]

As an example, take the well-known Kitagawa random number table.[247] The Kitagawa table was constructed as follows:[248]

(i) On each page of the random number table of R. A. Fisher and F. Yates, *Statistical Tables for Biological, Agricultural and Medical Research* (Oliver and Boyd, 1938),[249]

[245]It is also important to check the consistency and relations with other definitions of chaos to confirm the naturalness of the definition. The comparison is already summarized in **22.13**.

[246]For example, see A. L. Rukhin, "Testing randomness: a suite of statistical procedures," Theory Probab. Appl., **45**, 111 (2001). The author's math mentor Professor H. Watanabe once said,"Random number is like God. Its existence might be admissible, but, if you are shown 'this is Him,' it is quite doubtful."

[247]See T. Kitagawa, *Statistical Inference* (Suisoku-Tokei Gaku) (Iwanami, 1958).

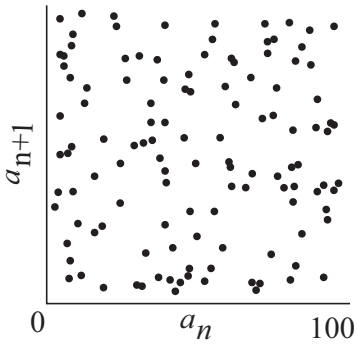
[248]in O. Miyataka and T. Nakayama, *Monte Carlo Method* (Nikkan-Kogyo, 1960)

[249]The Fisher-Yates table was constructed as follows: from the 20 digit logarithm table (of A. J. Thompson, *Logarithmetica Britannica: Logarithms to 20 Decimal Places 10,000-100,000* (This work of Dr. Thompson's is an attempt to commemorate in a worthy manner the first great table of common logarithms, which was computed by Henry Briggs and published in London in 1624.

two consecutive numbers are paired, and
(ii) The resultant 25 × 50 pairs were scrambled.
(iii) Then, the columns on different pages of the resultant scrambled table were exchanged.
(iv) Then, the following statistical tests were performed to each page, and the best 4 pages were kept:

(1) Frequency test: using the χ^2-test, to check whether all the digits appear equally frequently,
(2) 'Pair' test: the table is considered as the table of two consecutive number pairs, and the frequencies of the pairs were tested just as in (1).
(3) Poker test: the frequencies of the various patterns of the consecutive, e.g., 5 digit blocks (say, *abcde, abacd, aabac*, etc.) were tested.
(4) Gap test: Reading the table column-wisely, the spacing between consecutive identical digits were tested.

Is the Kitagawa table random? As illustrated in Fig. 23.1 it fails a simple test: numbers tend to change oscillatingly:



3	0	2	1	0	2	0	2	2	1
3	0	2	1	0	1	2	2	3	0
2	1	1	1	3	1	1	1	0	0
1	1	0	3	0	0	1	2	0	3
1	1	1	2	1	1	0	0	1	2
2	1	1	2	1	0	3	1	2	1
3	0	1	1	0	0	2	1	3	2
1	2	0	1	2	1	0	0	1	1
2	0	0	1	1	1	0	1	1	0
2	1	1	2	3	2	3	1	1	1

Figure 23.1: Shown on the Left is the first 5 rows of the Kitagawa table converted into a sequence of two digit number sequence $a_1, a_2, \cdots$, and then a_{n+1} is plotted against a_n. The square is divided into 100 square boxes of 10 × 10, and the numbers found in each square is counted as shown in the Right. If the sequence is random, the points must be distributed uniformly on the square, so in the right figure, we expect 79 points inside the square and 45 on the periphery. Actually, there are 65 inside and 59 outside, failing the uniformity test with the P value less than 0.5 %. In short, $\{a_n\}$ HAS a tendency to change in an oscillatory fashion.

Incidentally, it is said that, when a person is asked to write down as random a

It brings together the series of nine separate parts, issued between 1924 and 1952 from University College, London, in Karl Pearson's Tracts for Computers series; reprinted from Cambridge University Press, 2008), 15,000 digits were selected randomly, and then they were randomly arranged. However, the digit 6 appeared slightly more frequently than others, so 50 of them were randomly selected and replaced with other digits randomly.

number sequence as possible, the randomness of the produced number sequence and the intelligence of its writer are positively correlated.

The best random number is supposedly the natural random number that can be made by counting the number of, e.g., β-decay. Thus, we might say that without quantum phenomena there is no truly random numbers.

23.3 Featurelessness = randomness?

Statistical tests check the non-existence of particular patterns in the number sequence. The Kitagawa table was disqualified, because it has a significant (or detectable) pattern. Therefore, it seems to be a good idea to declare that the sequence without any feature (lawlessness) is a random number sequence. However, within our usual logical system that admits the exclusion of middle (i.e., there are only two possibilities A or non-A), 'that there is no feature' becomes a respectable feature;[250] we fall into an impasse that the random number sequence is a sequence with a characteristic that it has no characteristics.

von Mises (1883-1953) wanted to systematize probability theory based on randomness, but it was difficult because formalizing featurelessness or lawlessness was difficult. However, if we could positively characterize the feature that there is no feature, in other words, if we can define 'being without features' by an explicitly specifiable property, then 'there is no characteristic feature' is no more the negation of 'there are characteristic features.' Still, the characteristics such as 'lawlessness' or 'featurelessness" are ambiguous, allowing various interpretations.

23.4 Featurelessness = incompressibility?

Suppose we can find a feature (regularity) in a number sequence, we could save the phone charge by exploiting the regularity when we wish to send it to the second person. For example, if we wish to send $10101010\cdots10101010$ that has one million 10's, it is far better to send the message, "repeat 10 1,000,000 times," than to send the raw sequence itself. The number of digits required to describe a number N is asymptotically proportional to $\log N$, so such a regular sequence may be sent with the cost proportional to the logarithm of the original message length.[251] This is, of course, far more money-saving than the raw message.

---

[250] Of course, we must understand what 'features' mean.

[251] A student suggested that to send the number $\log N$, we could take its logarithm to compress it further and save money. Is it a good idea, or what is wrong?

Let us consider another example:

0273900749 7297363549 6453328886 9844061196 4961627734 4951827369 5588220757 3551766515
8985519098 6665393549 4810688732 0685990754 0792342402 3009259007 0173196036 2254756478
9406475483 4664776041 1463233905 6513433068 4495397907 0903023460 4614709616 9688688501
4083470405 4607429586 9913829668 2468185710 3188790652 8703665083 2431974404 7718556789
3482308943 1068287027 2280973624 8093996270 6074726455 3992539944 2808113736 9433887294
0630792615 9599546262 4629707062 5948455690 3471197299 6409089418 0595343932 5123623550

This sequence may look random, but it is the 480 digits starting from the 10,501st digit of π. If we wish to send this sequence, we can send a message, "the 480 digits starting from the 10,501st digit of π," and it is already shorter than the original message.[252] In this case as well, the message we must send is asymptotically proportional to the length of the part specifying the length of the sequence (the underlined part).

If we can compress the message, it is obviously non-random. Therefore, can we characterize the randomness of a message by the fact that it cannot be compressed (made shorter for communication) however we may try?

23.5 Algorithmic randomness

To compress a given sequence of symbols we must use its regularity and meaning, so whether we can recognize them or not is the key issue of the randomness of the sequence.[253] However, by whom should the regularities be recognized?

The basic idea of the algorithmic randomness due to Solomonov (1926-), Kolmogorov and Chaitin (1947-) is that this recognition should be done by the most powerful computer. Then, basic questions arise such as 'What is the most powerful computer?" and, in the first place, "What is a computer?"

A computer is a machine to perform computation. For this statement to make sense, we must know what 'computation' is.[254] Computation is to process a number into another number. The process of computation is not haphazard, but is understood to obey strictly certain rules. Therefore, we may say intuitively that

[252] However, the receiver of the message must perform a considerable procedure to obtain the message actually. Compressed information is often costly to expand. An interesting topic related to this is D. Bailey, P. Borwein and S. Plouffe, "On the rapid computation of various polylogarithmic constants," Math. Computat. **66**, 903 (1997) (`http://www.cecm.sfu.ca/ pborwein/`) and V. Adamchik and S. Wagon, "π: a 2000-year search changes direction," Mathematica in Education and Research, **5**(1) 11 (1996), D. H. Bailey, J. M. Borwein, P. B. Borwein, and S. Plouffe, "The quest for Pi," Math. Intelligencer **19**(1) 50 (1997).

[253] However, we do not discuss the vague concept called 'meaning.'

[254] What is a 'machine'?

computation is to transform one finite number sequence into another, using finitely many definite procedures.

23.6 Church's characterization of computation in plain terms

If we use the expression everyone understands by now, Church(1903-1995)'s proposal is essentially as follows. A function whose program can be accepted by a digital computer is a "partial recursive function,' and if the computation specified by the program is guaranteed to be completed within a finite time, the function is a 'recursive function' = computable function. Computation is a process to obtain the value of a computable function.

23.7 Number-theoretical functions

We consider only computation of a number with finitely many digits without any roundoff errors. Consequently, we can understand computation as a map: $\mathbb{N}^k \to \mathbb{N}$ ($k \in \mathbb{N}^+$). Such a map is called a number-theoretic function or arithmetic function. We consider only the computation of number-theoretic functions.

23.8 Church's basic idea

First, take a few functions that everyone intuitively accepts to be obviously computable as the starting point (see **23.10**). The totality of computable functions is constructed from these starting functions with a finite number of applications of the procedures that everyone agrees to be executable (see **23.11**).

To obtain values of the computable functions is called computation.

Thus, we need a characterization of 'computable functions.'[255]

23.9 Partial and total functions

In the theory of computation in contrast to the ordinary analysis, when we speak of a function, it need not be a map but can be a partial function. That is, $f(x_1, \cdots, x_n)$, where $x_1, \cdots, x_n$ are nonnegative integers (i.e., $x_i \in \mathbb{N}$), need not be meaningful (need not be defined) for all the n-tuples $\{x_1, \cdots, x_n\}$ of nonnegative integers (that is, the

[255] A classic introduction to the topic is M. Davis, *Computability and Unsolvability* (Dover, 1982). Newer textbooks include D. S. Bridges, *Computability, a mathematical sketchbook* (Springer, Graduate Texts in Mathematics 146, 1994), for example.

domain is not specified beforehand). For those tuples for which f is not defined, f is not evaluated. If f is defined on the totality of $\{x_1, \cdots, x_n\} \in \mathbb{N}^n$, it is called a total function.

23.10 Obviously computable functions

As the functions to start with, which everyone must agree to be computable, the following three functions S, P and C are adopted:
(A) $S(x) = x + 1$,
(B) $P_i^n(x_1, \cdots, x_n) = x_i$,
(C) $C_m^n(x_1, \cdots, x_n) = m$.
S is a function to give the successor of x in $\mathbb{N}$. P_i^n is a 'projection operator' to read the i-th variable out of n variables. C_m^n is a constant function assigning a constant m to all the n-tuples $\{x_1, \cdots, x_n\}$.

23.11 Basic operations

As unambiguous 'procedures' (basic operations) that everyone should agree to be applicable to any function let us accept the following I−III:
I Composition: From functions $g_1, \cdots, g_m$ and h we can make another function

$$f(x_1, \cdots, x_n) = h(g_1(x_1, \cdots, x_n), \cdots, g_m(x_1, \cdots, x_n)), \tag{23.1}$$

where h is an m-variable function and g_i $(i = 1, \cdots, m)$ are n-variable functions.
II (Primitive) recursion: Starting with $f(x_1, \cdots, x_n, 0) = g(x_1, \cdots, x_n)$, we can construct $f(x_1, \cdots, x_n, m)$ recursively as follows:

$$f(x_1, \cdots, x_n, m) = h(x_1, \cdots, x_n, m-1, f(x_1, \cdots, x_n, m-1)), \tag{23.2}$$

where g and h are, respectively, n and $n+2$ variable functions.
III Minimalization (or minimization) or unbounded search: Let $f(x_1, \cdots, x_n)$ be a total function. For each $\{x_1, \cdots, x_{n-1}\}$ we can determine the smallest x_n satisfying $f(x_1, \cdots, x_n) = 0$.[256]

23.12 Partial recursive function

A function that can be constructed from the basic functions (A)−(C) with a finite

[256] Here, it is crucial that f is a total function. Each step of the algorithm must be guaranteed to end within a finite number of steps, so f must be total.

number of applications of the basic procedures I–III is called a *partial recursive function*.

There is no problem with I being a computable procedure. II is the same. Perhaps it may be tedious, but applying finitely many steps patiently step by step can complete the procedure.[257].

However, Procedure III (minimalization) is tricky. Since f is a total function, for any $\{x_1, \cdots, x_{n-1}\}$ we can certainly evaluate $f(x_1, \cdots, x_{n-1}, m)$ for any m with a finitely many steps. Therefore, fixing $\{x_1, \cdots, x_{n-1}\}$, and putting m starting with 0 in the ascending order into f one by one, we can check whether $f(x_1, \cdots, x_{n-1}, m)$ is zero or not. If f becomes zero for the first time with $m = q$, then we define $h(x_1, \cdots, x_{n-1}) = q$. However, the existence of such a non-negative integer q is not known beforehand (in other words, we do not know beforehand whether $h(x_1, \cdots, x_{n-1})$ is a total function or not). Therefore, we cannot know beforehand whether the minimalization process even ends or not. Indeed, there is a way to check whether a given m is an answer or not however large it may be, but no one knows the upper bound of m such that if there is no answer up to the value there is really no answer.

Thus, a partial recursive function is a number-theoretical function which construction procedure can be described unambiguously (i.e., its algorithm is given). However, whether it can be actually computed (constructed) cannot be known beforehand (due to minimalization).[258]

Suppose we begin evaluating a partial recursive function for variable x. If we have not obtained the value after some computation, this may imply that the function is not defined for this x (because the minimalization step does not have a solution) or it is defined but we must be much more patient. We hesitate to declare such a function computable.

23.13 Recursive functions

The functions we can really compute must be such that not only its each computational step is explicitly and unambiguously specifiable, but also the whole computation is guaranteed to be completed with a finite number of steps. Such functions are called recursive functions. That is, total partial recursive functions are called

[257]Functions that can be constructed only with the aid of these two procedures are called primitive recursive functions.

[258]Informally (but actually in a not very inaccurate way), a function whose procedure to compute can be programmed on the usual digital computer is a partial recursive function. There is no guarantee that the program actually completes the computation and produces its value for all the inputs.

recursive functions. A recursive function is a function with an algorithm that is guaranteed to be completed with a finite number of steps for any (admissible) inputs.[259]

Note that there are only countably many recursive functions. All the recursive functions can be numbered as $f_1, f_2 \cdots$.

23.14 Computation, the Church thesis

Church defined 'computation' as a procedure to evaluate a recursive function.

That is, Church identifies the set of computable functions and the set of recursive functions. This proposal is called the Church thesis.[260] Impeccably unambiguous computation is possible only when the computational procedures are given purely syntactically (that is, given just as symbol sequences that do not require any interpretation). This is a sort of ultimate reductionism.

The crucial points of this proposal are that the algorithm is explicitly given and that the whole process is completed with a finite number of steps.

23.15 Church's proposal was not easily accepted

When Church's thesis was proposed, it was not immediately and generally accepted that being a recursive function is a convincing characterization of any computable function. The reason was that there was no clear feel for constructive procedures that may be explicitly written down; aren't there not recursive (that is, not I-III above) completely new types of algorithms with which different class of functions may become computable? Isn't the above proposal under the restriction of the era (i.e., the level of mathematics of the day)? Furthermore, since such an intuitively appealing concept as continuity requires, to be defined clearly, the axioms of the topological space, it is possible that apparently intuitively obvious basic procedures

[259] Continuing the above informal expression, we can say that a recursive function is a function which can be programmed on the usual digital computer, and the program produces a number (with sufficient but finite computational time and memory) for any (admissible) input.

[260] The definition here is consistent with M. Davis, *Computability and Unsolvability* (Dover, 1982). However, names and definitions are different in different books. In M. Li and P. Vitànyi, *An Introduction to Kolmogorov Complexity and Its Applications* (Springer, 1993) and J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison Wesley, 1979) Church's thesis is the proposal that partial recursive functions are computable. Bridges call them computable partial function. Davis call partial recursive functions as partially computable functions.

and basic functions may not be logically simple.[261]

Subsequently, various definitions of computability were proposed, but interestingly all the definitions were equivalent to Church's thesis. That is, it gained a certain signature of naturalness. Usually, this is the explanation of the relevant history, but actually, one of the reasons that Church confidently proposed his thesis was that various definitions were equivalent.[262] Still, as mentioned above, the oppositions could not be quenched.

However, the characterization of computability in terms of the Turing machine explained below (roughly speaking, a digital computer with indefinitely large memory capacity) silenced all the oppositions to Church's proposal. Church himself wrote that Turing computability did not require any preparation to relate constructibility and the existence of effective procedures in the ordinary sense. In short, for basic concepts clear consistency with our intuition is crucial, so the best characterization of basic concepts is often supplied by explicitly visualizable machineries.

23.16 Basic idea of Turing

If a computer is a machine that performs computation, to characterize computation, one should formalize/construct a machine that can do all the elementary steps 'mechanical calculation' can make. This is the basic idea of the Turing machine.[263]

> Turing chose the restrictions built in the basic mechanism/function of the Turing machine, taking account of the limits of our sensory organs and intelligence. Our sensory organs can distinguish only finitely many distinct signals, and the number of distinguishable states of our brain is also finite. The number of kinds and quantities of tasks our brain and effectors can perform are also finite. These conclusions may be reasonable, because even if the signals and states are continuous, they are meaningful only after 'quantized' in the actual noisy world, and because our brain is a finite object. In short, Turing conceived

[261] See R. Gandy, "The Confluence of Ideas in 1936," in *The Universal Turing Machine, a half-century survey* (Oxford University Press, 1988). Neither Gödel nor Post accepted the proposal. [1936 was full of disastrous events: remilitarization of the Rhineland, Spanish Civil War started, X'ian Incident; J. M. Keynes, *The general theory of employment, interest, and money*].

[262] The situation is explained in detail in W. Sieg, "Step by recursive step: Church's analysis of effective calculability," Bull. Symbolic Logic **3**, 154 (1997).

[263] C. Petzold, *The annotated Turing* (a guided tour through Alan Turing's historic paper on computability and the Turing machine) (Wiley, 2008) is strongly recommended. It includes historical comments on many related topics, many interesting anecdotes (and gossips). It is a serious book, but fun to read.

our brain as a finite automaton = an automaton that relies on finitely many symbols, finitely many rules and operations, and finitely many internal states.

Thus, he required an 'artificial brain' the following:
(i) Only finitely many kinds of symbols can appear on each cell (on the tape).
(ii) It can survey only a finitely many cells at once.
(iii) At each instance, it can rewrite only one cell.
(iv) Only finite range of the whole tape can be used (scanned).
(v) There are only finitely many states of the black box, and there are only finitely many instructions that can be performed.

The only idealization, from Turing's point of view, is that there is no memory capacity limit. However, since computers and our brains can indefinitely increase their external memory capacity, it is a benign idealization.

23.17 Turing machine[264]

A Turing machine consists of an infinite tape, a read/write head, and a black box with a finite number of internal states (Fig. 23.2). The tape is divided into cells. The read/write head can scan only one cell on the tape at a time. The head position and what it should do is specified by the Turing program. See **23.18**.

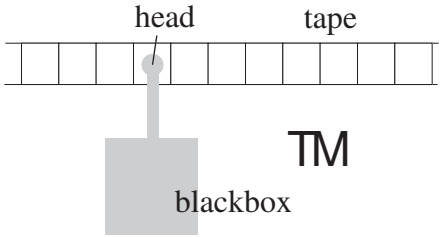


Figure 23.2: A Turing machine consists of an infinite tape that is divided into cells, a read/write head that can scan a single cell at a time, and a black box with a finite number of the internal states.

23.18 Turing program

A Turing program is a finite set of four tuples $(q, S, *, q')$, where
(1) q and q' are the internal states of the black box,

[264]There are many different versions, e.g., with many tapes or a tape that is only infinite in one direction, but the reader has only to be able to have a general notion of TM. See J. H. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison Wesley, 1979), for example. The exposition here uses the machine revised by Kleene and M. Davis (who was in Illinois Math department and is know for his contribution to Hilbert's tenth problem).

(2) S is the symbol in the cell being scanned now (it may be 1 or blank B, or 0, 1, B, etc., depending on authors, but in any case there are only finitely many of them), (3) $*$ denotes R, L or S': R (resp., L) implies that the head moves to the right (resp., to the left) in the next time step (or it may be better to say the tape is moved in the opposite direction); S' implies that the head does not move but rewrite the symbol in the cell being scanned as $S \to S'$.

The implication of $(q, S, *, q')$ is as follows:
If the internal state of the black box is q and if S is written in the tape cell being scanned by the head, then the head performs $*$, and the internal state of the black box is changed to q'.

A *Turing machine* may be identified with its Turing program.[265] That is, the machine is understood as a single task machine.

The input non-negative integer x is written on the tape according to a certain rule as, e.g., a 01 sequence (there are many different schemes, but here no details are important). The Turing machine (its black box) is initially in the *initial state* q_I, and the head is located at the leftmost non-blank cell on the tape (the portion where numbers are written is bounded).

The machine has a special final state called the *halting state* q_H. The sequence written on the tape when the machine state reaches q_H is interpreted as the computational result y of the Turing machine. Thus, the Turing machine defines a function $x \to y$. If the Turing machine halts for any input, it defines a total function. However, a Turing machine may not define a total function, because it may not halt (q_H may never be reached) for particular inputs.[266]

23.19 Turing computability

A function defined by a Turing machine that halts for any input is called a Turing computable function. The fundamental theorem is:

Theorem [Turing] Turing computable functions are computable functions in Church's sense and vice versa.

This theorem is proved through demonstrating that the three basic functions S, P and C and the fundamental operations I-III can be Turing-programmable. For

[265] To identify the program and the machine is not as simple as we think. The program lives in the world of symbols, which is distinct from the real world, so how to specify the correspondence between the symbols and the actual movements of the parts of the actual machine is nontrivial. We have already mentioned the concept called 'adaptors.' This will be discussed in Chapter 4.

[266] Will the Turing machine ever halt for input x? This is the famous *halting problem*. There is no algorithm to answer this question. This is a typical *decision problem* that cannot be decided.

example, Davis' textbook explains detailed construction of the programs.[267]

The equivalence of Church's computability and Turing's computability convinced many people the naturalness of Church's thesis.[268]

23.20 Universal Turing machine

Notice that a Turing machine can be encoded as a number sequence (i.e., we can make an integer that can be deciphered to give a Turing program[269], e.g., the Gödel numbering of the Turing program).[270] Now, it is possible to make a machine (= compiler) that spits the four-tuples of the Turing program upon reading the corresponding Gödel number. Then, if we subsequently feed the ordinary input to this master Turing machine, it can perform any computation that can be done by any Turing machine (it is a by-now common programmable digital computer).

A formal proof of this statement may not be simple, but we are not at all surprised thanks to our daily experience. This machine is called a *universal Turing machine*. We may understand it as an ideal digital computer without any memory capacity limit. Turing demonstrated that everything a machine with constraints (i)-(v) above

[267] Reading the programs to check their functions (to check that indeed they work) is a matter of patience; to write such a program is something like writing a program in a machine language (less efficient, actually), so for the ordinary scientists checking the demonstration is probably useless. Therefore, no further discussion will be given on the equivalence demonstration. Now that digital computers are everywhere, the theorem should be almost obvious.

[268] However, this does not mean the acceptance of Turing's basic idea, which seems to have motivated the formulation of Turing machines, that we are finite automata. Gödel (1906-1978) and Post (1897-1954) always believed that our mathematical intelligence was not mechanical. Especially, Gödel argued that our capability to manipulate abstract concepts is not restricted by the finiteness that Turing respected literally; the restrictions apply only when we manipulate (potentially) concrete objects such as symbol sequences; we had to take into account non-finitary creative thoughts to understand our mathematical capabilities.

⟪**Natural intelligence and finiteness constraints**⟫ Perhaps, Gödel may have wished to say that our natural intelligence is not restricted by the finiteness constraints. Our natural intelligence is 'embodied.' It is open to the external physical world through our body. It may be more sensible to idealize our brain as a nonfinitary system. Thus, after all, Gödel's intuition may well be correct. Indeed, abstract concepts are much more concretely supported than concrete concepts by the materialistic basis (or, we could even say, by the molecular biological basis) that is ancient phylogenetically. That is, abstract concepts are directly connected to our body. Abstract concepts are more embodied than concrete concepts.

[269] q, L, etc., appearing in the Turing program is encoded into numbers.

[270] ⟪**Gödel numbering**⟫ Assigning positive integers to symbols, we can convert any symbol sequence into a positive number sequence $n_1 n_2 \cdots n_k \cdots$. This is converted into $N(n_1 n_2 \cdots n_k \cdots) = 2^{n_1} 3^{n_2} \cdots p_k^{n_k} \cdots$, where p_k is the k-th prime. We can decipher n_i form N uniquely.

can perform can be done by a universal Turing machine. Thus, we may regard a universal Turing machine as the most powerful computer. If we accept Turing's analysis of our intelligence, any intelligent task we can do can be done by a universal Turing machine (with a suitable input).

A universal Turing machine is not a parallel computer. However, whether a machine is parallel or not is not fundamental. It is still a finite automaton. Therefore, the universal Turing machine being not parallel is an unimportant restriction. In the theory of computation we totally ignore computational speed; what matters is whether a required computation can be performed within a finite time.[271]

23.21 All the universal TM are equivalent

Since there are many ways to formulate Turing machines, universal Turing machines cannot be unique. We wish to have the most powerful computer, so perhaps we have to look for the most powerful universal Turing machine. Actually, all the universal Turing machines are equally powerful, so we may choose any of them for our purpose:

Theorem [Solomonov-Kolmogorov] Let M and M' be two universal Turing machines, and $\ell_M(x)$ (resp., $\ell_{M'}(x)$) be the length of the shortest program for M (resp., M') to produce output x (measured in, say, bits). Then, we have

$$\ell_M(x) \preceq \ell_{M'}(x),\ \ell_{M'}(x) \preceq \ell_M(x), \tag{23.3}$$

where $A(x) \preceq B(x)$ implies that there is a positive constant c independent of x such that $A(x) \leq B(x) + c$; c may depend on A and B.

The key to prove this theorem is that M can emulate M' and vice versa. For example, the program for M to emulate M' must be finite, however long it may be. Therefore, if we disregard the length of the program for this overhead (that is, if we write this length as c in the definition of $\preceq$), the length of the needed program does not change whether x is computed directly on M' or computed on the emulated M' on M. The meaning of $\preceq$ is just the inequality disregarding an additive constant (corresponding to the overhead), so the theorem should hold.

Thus, when we wish to write the shortest program length $\ell_M(x)$ (i.e., when we wish to compress x), we will not explicitly specify the universal Turing machine M to use and write simply $\ell(x)$.

[271] Quantum computers may drastically change the required time, but even they cannot compute Turing noncomputable functions. Thus, when we ask the fundamental question what computers can ever do, quantum computers need not be considered.

23.22 Program length, a preliminary observation

Let $\omega[n]$ be the first n digits of a binary sequence ω. How does $\ell(\omega[n])$ (see **23.21**) behave generally?

For example, for the uninteresting $1111\cdots$, asymptotically the information required to specify the number of digits n dominates the program, so $\ell(11\cdots[n])$ behaves as $\log n$. For a number sequence ω with an obvious regularity the shortest program to specify $\omega[n]$ is independent of n except for the portion needed to specify n itself. There are only countably many such regular sequences.

On the other hand, in the $n \to \infty$ limit if $\ell(\omega[n])/\log n$ can be indefinitely large, the pattern in the sequence cannot be specified asymptotically by a finite length program, so it should not be simple. However, such sequences include many subtle sequences.[272]

If no regularity is discernible at all in ω, to specify $\omega[n]$ requires to specify almost all the n digits, so we expect $\ell(\omega[n]) \sim n$. In other words, a typical random sequence must be such ω that for infinitely many n $\ell(\omega[n]) \sim n$.

23.23 Algorithmic randomness

Let us call a sequence ω such that for infinitely many n $\ell(\omega[n]) \sim n$ an algorithmically random sequences.

Definition 2.11.1

$$K(\omega) \equiv \limsup_{n\to\infty} \ell(\omega[n])/n \tag{23.4}$$

is called the randomness of a binary sequence $\omega \in \{0,1\}^{\mathbb{N}}$. Here, $\omega[n]$ is the first n digits of ω and $\ell(\omega[n])$ is the length of the shortest program (written in 01) to produce $\omega[n]$.[273]

Thanks to Kolmogorov and Solomonov (**23.21**) the randomness does not depend on the choice of the universal Turing machine M (so it is not written already).

Kolmogorov used the word 'complexity' to describe the above quantity. in this lecture notes the word should be reserved for genuine complex systems, so we use the word 'randomness' instead.[274]

[272] This situation is exactly the same as the KS entropy zero dynamical systems.

[273] The reason why we must use lim sup is, for example, we cannot ignore the appearance of meaningful sequences occasionally. That is why it was said $\ell(\omega[n]) \sim n$ for infinitely many n instead of all n.

[274] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*

23.24 Undecidability of random sequence

Notice that no program can be written to compute $K(\omega)$ (there is no algorithm for it). This should be easily inferred from the appearance of the expression as 'the shortest program' in the definition. For an arbitrarily chosen number sequence, except for almost trivial cases, it is hard to show that it is not a random sequence, because we must write down a short program to produce it. Worse still, it is harder to claim that a given sequence is random, because we must show that however hard one tries one cannot write a short program. Therefore, for a given number sequence, generally we cannot compute its randomness K (except for some almost trivial cases with $K = 0$, it is very hard, if not impossible.)

However, we can say something meaningful about a set of number sequences. For example, for almost all (with respect to the Lebesgue measure) numbers in $[0, 1]$ their binary expansion sequences are algorithmically random. All the algebraic numbers[275] are not random. In the next section we will discuss the average randomness.[276]

23.25 Is our characterization of randomness satisfactory?

To make the concept 'random' mathematical, we identified it with the lack of any computable regularity. This identification does not seem to contradict our intuition. However, is it really true that all the regularities are all those detectable by computers? There can be the following fundamental question: why can we say that if computers cannot detect any pattern, Nature herself cannot, either? In this characterization don't we admit that the Turing computer or computation itself is more fundamental a concept than randomness? Is this consistent with our intuition?

(Springer, 1993) is the standard textbook of the Kolmogorov complexity. The reader will realize that there are many kinds of definitions and concepts, but here, to be simple, the most basic definition is used. The explanation in this section roughly follows A. K. Zvonkin and L. A. Levine, "The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms," Russ. Math. Surveys **25**(6), 83 (1970).

[275] numbers that can be zeros of integer coefficient polynomials.

[276] There is an attempt to make a computable measure of randomness. One approach is to use much less powerful computers. However, such an approach appears to be fundamentally off the mark to characterize the concept of randomness, because 'randomness' may well be a transcendental concept. There can be a point of view that within our mathematics it is natural that we cannot tell whether a given sequence is random or not in general.

23.26 Axiomatic approach to randomness

If we consider randomness as a fundamental concept, as long as there is no other concepts that we can accept as more fundamental, we cannot define it. In the end, randomness would be formalized only as a primitive concept of an axiomatic system for randomness, just as points and lines in Euclidean geometry. As far as the author is aware, the most serious approach in this direction is due to van Lambalgen.[277] As we have seen, randomness is algorithmically characterized by incompressibility. We needed the theory of computation to define incompressibility unambiguously. The axiomatic approach may be roughly interpreted as an attempt to axiomatize the concept corresponding to 'incompressibility.'[278]

As we have seen in Chapter 1, randomness is really significant only in nonlinear systems. We have also seen that the standard axiomatic system of sets is a legitimate heir of Fourier analysis, so to speak. A wild and heretic guess is that even on the foundation of mathematics is a shadow of linear systems, so it is not very suitable to study nonlinear systems and the concept of randomness.

[277] M. van Lambalgen, "The axiomatization of randomness," J. Symbolic Logic **55**, 1143 (1990).

[278] This axiomatic system is not compatible with the standard axiomatic system of sets (ZFC). The author does not understand how serious this incompatibility is. See M. van Lambalgen, "Independence, randomness and the Axiom of Choice," J. Symbolic Logic **57**, 1274 (1992). See also "Logic: from foundations to applications, European logic colloquium" edited by W. Hodges, M. Hyland, and J. Truss (Clarendon Press, 1996) Chapter 12 "Independence structures in set theory."