Algorithmic Randomness

Cristian S. Calude

(with some changes by Andre Nies, 2010)

- Understand a new type of complexity
- Understand algorithmic randomness from three points of view: computational, physical, and meta-physical
- Discuss open questions.

- Simulation and universality
- Program-size complexity
- Omega numbers
- Algorithmic randomness
- Uncertainty and incompleteness
- Quantum randomness
- Probabilistic computation
- Open questions

- C. S. Calude. *Information and Randomness – An Algorithmic Perspective*, Springer-Verlag, 2002.
- G. J. Chaitin. *From Philosophy to Program Size*, 8th Estonian Winter School in Computer Science, Institute of Cybernetics, Tallinn, 2003.
- G. J. Chaitin. *Meta Math!*, Pantheon, 2005.
- M. Sipser. *Introduction to the Theory of Computation*, PWS 1997.

- C. S. Calude. Algorithmic randomness, quantum physics, and incompleteness, in M. Margenstern (ed.). *Proc. Conf. "Machines, Computations and Universality"*, Lectures Notes in Comput. Sci. 3354, Springer, Berlin, 2005, 1–17.
- C. S. Calude, J. Casti. The jumble cruncher, *New Scientist*, 25 September 2004, 36–37.
- C. S. Calude, Elena Calude, M. J. Dinneen. A new measure of the difficulty of problems, *Journal for Multiple-Valued Logic and Soft Computing* 12 (2006), 285–307.
- C. S. Calude, M. J. Dinneen. Exact approximations of omega numbers, *Int. Journal of Bifurcation & Chaos* 17, 6 (2007), 1937–1954.

- C. S. Calude, M. A. Stay. From Heisenberg to Gödel via Chaitin, *International Journal of Theoretical Physics* 44, 7 (2005), 1053–1065.
- C. S. Calude, M. A. Stay. Natural halting probabilities, partial randomness, and Zeta functions, *Information and Computation* 204 (2006), 1718–1739.
- C. S. Calude, M. A. Stay. Most programs stop quickly or never halt, *Adv. Appl. Math.* 40 (2008), 295–308.
- C. S. Calude, K. Svozil. Quantum randomness and value indefiniteness, *Advanced Science Letters* 1 (2008), 165–168.

Bits and bit-strings

- Bit strings: ε (empty string), $0, 1, 00, 01, 10, 11, \ldots$
- $B = \{0, 1\}$ and B^* is the set of all binary strings
- Strings can be concatenated: from x and y get xy
- The length of the string x is denoted by $|x|$
  - $|\varepsilon| = 0, |0| = |1| = 1, |00| = |01| = |10| = |11| = 2, \ldots$
  - $|xy| = |x| + |y|$
- Let $\mathrm{bin}(n)$ be the binary representation of the number $n + 1$ without the leading 1. For example: $0 \mapsto \varepsilon$, $1 \mapsto 0$, $2 \mapsto 1 \ldots$ This is a bijection between non-negative integers and binary strings
- The quasi-lexicographical order: $x < y$ iff $\mathrm{bin}^{-1}(x) < \mathrm{bin}^{-1}(y)$

Definitions and examples 1

A *machine* is a partial computable function from binary strings to binary strings.

- The identity $\psi(x) = x$, $x \in B^*$ is computable by a machine.
- The function $\delta(x) = 1$ if x is of the form $111\ldots10$ and $\delta(x) = 0$ otherwise is computable by a machine.
- The partial function $\delta'(x) = 1$ if x is of the form $111\ldots10$, and undefined otherwise, is computable by a machine.
- The machines can be effectively listed as $M_0, M_1, \ldots$.

Definitions and examples 2

A set S of strings is **prefix-free** if no string in S is a proper prefix of a string in S.

- The sets $\{x \in B^* : |x| = n\}$, $\{1^n 0 : n \geq 1\}$ are prefix-free.
- The set $\{x_1 x_1 x_2 x_2 \dots x_n x_n 01 : x_1 x_2 \dots x_n \in L\}$ is prefix-free for every $L \subseteq B^*$.
- The set $\{x_1 0 x_2 0 \dots x_n 1 : x_1 x_2 \dots x_n \in L\}$ is prefix-free for every $L \subseteq B^*$.
- B^* is not prefix-free.

Definitions and examples 3

A **prefix-free** (or **self-delimiting**) machine is a machine whose domain is prefix-free.

- The identity $\psi(x) = x$, $x \in B^*$ **is not** computable by a prefix-free machine; in fact no total function is computable by a prefix-free machine.
- The restriction of the identity $\psi(x) = x$ to the set $\{1^n 0 \colon\ n \geq 1\}$ **is** computable by a prefix-free machine.
- The prefix-free machines can be effectively listed as $\tilde{M}_0, \tilde{M}_1, \ldots$ (exercise).

Simulation

A machine T simulates a machine S if for every string x there **effectively exists** a string y such that $T(y) = S(x)$.

Theorem [Simulation]

Every machine can be simulated by a prefix-free machine.

▸ Proof

Universality

Theorem [Universality for machines]

We can construct a machine which can simulate every machine.

▸ Proof

Theorem [Universality for prefix-free machines]

We can construct a prefix-free machine which can simulate every prefix-free machine.

▸ Proof

Cost of simulation

Theorem [Cost-theoretic universality for machines]

There is a machine W such that for every machine T there effectively exists a constant $a = a_{W,T}$ as follows:
for each string x there exists a y with $|y| \leq |x| + a$ such that $W(y) = T(x)$.

Theorem [Cost-theoretic universality for prefix-free machines]

There is a prefix-free machine U such that for every prefix-free machine S there effectively exists a constant $c = c_{U,S}$ as follows:
for each string x there exists a z with $|z| \leq |x| + c$ such that $U(z) = S(x)$.

Exercises 1

1. Prove that bin is a bijection; in what sense is bin 'computable'?
2. Define a prefix free machine S such that for each n, there is exactly one string x with $S(x) = bin(n)$, and

$$|x| = 2|bin(n)| \leq 2 \lg n + 1.$$

Thus, $K(bin(n)) \leq 2 \lg n + c$.

Leibniz

Leibniz's 1686 dictum: **A theory must be simpler than the data is explains.**

Reason: a 'law' which is as complicated as the experimental data that it explains is meaningless; the 'law' explains nothing.

Understanding is compression.

Program-size complexity

Let S be a a prefix-free machine. The **program-size complexity** $K_S(x)$ is the size in bits of the smallest program for S to compute x:

$$K_S(x) = \min\{|p| : \ p \in B^*, S(p) = x\},$$

with the convention that strings not produced by S have infinite complexity.

Universality revisited

Theorem [Cost-theoretic universality, again]

We can effectively construct a prefix-free machine U such that for every prefix-free machine S there effectively exists a constant $c = c_{U,S}$ such that for each input string x:

$$K_U(x) \leq K_S(x) + c.$$

Definition

We call a machine U as in the foregoing theorem a **universal prefix-free machine**.

We pick a particular natural U to use as the basis for AIT. Let K denote K_U.

▸ Natural U

Exercises 2

1. Prove that every universal machine W is onto, i.e. for every y there exists x such that $W(x) = y$. Same for a universal prefix-free machine.
2. Is every universal machine W **infinitely** onto? I.e., do there exist for every y infinitely many x such that $W(x) = y$? Same question for a universal prefix-free machine.
3. Let U be a universal prefix-free machine such that $U(x) = y$. Is the restriction of U to $\mathrm{dom}(U) \setminus \{x\}$ is still universal?

Examples of string complexity

- *low complexity strings:*

$$K(0^n) \leq 2\log n + c, K(1^n) \leq 2\log n + c$$

- *intermediate complexity strings (borderline algorithmically random strings):* if $x^* = \min\{p\colon\ U(p) = x\}$, then

$$K(x^*) \geq |x^*| - c$$

(hardish exercise)

- *high complexity strings (algorithmically random strings):*

$$\max\{K(x)\colon\ |x| = n\} = n + K(\mathrm{bin}(n)) + c \leq n + 2\log n + c$$

▸ Comment

Incomputability of K

Theorem [Incomputability of K]

The program-size complexity K is not computable.

▸ Proof

One can show that in fact K as an oracle computes the halting problem.

Computability in the limit of K

Let for $t \geq 1$

$$K_t(x) = \min\{|y|: \ y \in B^*, U(y) = x \ \text{ in at most } t \text{ steps}\}.$$

Note that $t, x \to K_t(x)$ is computable.

Theorem [Computability in the limit of K]

$K(x) = \lim_t K_t(x)$.

This is clear because if y is a shortest program such that $U(y) = x$, and this computation takes t_0 steps, then $K_t(x) = K(x)$ for all $t \geq t_0$.

Exercises 3

1. Prove that for every universal prefix-free machines U and V there is a constant $c = c_{U,V}$ such that for all strings x:

$$| K_U(x) - K_V(x) | \leq c.$$

2. Prove that that there exists $c > 0$ such that

$$K(x) \leq |x| + 2 \log |x| + c, \text{ for all } x.$$

3. How 'inconvenient' is the dependency of K_U on U?

Exercises 4

1. Consider the text:

   Study this paragraph and all things in it. What is vitally distinct about it? Actually, nothing is wrong, but you must admit that it is most unusual. Don't just zip through it quickly but study it scrupulously. With a bit of luck you should spot what it is so particular about it. Can you say what it is? Try hard as it isn't all that difficult.

Exercises (continued) 4

- Add one sentence to the paragraph without distorting its 'peculiarity'.
- The program-size complexity of the above text is small, intermediate, or maximal? Please explain.

Program-size vs. time

Let t_p be the exact number of steps till $U(p)$ halts (if indeed it halts).

Theorem [Size-Time]

There is a constant c such that if a program p halts on U, then the time t_p it takes $U(p)$ to halt satisfies

$$K(\mathrm{bin}(t_p)) \leq |p| + c. \tag{1}$$

Proof: the prefix-free machine T on input p counts the number of steps t_p till $U(p)$ halts. If this happens, T prints $\mathrm{bin}(t_p)$.
Since this machine is simulated by U with an increase in program length of some c, we obtain the required inequality.

Asymptotically algorithmically random strings

A binary string x is "asymptotically algorithmically random" if $K(x) \geq |x| - \log(|x|)$.

Theorem

Most binary strings of a given length n are asymptotically algorithmically random, because the set of such strings has high density:

$$\#\{x \in B^* : |x| = n, K(x) \geq n - \log(n)\} \cdot 2^{-n} \geq 1 - 1/n.$$

This tends to 1 when $n \to \infty$.

Incompressible strings

We say that string x is b-incompressible if $C(x) \geq |x| - b$. Otherwise, x is called b-compressible.

In assigment 4 you show that for each n, at least $2^n - 2^{n-b} + 1$ strings x of length n are b-incompressible.

This means, only a fraction of about 2^{-b} is b-compressible.

Berry's Paradox

Let x be the least natural number that cannot be described in natural language by fewer than 100 symbols.

This is a description of that number x with 90 symbols.

Chaitin's incompleteness theorem

Theorem (Chaitin)

A formal system $\mathcal{F}$ can prove b-incompressibility only for finitely many strings.

- The proof is by contradicton.
  If the statement fails, then for infinitely many n, the following is a description with only $\log_2 n + c$ bits of an n-bit string:

  > *"the first string of length n for which $\mathcal{F}$ proves b-incompressibility".*

  Here the constant c only depends on $\mathcal{F}$ and b.
- Hence, if n is sufficiently large so that $\log_2 n + c < n - b$, such a string is b-compressible, contradiction.

Infinite sequences

A binary infinite sequence

$$\mathbf{x} = x_1 x_2 \cdots x_n \cdots$$

is a sequence in which each term is 0 or 1.

The prefix of length n of the sequence $\mathbf{x}$ is

$$\mathbf{x}(n) = x_1 x_2 \cdots x_n \in B^*.$$

The program-size of the sequence x is given by the sequence

$$(K(\mathbf{x}(n))_{n \in \mathbb{N}}.$$

The binary representation of the number 0.40626 is

$$0.011010000\cdots$$

Consider the sequence

$$\mathbf{x} = 011010000\cdots$$

This has slow-growing complexity: There exists $c > 0$ such that for each n, $K(\mathbf{x}(n)) \leq 2\log n + c$.

Consider the number π written in binary and let

$$0.\pi_1\pi_2\cdots\pi_n\cdots$$

be the infinite binary sequence of the fractional part of π. There is a constant $c' > 0$ such that for each n, $K(\mathbf{\Pi}(n)) \leq 2\log n + c'$, where

$$\mathbf{\Pi} = \pi_1\pi_2\cdots\pi_n\cdots$$

Sequences with maximal complexity

Are there sequences $\mathbf{x} = x_1 x_2 \cdots x_n \cdots$ having **all prefixes algorithmically random**

$$\exists c \forall n \, [K(\mathbf{x}(n)) \geq n + K(\mathrm{bin}(n)) - c]?$$

The answer is **negative**.

Are there sequences **x** having **all prefixes with intermediate complexity**:

$$\exists c \forall n \, [K(\mathbf{x}(n)) \geq n - c]?$$

The answer is **affirmative**. These sequences have **nearly maximal complexity**.

Omega numbers

The 'canonical' example of a sequence with nearly maximum complexity K is Chaitin's Omega, the "halting probability" of U:

$$\Omega_U = \sum_{p \in \mathrm{dom}(U)} 2^{-|p|}.$$

The Kraft inequality states that for a prefix-free set S,

$$\sum_{p \in S} 2^{-|p|} \leq 1.$$

This implies that $\Omega_U \leq 1$; in fact, $\Omega_U \in (0, 1)$.

Theorem [Omega solves the halting problem]

With the first n bits of Omega we solve the halting problem for every program of length less than or equal to n.

In other words, if we put the first n bits of Omega on the oracle tape, we can decide whether $U(q)$ halts, for any q such that $|q| \leq n$. ▸ Proof

Corollary [Incomputability of Omega]

The sequence $\omega_1\omega_2\cdots\omega_n\cdots$ is not computable.

Some of these bits have actually been determined. For a natural choice of the machine U, the first 40 bits of $\Omega = \Omega_U$ are

$$0001000000010000101001110111000011111010.$$

If we knew the first 5,000 bits, we would know if the Riemann hypothesis is correct. For the Four colour theorem we need even fewer bits. ▸ OmegaMedia

Omega: transcendence

A number is called algebraic if it is a zero of a polynomial with rational coefficients. Otherwise, it is called transcendental.
Example: $\sqrt{2}$, and $\sqrt{2}+\sqrt{3}$ are algebraic; e, π are transcendental.

Corollary [Transcendence of Omega]

The number Ω is transcendental.

Proof: every algebraic number is computable, because there is an algorithm (Newton, for instance) to find the zeros of a polynomial with rational coefficients.

Omega: prefix complexity

Theorem [Omega has nearly maximum complexity]

Let R be a universal prefix-free machine, and let $\Omega_R = 0.\omega_1\omega_2\cdots\omega_n\cdots$.
There exits a constant $c > 0$ such that for all $n \geq 1$,

$$K(\omega_1\omega_2\cdots\omega_n) \geq n - c.$$

▸ Proof

Definition

A reals r whose binary expansions are sequences with nearly maximum complexity, i.e. there is c such that $K(r_1 \ldots r_n) \geq n - c$ for each n, is called **algorithmically random**, or shortly **random**; .

Theorem [Omega is c.e.]

The number Omega is c.e., that is, the limit of an increasing computable sequence of rationals in $(0, 1)$.

▸ Proof

So, Omega is a c.e. algorithmically random real. In fact, the converse is also true. Say that $r \in (0, 1)$ is an Omega number if it satisfies the definition of Omega for some universal prefix-free machine.

Theorem [Omega = c.e. and random]

The set of Omega numbers coincides with the set of c.e. random numbers.

1. The following number is connected with the halting problem:

$$\Xi = \sum_{\mathrm{bin}(i)\in \mathrm{dom}_U} 2^{-i}.$$

   Is Ξ computable, c.e., algorithmically random?

2. Let $\mathit{Time}(p)$ be the running time of the computation $U(p)$, i.e. if $U(p)$ halts, $\mathit{Time}(p) = t_p$, otherwise $\mathit{Time}(p) = \infty$, and define

$$\Upsilon = \sum_i 2^{-i} / \mathit{Time}(\mathrm{bin}(i)).$$

   Is Υ computable, c.e., algorithmically random?

1. Let α be a real in $(0, 1]$. Show that the following conditions are equivalent:
   - There is a computable, nondecreasing sequence of rationals which converges to α.
   - The set of rationals less than α is c.e.
   - There is an infinite prefix-free c.e. set $A \subseteq B^*$ with $\alpha = \Omega_A$.

The probability that a program generated by a coin tossing run on the prefix-free machine S will produce the output x is:

$$P_S(x) = \sum_{\{y\,:\ S(y)=x\}} 2^{-|y|}.$$

The coding theorem

Put $P(x) = P_U(x)$. The Coding Theorem states that $K(x)$ is within a constant of $-\log_2 P(x)$. This shows that least program size for describing a string is related to the entropy.
(The entropy of a partition of the probability space (X, P) into events $A_1, \ldots, A_n$ is defined to be $-\sum_i P(A_i) \log_2 P(A_i)$.)

Theorem [Coding theorem]

There exists a constant $c > 0$ such that for all x,

$$| K(x) + \log_2 P(x) | \leq c.$$

The Kraft-Chaitin Theorem

To prove the coding theorem we need the following main result from AIT.

Theorem

Let $(r_i, x_i)_{i \in \mathbb{N}}$ be an effective list of pairs of a natural number and a string, such that

$$\sum_i 2^{-r_i} \leq 1.$$

Then there is a prefix-free machine M such that for each i, there is a program p of length r_i such that $M(p) = x_i$.

Proof: see literature. Note: $(r_i, x_i)_{i \in \mathbb{N}}$ is called a Kraft-Chaitin sequence. A pair (r, x) is called a request.

Proof of Coding Theorem

Clearly $2^{-K(x)} \leq P_U(x)$ and hence $K(x) \geq -\log_2 P_U(x)$, since a shortest program for x contributes $2^{-K(x)}$ to $P_U(x)$.

Now we show that $K(x) \leq -\log_2 P_U(x) + c$ for some c. Build a KC-sequence as follows: when we see that now at the current stage s, we have $2^{-r} \leq P_{U_s}(x)$, then we put request $(r+1, x)$ into the sequence.

- One checks that this is a KC sequence.
- For the machine M we now obtain via the KC theorem, $K_M(x) \leq -\log_2 P_U(x) + 2$.

Omega as a product

Let w be such that for each $s \in \mathrm{dom}(U)$, w is not a prefix or an extension of s, and let $S = \{s_1 s_2 \cdots s_n w : \ s_i \in \mathrm{dom}(U), \mathrm{bin}^{-1}(s_1) \leq \mathrm{bin}^{-1}(s_2) \leq \ldots \leq \mathrm{bin}^{-1}(s_n)\}$. Define

$$W(s_1 s_2 \cdots s_n w) = U(s_1)U(s_2)\cdots U(s_n).$$

Theorem [Product Omega]

The prefix-free machine W is universal and

$$\Omega_W = 2^{-|w|} \cdot \prod_{p \in \mathrm{dom}(U)} \frac{1}{1 - 2^{-|p|}}.$$

Furthermore, $\Omega_U \leq \Omega_W \leq 2\Omega_U$.

▸ Proof

Omega as a zeta function

The **Zeta number** of a prefix-free universal machine U is

$$\zeta_U = \sum_{\{n \geq 1 :\ \mathrm{bin}(n) \in \mathrm{dom}(U)\}} \frac{1}{n}.$$

Theorem [Zeta number is random]

The number ζ_U is c.e. and random, and $\Omega_U > \zeta_U > \Omega_U/2$.

▸ Proof

1. Construct a prefix-free universal machine U' such that $\Omega_{U'} = \frac{1}{2}\Omega_U$. What about changing $\frac{1}{2}$ into $\frac{3}{7}$?
2. **[Bonus]** Prove that the set of Omega numbers is dense in (0,1), that is, every real in (0,1) is a limit of a sequence of Omega numbers.
3. **[Bonus]** Let U be a prefix-free universal machine machine. Assume that we know exactly which programs p of length less than or equal to m halt. How many bits of Ω, $\omega_1, \omega_2, \cdots, \omega_i$, can we compute?

Characteristics of algorithmic randomness

- It satisfies **all** computable enumerable statistical properties of randomness.
- It is **unpredictable**.
- It is **Turing incomputable**.

What is quantum randomness?

Quantum randomness appears to occur in two different scenarios:

(i) the complete impossibility to predict or explain the occurrence of certain *single* events and measurement outcomes from any kind of operational causal connection, and

(ii) the concatenation of such single quantum random events forms sequences of random bits which can be expected to be equivalent stochastically to white noise: it carries the least correlations, as the occurrence of a particular bit value in a binary expansion does not depend on previous or future bits of that expansion.

Characteristics of quantum randomness

- It has been confirmed by theoretical and experimental research.
- It passes **all reasonable** statistical properties of randomness.
- It is **Turing incomputable**. ▸ Details
- Can be easily and reliably produced: A photon generated by a source beamed to a semitransparent mirror is reflected or transmitted with 50 per cent chance, and these measurements can be translated into a string of quantum random bits. ▸ JPict

Quantis

Quantis: quantum mechanical random number generator produced and sold by *id Quantique* of the University of Geneva

The hybrid computer "PC plus Quantis" (used theoretically to generate an infinite sequence of quantum random bits) trespasses the Turing barrier. The machine **exists** and **is used**.

1. What is the computational power of the hybrid machine "PC plus Quantis"?
2. How random is quantum randomness? More precisely, to what extent is quantum randomness algorithmic randomness?
3. How accurate are simulations powered by quantum random bits? Precisely, is Rabin's probabilistic test of primality exact (not only highly probable) if powered by a quantum source of randomness?

Proof

Let S be a machine and construct the prefix-free machine T by

$$T(x_1 0 x_2 0 \dots x_n 1) = S(x),$$

where $x = x_1 x_2 \dots x_n$ (if $x = \varepsilon$ we take 1). ▸ SimulTheorem

Proof

Let $(T_i)_{i\in\mathbb{N}}$ be a computable enumeration of all machines and construct the machine W by

$$W(1^i 0x) = T_i(x).$$

▸ UnivTheorem

Proof

Let $(S_i)_{i\in\mathbb{N}}$ be a computable enumeration of all prefix-free machines and construct the prefix-free machine U by

$$U(1^i 0x) = S_i(x).$$

▸ UnivTheorem

▸ UnivRevisted

Proof

We prove the following stronger result:

There is no partial computable (p.c.) function φ from strings to non-negative integers with infinite domain such that $K(x) = \varphi(x)$, for all $x \in \mathrm{dom}(\varphi)$.

Assume, by absurdity, that $K(x) = \varphi(x)$, for all $x \in \mathrm{dom}(\varphi)$, where φ is as above.
Let $A \subset \mathrm{dom}(\varphi)$ be a computable, infinite set, and let f be the partial function given by

$$f(0^i 1) = \min\{x \in A \colon\ K(x) \geq 2^i\}, i \geq 1.$$

Proof continued

Since $\varphi(x) = K(x)$, for $x \in A$, it follows that f is a p.c. function. Moreover, f has a computable graph and f takes as values strings of arbitrarily length.
By definition, for all $i > 0$,

$$K(f(0^i1)) \geq 2^i.$$

As $S(0^i1) = f(0^i1)$ is a prefix-free machine, in view of the universality theorem there exists $c > 0$ such that for all $i > 0$ we have:

$$2^i \leq K(f(0^i1)) \leq K_S(f(0^i1)) + c \leq i + 1 + c,$$

a contradiction for sufficiently large i. ▸ IncomputTheorem

Proof

Intuitively, $time(p)$ either calculates the number of steps t_p till $U(p)$ halts and prints $\mathrm{bin}(t_p)$, or, if $U(p)$ is not defined, never halts. The constant c can be taken to be less than or equal to 2, as the counting instruction is used only once, and we need one more instruction to print its value; however, we don't need to print the value $U(p)$.

Proof continued

Without loss of generality, we assume that U has a built-in counting instruction. Based on this, there is an effective transformation which for each program p produces a new program $time(p)$ such that there is a constant $c > 0$ (depending upon U) for which the following three conditions are satisfied:

1. $U(p)$ halts iff $U(time(p))$ halts,
2. $|time(p)| \leq |p| + c$,
3. if $U(p)$ halts, then it halts at the step $t_p = \mathrm{bin}^{-1}(U(time(p)))$.

Proof continued

We have:

$$\begin{aligned} K(\mathrm{bin}(t_p)) &= K(U(time(p)) \\ &= \min\{|w|:\ U(w) = U(time(p))\} \\ &= |time(p)| + c \\ &\leq |p| + c. \end{aligned}$$

▸ SizevsTimeTheorem

Proof

First we prove that for every $n \geq 4$ and $t \geq 2^{2n-1}$, we have:

$$2^{|\mathrm{bin}(t)|} > 2^n \cdot |\mathrm{bin}(t)|. \tag{2}$$

Indeed, the real function $f(x) = 2^x/x$ is strictly increasing for $x \geq 2$ and tends to infinity when $x \to \infty$. Let $m = |\mathrm{bin}(t)|$. As $2^{2n-1} \leq t < 2^{m+1}$, it follows that $m \geq 2n-1$, hence $2^m/m \geq 2^{2n-1}/(2n-1) > 2^n$. The inequality is true for every $|\mathrm{bin}(t)| \geq 2n-1$, that is, for every $t \geq 2^{2n-1}$.

Proof continued

Next we take $n = N + c + 1$ in (2) and we prove that every asymptotically algorithmically random time $t_p \geq 2^{2N+2c+1}$, $N \geq 2$, does not satisfy the inequality (1):

$$|\mathrm{bin}(t_p)| - \log |\mathrm{bin}(t_p)| \leq K(\mathrm{bin}(t_p)) \leq N + c + 1,$$

$$n < |\mathrm{bin}(t_p)| - \log |\mathrm{bin}(t_p)| \leq N + c + 1.$$

Consequently, no program of length N which has not stopped by time $2^{2N+2c+1}$ will stop at an asymptotically algorithmically random time.

▸ WhatTimeProgramsStop

Proof

Given $\omega_1\omega_2\cdots\omega_n$, we run in parallel $U(p)$ on more and more programs, till we get programs $p_1, \ldots, p_N$ such that

$$\sum_{i=1}^{N} 2^{-|p_i|} \geq 0.\omega_1\omega_2\cdots\omega_n.$$

Now suppose $\omega_1\omega_2\cdots\omega_n$ is the beginning of Ω in binary. If a program q with $|q| \leq n$ is different from all p_i above, then it doesn't stop: otherwise

$$\Omega < 0.\omega_1\omega_2\cdots\omega_n + 2^{-n} \leq \sum_{i=1}^{N} 2^{-p_i} + 2^{-|q|} \leq \Omega,$$

a contradiction. ▸ OmegaHaltTheorem

Proof

We will write Ω for Ω_R, and we let

$$\Omega_s = \sum\{2^{-|p|} : \ R(p) \downarrow \text{ in at most } s \text{ steps}\}.$$

Let N be the (plain) machine that works as follows on an input x of length n.

(1) Wait for t such that $0.x \leq \Omega_t < 0.x + 2^{-n}$.

(2) Output the least string$y$ such that $R(y)$ doesn't halt in t steps.

If $x = \omega_1 \ldots \omega_n$ then such a t exists. By stage t all R-descriptions of length $\leq n$ have appeared, for otherwise $\Omega \geq \Omega_t + 2^{-n}$, contrary to choice of t. Thus $K(y) > n$ where $y = N(x)$.
This implies

$$\forall n\,[K(\omega_1 \ldots \omega_n) + c \geq K(N(\omega_1 \ldots \omega_n)) > n]$$

for an appropriate constant c.

▸ OmegaMaxComplTheorem

Proof

Let f be a computable enumeration of the domain of U. Then the sequence of rationals

$$r_N = \sum_{i=1}^{N} 2^{-|f(i)|}$$

is computable, increasing and converges to Ω.

▸ CEOmegaTheorem

Proof

Clearly, $\mathrm{dom}(U)w \subset S = \mathrm{dom}(W)$. Given m bits of Ω_W we can decide the halting status of all programs $p \in S$ with $|p| \leq m$; this set includes all programs $p \in \mathrm{dom}(U)$ which halt and have at most $m - |w|$ bits in length. Repeating the proof of the randomness of Ω_U, construct a computable function f such that

$$K(f(w_1 w_2 \cdots w_m)) > m - |w|,$$

where $\Omega_W = 0.w_1 w_2 \cdots w_m \cdots$

Proof continued

By the universality of U, there is a constant $c > 0$ such that

$$K(w_1 w_2 \cdots w_m) + c \geq K(f(w_1 w_2 \cdots w_m)) > m - |w|,$$

so

$$K(w_1 w_2 \cdots w_m) > m - (c + |w|).$$

▸ OmegaProductTheorem

Proof

Given m bits of $\zeta_U = 0.z_1 z_2 \cdots z_n \cdots$ we can decide the halting status of all programs $\mathrm{bin}(j) \in \mathrm{dom}(U)$ with $|\mathrm{bin}(j)| \leq m$: indeed, since

$$\frac{1}{j} = 2^{-\log_2 j} > 2^{-\lfloor \log_2 j \rfloor - 1},$$

the longest of these programs would affect at least the last bit of $zeta_U$. We repeat the proof of the randomness of Ω_U.

▸ ZetaOmegaTheorem

Proof

To prove (**??**) we consider for every $k \geq 1$ the strings

$$w_1 x_{F(1)} w_2 x_{F(2)} \dots w_k x_{F(k)}, \tag{3}$$

where each w_j is a string of length $F(j) - F(j-1) - 1, F(0) = 0$, that is, all binary strings of length $F(k)$ where we have fixed bits at the positions $F(1), \dots, F(k)$.
It is clear that $\sum_{i=1}^{k} |w_i| = F(k) - k$ and the mapping

$$(w_1, w_2, \dots, w_k) \mapsto w_1 w_2 \dots w_k$$

is bijective, hence to generate all strings of the form (3) we only need to generate all strings of length $F(k) - k$.

Proof continued

Next we consider the enumeration of all strings of the form (3) for $k = 1, 2, \ldots$. The lengths of these strings will form the sequence

$$\underbrace{F(1), F(1), \ldots, F(1)}_{2^{F(1)-1} \text{ times}}, \ldots, \underbrace{F(k), F(k), \ldots, F(k)}_{2^{F(k)-k} \text{ times}}, \ldots$$

which is computable and satisfies the inequality (4) as

$$\sum_{k=1}^{\infty} 2^{F(k)-k} \cdot 2^{-F(k)} = 1.$$

Theorem [Kraft-Chaitin theorem]

Let $n_1, n_2, \ldots$ be a computable sequence of non-negative integers such that

$$\sum_{i=1}^{\infty} 2^{-n_i} \leq 1. \tag{4}$$

Then, we can effectively construct a prefix-free sequence of strings (that is no w_i is a proper prefix of any w_j with $i \neq j$) $w_1, w_2, \ldots$ such that for each $i \geq 1$, $|w_i| = n_i$.

Proof continued

By Kraft-Chaitin theorem, for every string w of length $F(k) - k$ there effectively exists a string z_w having the same length as w such that the set $\{z_w : |w| = F(k) - k, k \geq 1\}$ is prefix-free.

Indeed, from a string w of length $F(k) - k$ we get a unique decomposition $w = w_1 \dots w_k$, and z_w as above, so we can define $S(z_w) = w_1 x_{F(1)} w_2 x_{F(2)} \dots w_k x_{F(k)}$; S is a prefix-free machine.

Proof continued

Clearly, for all $k \geq 1$:

$$\Delta_S(w_1 x_{F(1)} w_2 x_{F(2)} \dots w_k x_{F(k)})$$

$$\leq \nabla_S(w_1 x_{F(1)} w_2 x_{F(2)} \dots w_k x_{F(k)}) \leq \mathrm{bin}^{-1}(z_w) \leq 2^{F(k)-k+1} - 1.$$

In particular, $\Delta_S(x_1 \dots x_{F(k)}) \leq 2^{F(k)-k+1} - 1$, so by the universality theorem we get the inequality (**??**).

▸ LowComplexityTheorem

Proof

If the sequence were random, then the formal uncertainty principle (**??**) will hold true, hence for each $k \geq 1$, we would have the following contradictory pair of inequalities:

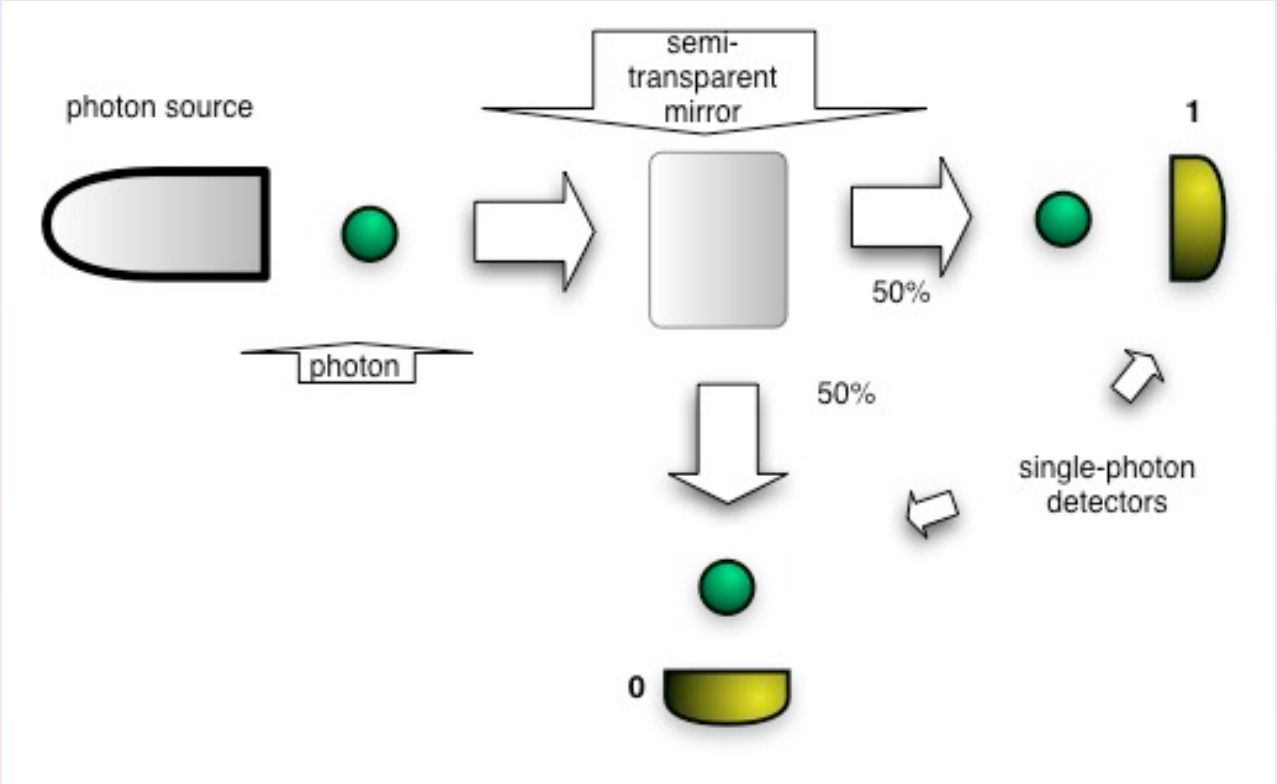
$$\varepsilon_1 \cdot \frac{1}{\Delta_{F(k)}} \leq \Delta_U(x_1 \dots x_{F(k)}) \leq \varepsilon \cdot 2^{F(k)-k}.$$

▸ NonRandTheorem

Proof

We deduce the information-theoretic incompleteness theorem from the uncertainty relation (**??**). Assume by absurdity that *ZFC* can determine infinitely many digits of $\Omega_U = 0.\omega_1\omega_2\ldots$. Then, we could enumerate an infinite sequence of digits of Ω_U, thus contradicting the above theorem.

▸ UncertImpliesIncompletTheorem



Optical system for generating quantum random bits

▸ ORG

Axioms for quantum randomness

- The single outcome from which quantum random sequences are formed, occurs unbiased; i.e., for the ith outcome, there is a 50:50 probability for either 0 or 1:

$$\mathrm{Prob}(x_i = 0) = \mathrm{Prob}(x_i = 1) = \tfrac{1}{2}. \tag{5}$$

- There is a total independence of previous history, such that no correlation exists between x_i and previous or future outcomes. This means that the system carries no memories of previous or expectations of future events. All outcomes are temporally "isolated" and free from control, influence and determination. They are both unbiased and self-contained.

Incomputability of quantum randomness

Value indefiniteness manifests itself in the "scarcity" or non-existence of two-valued states –interpretable as classical truth assignments – on all or even merely a finite set of physical observables (Kochen-Specker theorem).

If we assume the axioms of randomness and value indefiniteness, then every sequence of quantum random bits is incomputable, i.e. there is no single machine which can generate exactly the bits of the sequence.

▸ ORG

Philosophical resonances

- An algorithmically random string is one for which there is no theory in Leibniz's sense.
- An algorithmically random string is "unexplainable", "incomprehensible" except as 'a thing in itself' (*Ding an sich* in Kant's terminology).

▸ StringComplexity

▸ IncomputTheorem

An illustration of the coding theorem

Look at a monkey trying to "type" the entire work of Shakespeare, of say $1,000,000$ bits long. If the monkey types "at random" on a dumb typewriter, the probability that the result is Shakespeare's work is

$$2^{-1,000,000};$$

if the monkey sits in front of a computer terminal, then the algorithmic probability that it types the same text is

$$2^{-K(\text{Shakespeare})} \asymp 2^{-250,000},$$

an event with an extremely small chance to happen, but still more likely than the first event.

An illustration of the coding theorem (continued)

The use of the typewriter reproduces exactly the input produced by typing while a computer "runs" the input and produces an output. Consequently, a random input to a computer is much more likely to produce an "interesting" output than a "random" input to a typewriter. Is this a way to create "sense" out of "nonsense"?

▸ CodingTheorem

Ω in the media

The CBS drama TV show Numb3rs, `http://www.cbs.com/primetime/numb3rs/`, season 5; episode 5; scene 6) includes the following dialogue:

> LARRY: *Ah, Charles, my ambulatory reference book. Chaitin's Omega Constant...?*
>
> CHARLIE: *Omega equals .00787499699. Why, what're you working on?* (sees the file, reacts) *Oh. FBI file.*

The math is explained at `http://numb3rs.wolfram.com/505`.

▸ Incomputability of Omega

Numb3rs: Ω videoclip

▸ Incomputability of Omega