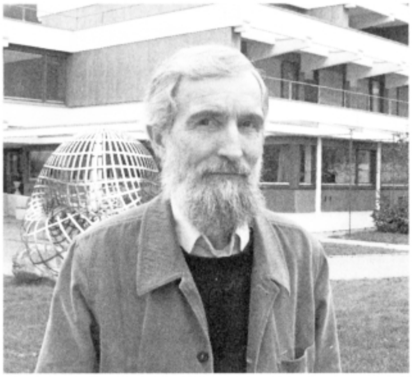

Per Martin-Löf

aus
Perci Diaconis & Brian Skyrms,
Ten great ideasb about chance.
Princeton University Press 2018

CHAPTER 8

ALGORITHMIC RANDOMNESS

Can computers generate random sequences? Random number–generating programs purport to do so. Can nature generate random sequences? Do we really have a clear idea what an objectively random sequence really is? You will remember that when we left von Mises theory in chapter 4, even at that highly idealized level, the theory of a random sequence was not theoretically satisfactory.

But random sequences are also important at a very practical level. In almost every walk of scientific life, simulations have a role to play, and these require sources of random numbers. So do secret codes used by spies, banks, and the Internet for secure communications and secure transactions. It turns out that often they are not sufficiently random, and then some bad science and bad security results.

The efforts to generate and test random numbers have deep philosophical tendrils. Our chapter begins at the practical end and then turns to logicians' efforts to define perfect randomness. The last section combines these concerns.

COMPUTER GENERATION OF RANDOM NUMBERS

Computers usually work with finite things, and we focus on an easy, widely used case: generating random numbers in the natural numbers $\{0, 1, 2, \ldots, N\}$. Thus one wants a sequence, $X(1), X(2), X(3), \ldots$, in this range that is *uniform* and *independent.*

The standard schemes are deterministic. They choose $X(1)$ "somehow"—by human input of the seed or using the number of milliseconds since the start of day—and proceed by

$$X(n+1) = f(X(n)),$$

with f being a fixed function. A classic example, RANDU, had $f(j) = 65{,}539 \cdot j \pmod{2^{31}}$. RANDU was a widely used random number generator in the 1960s, until it was shown that when used to plot random points in three dimensions—the coordinates of a point being three consecutive "random" numbers—the points clustered on planes. This is a decidedly nonrandom outcome! In 1968 George Marsaglia published a proof that all schemes in the same general class as RANDU would have the same defect.[1]

More sophisticated variants use higher order recursions. In an early "bible" of random number generation,[2] the author's favorite generator was $X_{n+1} = X_{n-24} \cdot X_{n-55} \pmod{2^{32}-1}$. This requires starting with seed $X_1, X_2, \ldots, X_{55}$. The most popular modern generator, Mersenne Twister,[3] uses a similar scheme.

Do these schemes work? Well, yes and no. For some tasks, such as computing integrals and playing computer games, they have usually done well. However, there is a long history of failures as well. In 1993, a New York Times article[4] recorded a failure of several new and improved random number generation schemes to solve routine instances of a statistical physics problem whose correct solution was known analytically. We know casino hustlers who capitalize on the fact that today's slot machines work with the very simple generators just described. By observing a few hundred pulls of the handle, they

figure out N and $f(i)$. If they know these and the current $X(n)$, they know $X(n+1)$, $X(n+2)$, and so on. They watch play until a big jackpot is due. Then they "accidentally" spill coffee on the current player and (after Oops, please take this \$50 chip for cleaning), they take over, play, and collect. Bank fraud and security break-ins on computer systems are reported daily.

If casinos, bankers, and the CIA can't get it right, this suggests that there might be a basic problem! Random numbers are usually tested by using a battery of ad hoc tests. (For instance, will high and low follow each other like coin tossing should? What about consecutive sets of 3, odd, and even, and so on?) These tests embody good common sense, but we must remember that random numbers are called on for very disparate tasks. Good for some tests does not mean good for others.

For Donald Knuth's random generator, the test that does it in is called the birthday spacings test. Get the generator to generate $X(1), X(2), \ldots, X(500)$, between 1 and 1,000,000. Order these (say, from smallest to largest). Consider the "spacings," largest–second largest, second largest–third largest, ... Look at the number of repeated values among the spacings. The approximate distribution for the number of repeats can be determined theoretically, and Knuth's generator produced numbers 16 times too large. There are some tests with some claim to generality: if you pass this test, you will pass a whole slew of other tests. (One is called the *spectral test.*) But these are far too limited to capture the vast range of applications.

It is natural to ask for Nature's help. After all, quantum mechanics and thermal noise are supposed to be truly random. We consulted with a wonderful group of physicists who wanted to put a huge supply of good random bits on a disk at the end of their book.[5] Here is what they wound up doing. They started with "electrical noise" measured via the delay times in a leaky capacitor. This generated a long series of random binary digits. Testing showed that these were not so random. You could see periodic fluctuations, which were traced to the 24-hour variations in the power supply! A thousand such strings were generated. They were combined into one string by taking the sum (mod 2) of the thousand binary digits in each position. This final string was then scrambled using the data encryption standard. These were the final digits used.

A host of other techniques have been proposed. These range from using the quantum mechanical fluctuations in the clicks of a Geiger counter to using a lava lamp. Such schemes can be combined with each other and with the deterministic mathematical generators described before. There are no theoretical guarantees, and it all seems terribly ad hoc for such an important enterprise.

One hopeful development is the use of the logic of complexity theory, as in the scheme of Blum and Michali.[6] These authors offer a generator with the following property: if it fails any polynomial time test, then there is an explicit way to factor that is much faster than any known method. (Thus, if factoring is hard, our numbers are secure. But, of course, if factoring can be done efficiently, all bets are off.) This is close in spirit to the algorithmic complexity accounts treated subsequently in this chapter.

We conclude this practical section with some practical advice on the use of random number generators:

Use at least two generators and compare results. (We recommend Mersenne Twister and one of the generators offered in *Numerical Recipes*.)

Put in a problem with a theoretically known answer to run along with the other simulations.

Think of using random number generators like driving a car. Done with care, it is relatively safe and useful.

ALGORITHMIC RANDOMNESS

The algorithmic theory of randomness, our eighth great idea, reached its modern form with Per Martin-Löf's 1966 "The Definition of Random Sequences."[7] The concept of an objectively random sequence—one that is perfectly random—is made precise using the theory of computation.

As we saw in chapter 4, the problem that the algorithmic theory of randomness solves was put forward by Richard von Mises in 1919. Von Mises wanted to ground the application of probability to reality by putting forward an idealized mathematical model of random phenomena. We can see this as an attempt to sidestep the fallacies that small-probability events don't happen. Instead of starting with the random process of coin flipping and arguing that it is "morally certain" that the resulting random sequence would have certain properties, von Mises wants to give the theory of a random sequence directly.

Let's recall where we left his program: (1) A random sequence of 0s and 1s should have a limiting relative frequency, and (2) limiting relative frequency should be the same for any infinite subsequence selected out by an admissible place-selection function. (Admissible is left to be defined.) For example, consider the sequence of alternating 1s and 0s:

1010101010101010101010101010101010 . . . .

The limiting relative frequency of 1s is $\frac{1}{2}$. But the place-selection function that selects odd members of the sequence gives

11111111111111111111111111111111111 . . . ,

and that which selects even members gives

000000000000000000000000000000000000s . . .

for limiting relative frequencies of 1 and 0, respectively.

Do von Mises random sequences exist? That depends on the class of admissible place-selection functions. Too few place-selection functions give patently nonrandom sequences. For an extreme example, suppose we have just the preceding two place-selection functions. Then the sequence

11001100110011001100 . . .

would count as random, since each of the two place-selection functions selects a sequence

1010101010101010 . . . ,

which has a limiting frequency $\frac{1}{2}$, just as the original sequence. (Notice that in these examples, the limiting relative frequency of 1s is approached from above, e.g., 1. $\frac{1}{2}, \frac{2}{3}, \frac{1}{2}, \frac{3}{5}, \frac{1}{2}, \frac{4}{7}, \ldots$)

But you can easily think of an additional place-selection function that will change the relative frequency, for instance, pick every fourth entry. And you could then easily construct a sequence that is random according to the expanded class of place selections.

This raises a general question. What if there are a lot of place-selection functions? Can we always cook up a sequence that is random according to all of them? The answer depends on just what you mean by "a lot."

Critics of von Mises[8] were quick to point out that if *all* functions, in the set-theoretic sense, are included, then there are simply no random sequences. Abraham Wald[9] (who thought of functions not as sets but rather as rules that could be explicitly described) proved that given any countably infinite set of place-selection functions, one can indeed cook up von Mises random sequences relative to that class of place-selection functions. The question was left hanging as to whether there was some natural set of functions to use.

An answer was suggested by the theory of computability, but this was possible only after that theory had been developed by Turing, Gödel, Church, and Kleene in the 1930s. The idea of applying computability to von Mises' definition of a random sequence was due to Alonzo Church in 1940.[10]

Church suggested taking the admissible place selection functions as the computable ones—ones that could be implemented by a Turing machine.

This seemed like a natural choice, and since there are a countable number of Turing machines and a noncountable number of sequences, there are plenty of sequences that are random in this sense.

Unfortunately, the definition has a flaw. Von Mises–Church random sequences lack some of the properties that they should have. In particular, some sequences that are random in this sense approach their limiting relative frequency from one side. This means that they are vulnerable to a gambling strategy. Von Mises had considered the impossibility of a successful gambling system the sine qua non of a random sequence:

> By generalizing the experience of the gambling banks, deducing from it the Principle of the Impossibility of a Gambling System, and including this principle in the foundation of the theory of probability, we proceed in the same way as the physicists did in the case of the (conservation of) energy principle.[11]

Such violating sequences could hardly be taken as paradigms for outcomes of fair coin flips.

In a deeper sense, Church's idea of using computability to define randomness was correct. The problem was the way in which it was used. In fact, the vulnerability to betting systems has nothing to do with computability. Ville, in 1939,[12] showed that for *any* countable set of place-selection functions, there is a von Mises random sequence in which relative frequency of H is $\frac{1}{2}$, but for all but a finite number of initial segments, the relative frequency of H is not less than $\frac{1}{2}$. (Like our previous examples: 10101010 . . . and 110011001100. . . . In fact, the random sequences constructed by Wald to prove the consistency of von Mises' definition all had this property.) The set of place-selection functions proposed by Church is countable, so it inherits the problem.

The source of the problem is not the idea of using computability. Rather, von Mises' idea of defining randomness by means of place-selection functions *alone* appears to be defective.

Church–von Mises randomness requires random sequences to pass only one sort of test for randomness. Sequences can pass this test and fail others. We want random sequences to pass all tests of randomness, with tests being computationally implemented.

Per Martin-Löf found how to do this in 1966.[13] We shall see that two other, apparently different, ways of incorporating computability in a definition of randomness gave definitions that turned out equivalent to that given by Martin-Löf.

COMPUTABILITY

If you know how to program in any computer language, you already know what computability is. They all allow you to compute the same functions, although the required program may be shorter in some languages than others. Nevertheless, we include a brief account of the birth of computability theory because it provides a case study in the robust mathematical explication of a philosophical notion.

What is a computation? It is a philosophical question that goes back at least to Hobbes and Leibniz. Hobbes maintained that all thought was a kind of calculation—an idea that had a second life in the artificial intelligence community in the late twentieth century. Leibniz envisioned a universal formal language of thought and a system of rules of valid inferences, which together reduce any truth step by step to an identity. Empirical truths would require an infinite number of steps, recapitulating God's reasoning in deciding whether to create this world rather than another. But mathematical truths required only a finite number of steps, so in principle any mathematical question could be settled by logical analysis. To this end Leibniz worked both on formal logic and on the invention and construction of a calculating machine.

Leibniz' program, sans theology, lived into the twentieth century in the views of Bertrand Russell and David Hilbert. Russell held that all mathematics was reducible to logic. Hilbert thought that every mathematical problem could be decided. Hilbert put a sharp-enough point on the problem of computation to motivate its solution, in his statement of the *Entscheidungsproblem*: "Is there an algorithm that takes as input a mathematical statement and outputs 1 if it is true and 0 if it is not?" Hilbert and Ackermann (in 1928)[14] specifically ask the question for first-order logic (logic with individuals, predicates, and quantifiers *all* and *some* over individual variables): "The Entscheidungsproblem is solved when we know a procedure that allows for any given logical expression to decide by finitely many operations its validity or satisfiability." Using ideas pioneered by Kurt Gödel, a negative answer was found almost simultaneously by Alonzo Church and by Alan Turing. Leibniz was wrong!

The analysis required, at the onset, a precise theory of computation. A number of rather different ideas were put forward.

TURING COMPUTABILITY

Computers, at the beginning of the twentieth century, were people. They sat at desks and carried out computations according to instructions,

using paper and pencil. Alan Turing gave an abstract version of this process allowing unlimited paper and thus produced the very simple and intuitive notion of a Turing machine. Think of the pieces of paper available for computation strung together as a tape, infinite in both directions. Each piece of paper is called a cell of the tape. At any time, the head of the machine is over some cell, scanning what is written in the cell—which can either be a 0, 1 or B (blank). There is a special cell designated as the starting cell. There is a finite set of internal states, including a special starting state. Depending on the symbol scanned and the internal state of the machine, the machine performs an operation, either

Write 0, 1, or B in the cell it is scanning.
Move one cell to the left or to the right.

Alan Turing

When the action is taken, the machine takes on a (possibly) new state. Thus the dynamics of the machine in discrete time is characterized by a set of quadruples:

<current state, symbol scanned, operation, new state>,

There are no two quadruples with the same initial pair; the dynamics is deterministic. If the machine is in a state, scanning a symbol, such that there is no corresponding quadruple in its instruction set, the machine *halts*.

Here is an example. The machine starts with an empty tape, full of blanks. It is in its start state, which we will call S_0. It prints a zero and enters state S_1. This is accomplished by the quadruple

$$<S_0, B, 0, S_1>.$$

Now it is in state S_1, scanning the 0 that it just printed. The following instruction tells it to move to the right one cell and assume a new state, S_2:

$$<S_1, 0, R, S_2>.$$

Now it is again scanning a blank, but in state S_2. The next instruction tells it to print a 1 in that blank and assume state 3:

$$<S_2, B, 1, S_3>.$$

It is now in state S_3, scanning a 1. Our final instruction tells it to move to the right and go back to state S_0.

$$<S_3, 1, R, S_0>.$$

It is now in state S_0, scanning a blank, just as it started; these four instructions define a Turing machine that prints out the infinite sequence

01010101010101. . . .

Everything is finite about the machine (states, symbols, and instruction set), with the exception of unlimited tape. For a machine that computes the value of a function, the tape is assumed to start with the head over the leftmost cell of encoded values of the arguments. The machine halts with the head over the encoded value of a function. The machine may not halt for certain inputs, and in this case the function is a partial function.

For validity of first-order logic to be *decidable*, there would have to be a Turing machine that, when input a suitably encoded version of a formula of first-order logic, would output a 1 if the formula is valid and a 0 if not. A (nonempty) set is *computably enumerable* if it is the range of a total computable function. That is to say that there

is a Turing machine that when given inputs 1, 2, 3, . . . would list the members of the set. Although Gödel showed (by his completeness theorem) that the valid formulas of first-order logic are computably enumerable, Turing and Church showed that that validity is not decidable.[15] The *Entscheidungsproblem* for first-order logic is unsolvable.

Turing did this via another unsolvability result. First, he showed that it was possible to build a *universal Turing machine*—one that could emulate every other Turing machine if fed the appropriate input. Then he asked whether there was a Turing machine that decides the halting problem for all Turing machines. Is there, that is to say, a machine that when fed in the description of an arbitrary machine and an input for that machine outputs a 1 if the target machine will halt with that input and a 0 if not? He showed that the supposition that there is a machine that decides the halting problem leads to a contradiction. If there were such a machine, it would be possible to construct another that halts if and only if it doesn't. This undecidability of the halting problem leads to a proof of the undecidability of the decision problem for first-order logic.

RECURSIVE FUNCTIONS

Church took a different route to computability. His first approach was through his λ-calculus, which is the basis for the programming language LISP. Later he and his student Stephen Kleene, following the lead of Gödel, used recursive functions.

The *primitive recursive functions* are gotten from *zero, successor, and projection* functions, by closure under *composition* of functions, and *primitive recursion*:

The *zero* functions are $f(x_1, \ldots, x_k) = 0$.

Successor $S(x) = x + 1$,

Projection $I_n^k(x_1, \ldots, x_k) = x_n$.

Composition of f and $g_1, \ldots, g_n$, (bold **x** is a vector):

$$h(\mathbf{x}) = f(g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})).$$

Primitive recursion from f and g:

$$h(\mathbf{x}, 0) = f(\mathbf{x}),$$
$$h(\mathbf{x}, t+1) = g(t, h(\mathbf{x}, t), \mathbf{x}).$$

The *general recursive partial functions* are gotten from the primitive recursive functions by closing under a minimization operator, μ, that gives the smallest argument that gives the function a value of 0, if there is one. Where there isn't, the function is undefined. That is,

$$h(\mathbf{x}) = \mu y(g(\mathbf{x}, y) = 0)$$

is the smallest solution to the equation $g(\mathbf{x}, y) = 0$ if there is one and is undefined for values $\mathbf{x}$ where there is no smallest solution.

Turing proved that the general recursive partial functions are the Turing computable functions. They are undefined for inputs for which the machine doesn't halt. Two rather different ideas lead to the same place.

PROGRAMING LANGUAGES AGAIN

General recursive functions are also the functions that can be programmed in any modern computer language (size restrictions removed). If the programming language is restricted so that only bounded loops are possible, then it can compute only primitive recursive functions. With unbounded loops we get only partial functions because the program may not halt.

A ROBUST EXPLICATION OF COMPUTABILITY

All sorts of other approaches, including Markov algorithms,[16] Church's λ-calculus, much fancier Turing machines and register machines, computers with random access memory, and so forth, have been shown to lead to the same class of computable functions. This lends confidence to the intuition that this is the "right" notion of computability. In 1946 Kurt Gödel wrote, ". . . with this concept one has for the first time succeeded in giving an absolute notion to an interesting epistemological notion, i.e., one not depending on the formalism chosen."[17]

Kurt Gödel

RANDOMNESS

Since Church–von Mises randomness proved inadequate, Martin-Löf took a different approach. One could define random sequences by throwing out "atypical" classes—*null* classes—that were given probability 0 by a model of flipping a fair coin. The immediate problem is that each individual sequence has probability 0. Here computability comes to the rescue. We throw out only null sets that can be identified by a Turing machine. Since there are a countable number of Turing machines, throwing out all these null sets leaves a set of "typical" random sequences with probability measure one. (This again can be regarded as a way of avoiding the absurdities, discussed in chapter 4, of a literal application of Cournot's principle.)

The nonrandom sequences can be thought of as those failing more and more stringent statistical tests. Suppose we see reports of coin tossing:

010101.

It is a little suspicious. Suppose the sequence continues:

010101010101010101010101010.

That is a lot more suspicious. It is a lot less likely that fair coin flips should produce this sequence. (All the practical tests of computer programs to generate random numbers, with which we began this chapter, are of this character. They ask, in different ways, how likely it is that a sequence of coin flips could have produced this sequence.) Martin-Löf tests of randomness are a computable implementation of increasingly stringent tests of randomness. Here is how it goes.

Consider the set of infinite sequences of zeros and ones that continues some finite initial segment. These are called cylinder sets. Considered as sequences of tosses of a fair coin, any set corresponding to an initial segment of length n has probability $\left(\frac{1}{2}\right)^n$. A countable union of disjoint cylinder sets has a probability that is the sum of the probabilities of those sets.

Martin-Löf uses computability twice:

1. We restrict our attention to the case of unions of *computably enumerable* sequences of cylinder sets. That is, one can have a Turing machine that prints out characterizations of these, one after another. Call these the *effective sets*.
2. Less and less likely effective sets are intuitively less and less typical of a random sequence. One hundred alternations of heads and tails are quite suspicious. We can use a nested sequence of such sets that are less and less probable to approximate a definitely nonrandom set. That is, we consider a sequence $U_1, \ldots$ such that U_{n+1} is a subset of U_n, and the probability of U_n is at most $\left(\frac{1}{2}\right)^n$. We require that this sequence be *computably enumerable*. Such a sequence is a *Martin-Löf test*. It enumerates more and more stringent tests.

A set that is in the intersection of such a sequence is a *constructive nullset*. It fails the Martin-Löf test of randomness. A Martin-Löf random sequence is then defined as one that passes *all* the tests—one that is in no constructive null set.[18]

We can illustrate this by showing how our infinite sequence of alternating 0s and 1s fails to be random. The reasoning applies equally well to any computably enumerable sequence. Consider a cylinder set determined by the first n members of the sequence. This is an especially simple *effective set*.

The sequence of all cylinder sets determined by longer and longer initial segments of our original sequence:

all continuations of 0,
all continuations of 01,
all continuations of 010,
all continuations of 0101,

and so on, is thus a Martin-Löf test. The intersection of these cylinder sets is a Martin-Löf nullset whose only member is our original sequence, so it is not random.

There are a countable number of constructive nullsets, and each has probability zero, so the probability that a sequence is Martin-Löf random is equal to 1. Martin-Löf shows that each sequence that is random in his sense has a limiting relative frequency—a property that von Mises had to postulate separately for Kollektivs. This extends and unifies the connection between relative frequency and chance.

COMPUTABLE MARTINGALES

We have seen that von Mises considered the impossibility of a successful gambling system essential to the definition of a random sequence. This was his justification of the use of place-selection functions in his definition of randomness. You should not be able to make fair bets on a subsequence of a random sequence in a systematic way and make money. But Jean Ville had shown that the definition in terms of place selection is too restrictive. Some sequences that are random in the sense of Mises-Church are vulnerable to a gambling system—but not one so simple as to rely on computable place selection.

The idea of a gambling system for a sequence is made precise in the notion of a *martingale*. Suppose that you start with unit capital. You put some proportion of your capital in a bet that the first member of the sequence is a 1 or 0. If you stake everything on 1 and it is 1, you win and your capital is now doubled. Now you proceed, allocating proportions of your current stake on bets on the next member according to a rule that can consider the whole history of the sequence up to the point of the bet. The gambling strategy succeeds against a sequence if your stake grows to infinity.

Instead of explicitly writing down the strategy, the martingale can be characterized by the stake in hand at every point in the sequence.

So viewed, a martingale is a function, CAP, from initial segments to nonnegative reals, such that

$$\mathrm{CAP}(s) = \tfrac{1}{2}\,[\mathrm{CAP}(s \text{ followed by } 0) + \mathrm{CAP}(s \text{ followed by } 1)]$$

since the odds are fair. A martingale *succeeds* on a sequence if CAP goes to infinity in the limit.

For example, consider a betting strategy that puts all the capital on 1 if the last event is 0 and all the capital on 0 if the last event is 1. The resulting capital function is a martingale. It succeeds on the sequence

010101. . . .

It does not succeed on 00110011. . . . The strategy immediately goes bust on this sequence; CAP(00) = 0. But another martingale will succeed on this sequence.

It remains to impose computability constraints. The martingale is required to be computably enumerable. (The martingale is, in general, a function to real numbers, so computability is imposed by approximation. It is said to be computably enumerable if it is *left-computably enumerable*. That is to say, it is approximable from below by rational-valued functions.) Then one can define a random sequence in terms of impossibility of a gambling system. A sequence is random if no computably enumerable sequence succeeds in it. In 1971 Schnorr[19] proved that a sequence is random in this sense if and only if it is Martin-Löf random.

KOLMOGOROV COMPLEXITY

In the 1960s Kolmogorov returned to the foundations of probability from a new standpoint.[20] A finite sequence of 1000 alternating 0s and 1s is *algorithmically compressible* in that one can write a short program to generate it. This is because of its simple structure. Absence of structure is a way to define randomness. One can then measure randomness of a finite sequence, relative to some universal Turing machine, by measuring the length of the shortest program that will generate it. The shorter the program, the less random the sequence. Kolmogorov complexity was introduced independently by Gregory Chaitin in 1966.[21]

But the degree of randomness, thus defined, depends on the universal Turing machine (or the programming language) chosen. The dependence of the result on the universal Turing machine is limited, however, since any such machine can be programmed to simulate any other. The length of the simulation program is a constant, and length of shortest programs to generate a sequence must agree up to this constant. The differences may wash out at infinity (if everything goes well), but plausible application to finite sequences appears to depend on a plausible natural choice of a universal Turing machine.

In this approach to randomness or computational complexity, Kolmogorov and Chaitin were anticipated by Ray Solomonoff.[22] Solomonoff was partly inspired by a class on inductive logic taught by the philosopher Rudolf Carnap, which he sat in as an undergraduate at the University of Chicago. He thought that Carnap's project was good but that he was going about it in the wrong way. Solomonoff's idea was to use computational complexity in the construction of a universal prior.*

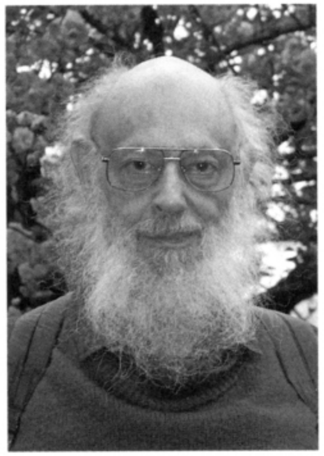
Since computational complexity depends on the choice of a Turing machine, or programing language, used to measure it, so does the universal prior. Some discussions try to slide by this by noting that one universal Turing machine can emulate any other, given the appropriate code. But the appropriate code may be long, and the difference in complexity may be large. Solomonoff, however, saw this not as a problem to be minimized, but just as a fact of life. The choice is subjective; it is a judgment based on our general experience. It is the choice of a prior, which is then updated by experimental results.

Solomonoff is unabashedly a subjective Bayesian:

> Subjectivity in science has usually been regarded as Evil—that is something that does not occur in "true science" that if it does occur, the results are not "science" at all. The great statistician, R. A. Fisher, was of this opinion. He wanted to make statistics a "true science" free of all the subjectivity that had been so much a part of its history.
>
> I feel that Fisher was seriously wrong in this matter. . . .

*The idea is spectacularly successful, provided that God uses a Turing machine to generate the universe.

In ALP (algorithmic probability), this subjectivity occurs in the choice of "reference"—a universal computer or universal computer language.[23]



Ray Solomonoff

It seems natural to apply ideas of computational complexity to von Mises' problem of defining a Kollektiv by passing to the limit. A Kollektiv would be an infinite sequence with incompressible initial segments. Applying computational complexity, as introduced above—*plain Kolmogorov* computational complexity—infinite random sequences do not exist!

Although the initial attempt to define a random infinite sequence in terms of Kolmogorov complexity failed, this turned out to be because the issue was not framed in exactly the right way. That is because a program (input) for a universal Turing machine contains both information about the sequence to be computed and also information about the length of the program. For the algorithmic complexity of the sequence, we are interested only in information about the sequence being computed. This can be achieved by restricting attention to a special kind of universal Turing machine—a *prefix-free* universal Turing machine. We measure the prefix-free complexity of a sequence K using a prefix-free universal Turing machine.[24, 25]

Then, at infinity, everything works. An infinite sequence is algorithmically random if there is a constant c such that complexity K of every initial segment of length n is greater than or equal to $n-c$. Algorithmically random infinite sequences now exist, and Schnorr (1971) also proved that these random sequences also coincide with the Martin-Löf random sequences.

There is thus one very strong and reasonably robust concept of algorithmic randomness. As Martin-Löf remarked, it gives a correct definition of von-Mises' idea of a Kollektiv. The frequentist ideas of von Mises are not so opposed to the measure-theoretic framework of Kolmogorov after all. *Flipping a fair coin infinitely often produces a Kollektiv with probability 1.*

VARIATIONS ON RANDOMNESS

If randomness is based on computation, then variations on randomness can be based on different computational notions. The theory can be pushed into higher realms of abstraction by equipping a Turing machine with an oracle. An oracle is a black box that will answer any of a certain class of questions that the Turing machine can put to it at any point in a computation. One can consider a universal Turing machine equipped with an oracle that decides the halting problem for any Turing machine. Such a machine is computationally more powerful than any Turing machine. Of course, such a machine does not decide the halting problem for Turing machines with such an oracle because that would lead to a contradiction.

One can then consider a super-oracle that decides the halting problem for these machines, and so forth, creating a hierarchy of computationally more powerful machines. This leads to a hierarchy of more and more stringent criteria for a random sequences, with Martin-Löf randomness being 1-randomness, the same notion with a Turing machine equipped with an oracle deciding the halting problem being 2-randomness, and so on. The class of $N+1$ random sequences is strictly contained in the class of N-random sequences.

In the other direction, weaker kinds of randomness can be gotten by limiting the kind of computation used to test for randomness.

This can be done in various ways. Schnorr proposed computable tests rather than computably enumerable tests, which gives a weaker notion of randomness. There is a theory of *P-randomness* using computability in polynomial time that parallels the theory discussed above. P-random sequences still possess many of the features one would want from a random sequence. Limiting computational resources in various ways gives weaker notions of randomness. If we are willing to fix on a programing language that we find natural, then Kolmogorov complexity gives us a usable measure for finite sequences.

SUMMING UP

The quest for the notion of an objectively random sequence had its origins in von Mises' attempt to formulate a pure frequency theory of probability based on an objectively disordered random sequence—his Kollektiv. This, most of us would agree, is a failed foundational program. The modern theory of algorithmic randomness proceeds within the Kolmogorov framework rather than being an alternative.

Now that Martin-Löf and others have developed a satisfactory concept of such a sequence via computability theory, we can ask what remains of von Mises' program. Returning to Borel (chapter 5), we can now say that Bernoulli trials—coin flipping—produce a von Mises Kollektiv with probability one. Then, due to de Finetti (chapter 7), exchangeable degrees of belief are equivalent to belief in a von Mises Kollektiv with uncertain frequency.