Information: The Algorithmic Paradigm

C. S. Calude
University of Auckland, NZ

CDMTCS-292
December 2006

Information: The Algorithmic Paradigm

Cristian S. Calude

Department of Computer Science
University of Auckland, New Zealand
www.cs.auckland.ac.nz/~cristian

1 Introduction

Information has a diversity of meanings, from everyday usage to a variety of technical settings. There is no single theory of information, but several theories, Shannon's information theory [27, 28, 16], semantic theories [2], logic of information [18], information algebra [21], philosophy of information [19], information flow [3], quantum information theory [24], evolutionary information [30], algorithmic information theory [15, 4], to name just a few. Each theory focuses on some specific aspects of information, and overlaps are minimal. Information is context-sensitive and heavily dependent on the adopted coding.

In this paper we will present, through a sequence of examples, some ideas and results of the algorithmic approach to information. In this approach information is measured by counting bits encoding computations.

2 Counting bits

Information, in a broad sense, can be measured in various units, from bits to dollars. In this paper we shall confine ourselves to bits. The bit, short for binary digit, was first used in 1946 by John Tukey. A single bit can hold only one of two values: 0 or 1. More information is obtained by combining consecutive bits into larger units, bit-strings (shortly, strings): 00, 01,10,11, 000, 001,..., 111, 0000,... Sometimes it is useful to consider the empty string denoted by λ.

Strings have length: the number of characters of a string. For example, the length of 0 is 1, the length of 1110111 is 7, the length of the empty string is 0. Strings can be concatenated: the concatenation of the strings x and y is xy. The length of xy is the sum of the lengths of x and y.

Bits can be very useful to measure information. The power of bits can be illustrated with the information which can be encoded in, say, 20 bits. With a simple strategy, twenty questions/answers elicit 20 bits of information, which correspond to a single choice among $2^{20} = 1,048,576$ equally probable alternatives. For example, with the information in a bit-string of length 20 one can identify any town in USA. The limit of this approach is most visible at the level of semantics: No meaning is captured! For example, translated in binary, *'happy birthday', 'ya dirthbppayh'* have the same information content.

The following *guessing game* is a more interesting example illustrating the power of bits: one person chooses a (secret) natural number and another person tries to guess it. The person who guesses is only allowed to ask questions of the following form: "Is your number less than n?" for every natural $n \geq 0$; the other person truthfully answers *yes* or *no*. The aim is to guess the number as fast as possible, that is, with as few questions as possible.

As an example consider the following questions:

1. Is your number less than 1?
2. Is your number less than 2?
3. Is your number less than 3?
4. Is your number less than 4?

...

and so on until the first *yes* comes out.

To guess the number 10 we need to ask the first 11 questions; in general, to guess the number n we have to ask the first $n+1$ questions. This solution leads to an encoding of all naturals numbers: with a bit-string of length $n+1$ we encode the number n.

Can we do it better? Certainly. For example, we start asking the questions "Is your number less than 2^i?" for $i = 1, 2, \ldots$ till we get the answer "yes". This will happen at a value i such that $2^{i-1} \leq n < 2^i$. Then, we continue by halving the length of the interval. For example, to guess the number 10 we need 8 questions (corresponding to $i = 1, 2, 4, 8, 16, 12, 9, 10$). In general, to guess the number n we have to ask the first $2\log n + 1$ questions (here $\log n$ is the integer part of $\log_2 n$, the base-2 logarithm of n). Note that this approach is better than the first one for $n > 3$.

Still, can we do it better? This is possible if we consider large enough numbers n: we can design better and better solutions. Does there exist an optimal solution? An answer will be given in the next section.

3 The halting problem

The halting problem (for Turing machines) is the problem to decide whether an arbitrary Turing machine eventually halts on an arbitrary input:

> Does there exist a Turing machine T_{halt} which given the code *code*(T) of a Turing machine T, and the input x, eventually stops and produces 1 if $T(x)$ stops and 0 if $T(x)$ does not stop?

Turing's result states that *the halting problem cannot be solved by any Turing machine*, i.e. there is no such T_{halt}. Here is an information-theoretical proof, [14, 4]. Instead of Turing machines we will deal with the informal notion of "program". We assume that programs incorporate inputs—which are coded as natural numbers. So, a program may run forever (does not halt) or may eventually stop, in which case it prints a natural number.

Assume that there exists a *halting program* deciding whether an arbitrary program will ever halt. Construct the following program, called P:

```
1. read a natural $N$;
2. generate all programs up to $N$ bits in size;
3. use the halting program to check for each generated program whether it
   halts;
4. simulate the running of the above generated programs, and
5. output a number different from each output produced by the above programs.
```

The program P halts for every natural N. How long is P? Answer: $\log N +$constant bits. Reason: to code N we need about $\log_2 N$ bits and the rest of the program P is constant.

For large N, the program P belongs to the set of programs having less than N bits (because $\log N + \text{constant} < N$). Accordingly, for such an N, the program P will be generated by itself at some stage of the computation. We have got a contradiction since P outputs a natural number different from the output produced by itself!

Consider now all programs of length at most n, i.e. $2^{n+1}-1$ programs. Some programs halt, some do not halt. If we order lexicographically all programs of length n and ask, for each such program, whether it halts or not, we get a bit-string of length $2^{n+1}-1$ encoding the whole information. Is it possible to encode the same amount of information with fewer bits?

The answer is affirmative and a solution will be presented in what follows. We need more technical details. We revert our discussion to Turing machines, but of a very special type, self-delimiting Turing machines. The domain of a Turing machine T—dom(T)—is the set of strings where T halts; if dom(T) is prefix-free, i.e. no string in dom(T) is a proper extension of another string in dom(T), then T is called a *self-delimiting Turing machine*. An important result is the *universality theorem*:

There effectively exists a self-delimiting Turing machine U, called universal, such that for every self-delimiting Turing machine T we can compute a constant c, depending only on U and T, satisfying the following property: if $T(x)$ stops, then $U(x') = T(x)$, for some string x' with length no longer than the length of x plus c.

In the framework of self-delimiting Turing machines the above coding problem can be stated as follows: given a universal self-delimiting Turing machine U and an integer $n > 0$, find an encoding via a bit-string shorter than 2^n from which one can check which program x of length less than n stops on U. This encoding was discovered by Chaitin in 1975 who introduced the Omega number, see [15, 4]:

$$\Omega_U = \sum_{\{x \mid U(x) \text{ halts}\}} 2^{-|x|}, \tag{1}$$

where $|x|$ denotes the length of x. We consider the binary expansion of Ω_U

$$\Omega_U = 0.\omega_1\omega_2\cdots\omega_m\cdots \tag{2}$$

Given the first n bits of the binary expansion (2) of Ω_U, $\omega_1\omega_2\cdots\omega_n$, we can decide which programs x of length less than n halt on U: we enumerate enough elements $p_1, p_2, \ldots, p_k$ in the domain of U till the sum $\sum_{i=1}^{k} 2^{-|p_i|}$ becomes larger than or equal to $0.\omega_1\omega_2\cdots\omega_n$. We have:

$$\{x \mid |x| \le n, U(x) \text{ halts}\} \subseteq \{p_1, p_2, \ldots, p_k\},$$

so the halting programs x with $|x| \le n$ can be obtained by eliminating from the set $\{p_1, p_2, \ldots, p_k\}$ all programs of length larger than n. Indeed,

$$\Omega_U < 0.\omega_1\omega_2\cdots\omega_n + 2^{-n},$$

and for every halting program $q \notin \{p_1, p_2, \ldots, p_k\}$ with $|q| \le n$ we then have:

$$\sum_{i=1}^{k} 2^{-|p_i|} + 2^{-|q|} \ge 0.\omega_1\omega_2\cdots\omega_n + 2^{-n} > \Omega_U,$$

a contradiction.

We have shown that the halting information for all programs of length less than or equal to n (a set containing $2^{n+1}-1$ elements) can be compressed into a string of length n: $\omega_1\omega_2\cdots\omega_n$.[1]

We are now in the position to give an answer to the question posed at the end of the section 2. Recall, we are interested in constructing an infinite prefix-free set of bit-strings to code as efficiently as possible all non-negative integers. The domain of a universal self-delimiting Turing machine is such a code (cf. [1]):

Let A be a set of bit-strings. The following two conditions are equivalent:

a) The set A is the domain of a universal self-delimiting Turing machine.

b) For every computable one-one function $g : \{0,1,2,\ldots\} \to \Sigma^*$ having a prefix-free range, there exist a computable one-one function $f : \{0,1,2,\ldots\} \to \Sigma^*$ and a constant $k \geq 0$ such that

a. $f(\{0,1,\ldots\}) \subseteq A$,

b. $|f(n)| \leq |g(n)| + k$, *for every* $n \geq 0$.

The good news is that coding with programs in the domain of a universal self-delimiting Turing machine is optimal up to an additive constant (and one can show that a better coding does not exist). The bad news is that this coding is computably enumerable, but not computable.

Finally, we ask the question: which problems can be solved knowing finitely many bits of Ω_U (see 2)? To answer this question we will present an implementation of a specific universal self-delimiting Turing machine U based on a register machine program, see [6]. A register machine has a finite number of registers, each of which may contain an arbitrarily large non-negative binary integer. The register machine U (labelled) instructions are:

L: EQ R1 R2 R3
L: SET R1 R2
L: ADD R1 R2
L: READ R1
L: HALT

The names of the above instructions are self-explanatory. For instance, the first instruction is the classical if-then-else condition. In all cases R2 denotes either a register or a binary constant of the form $1(0+1)^* + 0$, while R1 and R3 must be register variables.

A *register machine program* consists of a finite list of labelled instructions from the above list, with the restriction that the HALT instruction appears only once, as the last instruction of the list. The input data (a binary string) follows immediately after the HALT instruction. A program not reading the whole data or attempting to read past the last data-bit results in a runtime error. Some programs have no input data.

It is perhaps surprising that many problems in mathematics can be reformulated in terms of the halting/non-halting status of appropriately constructed self-delimiting Turing machines. For

[1] The converse implication is not true: we may know exactly which programs of length less than n halt and still not know any bit of Ω_U, cf. [5].

example, consider Fermat's Last Theorem, stating that there are no integers $x, y, z, n > 3$ such that $x^n + y^n = z^n$. We can construct a self-delimiting Turing machine T_{Fermat} which systematically enumerates all possible integers (for example, written in binary) $x, y, z, n > 3$, checks whether $x^n + y^n = z^n$, and stops if for some values x, y, z, n the relation is true (which would mean that the program has found a counter-example); otherwise, T generates a new 4-tuple x, y, z, n and repeats the above procedure. Fermat's Last Theorem is equivalent with the statement "T_{Fermat} never halts", hence knowing that Fermat's Last Theorem is true we know that T_{Fermat} never halts.

In this way we can measure the difficulty of Fermat's Last Theorem by the complexity of T_{Fermat}, for example, by the number of bits necessary to specify T_{Fermat} in some fixed formalism (say, U). Of course, there are many self-delimiting Turing machines equivalent to T_{Fermat}, so a natural way to evaluate the complexity is to consider the least complex such machine. And, of course, this extends to any problem Π for which we can construct a self-delimiting Turing machine T_Π such that Π is false if and only if $U(T_\Pi)$ halts (if such a program exists): the *difficulty* of such a problem Π is the minimal number of bits of Ω_U necessary to test whether C_Π stops on U.

Here are three important open questions that can be analysed with this method (cf. [6]):

- *Goldbach's Conjecture*:[2] the program T_{Goldbach} has 135 instructions totalling 3,484 bits.
- *Riemann Hypothesis*:[3] the program T_{Riemann} consists of 290 instructions totalling 7,780 bits.
- *Collatz' Conjecture*:[4] there is a *non-constructive* way to prove that there exists a program T_{Collatz} which never stops iff the Collatz' Conjecture is true.

Are the numbers specified above the exact *difficulties* of the corresponding problems? Definitely not, they are upper bounds! The bad news is that, as expected, the problem of computing the *difficulty* of a problem is not computable. The good news is one can work with upper bounds: changing U will result in a change of upper bounds, but the order of difficulty will be preserved, namely if Π_1 is more difficult than Π_2 for U, the same relation will be true for any other universal self-delimiting Turing machine U^*. Specifically, the above analysis shows that the Riemann Hypothesis is more difficult than the Goldbach's Conjecture. For Collatz' Conjecture we cannot even evaluate an upper bound for the difficulty as the proof is not constructive.

4 Can computers create information?

Can computation produce new information? To answer this question we will introduce a measure of information based on counting bits encoding computations. The motivation for this complexity measure may be rooted in Leibniz's work (1686): "A *theory* must be simpler than the data is explains ". Hence, a bit-string for which there is no *theory* is "unexplainable", "incomprehensible" except as 'a thing in itself' (*Ding an sich* in Kant's terminology).

Bearing in mind these facts we say that if a self-delimiting Turing machine T with program p produces the bit-string x, then p generates x via T, and the amount of information T extracts from x is

$$H_T(x) = \min\{|p| \mid T(p) = x\}.$$

[2] The conjecture was tested up to 4×10^{17}, see [25].
[3] One of the Clay Mathematical Institute Millennium Problems, see [31, 17].
[4] See more in [23]; the conjecture was tested and proved true up to $10 \cdot 2^{58}$, see [26].

It is possible that $T(p) = x$ is false for any program p; in this case $H_T(x) = \infty$. This definition heavily depends on T, but using the universality theorem we can make H as independent as possible on the underlying Turing machine because H_U is optimal up to an additive constant in the class of all possible H_T:

For every self-delimiting Turing machine T there exists a constant c (depending on U and T) such that for all strings x:

$$H_U(x) \leq H_T(x) + c.$$

So, for now on we shall fix a universal self-delimiting Turing machine U and write H instead of H_U.

If $H(x) > H(y)$, then the complexity of x is larger than the complexity of y, that is, x encodes more information than y. In this framework, to create information means to start with an input x and produce an output y which has more information than x, that is, $H(x) < H(y)$. Our initial question becomes: is there any computable process capable of producing infinitely many outputs each of which has more information than its corresponding input? The problem is trivial for finitely many inputs.

One possible way to answer the above question is to assume that we have a one-to-one self-delimiting Turing machine T that halts on infinitely many inputs x, each of which having $H(x) \leq |x| - c/2$, where c is a fixed constant. Is it possible that T produces infinitely many outputs with the property that $H(T(x)) \geq |T(x)| + c/2$, that is, T produces a fixed amount (c bits) of newly created information[5])? The answer is *negative*: *no T is capable of such performance.* Indeed, this is not possible because otherwise T would generate an infinite computably enumerable set of strings y with $H(y) \geq |y| + c/2$, an impossibility (because the set $\{z \mid H(z) \geq |z| + c\}$ is immune, see [4]).

The above result suggests that a computer cannot create too much new information. Then the next question is: how much information can we expect to be created by computation?

A "Gödelian theory" is a finitely-specified, arithmetically sound, consistent theory strong enough to formalize arithmetic. For example, ZFC—Zermelo-Fraenkel set theory with choice, the classical axiomatic system in which virtually all current mathematics can be formalised—is a Gödelian theory. Define a new complexity measure

$$\delta(x) = H(x) - |x|.$$

The motivation in working with δ instead of H is the following. The complexity measures H and δ are similar as δ is defined from H and a simple computable function; for example, both measures are uncomputable. But H and δ differ in an *essential* way: given a positive N, the set $\{x \mid H(x) \leq N\}$ is finite while the set $\{x \mid \delta(x) \leq N\}$ is infinite. A sentence with a large δ-complexity has also a large H-complexity, but the converse is not true. For example, the H-complexity of (true) sentences of the form "$1 + n = n + 1$" tends to infinity as $n \to \infty$; however, their δ-complexity is bounded.

We can now state the main result in [9]:

For every Gödelian theory there exists a constant N, such that the theory proves no statement x with $\delta(x) > N$.

[5] This is a very small increase in information.

Any Gödelian theory can be used to prove theorems which have a bit more information than the theory itself, but not too much: everything is "hardwired" into the theory, there is very little room for "creativity" to produce more information.

The above result is a form of Gödel's incompleteness:

Any statement x with $\delta(x) > N$ cannot be proved by the Gödelian theory.

Even more, the set of statement which cannot be either proved or disproved by the Gödelian theory is large.

5 The algorithmic coding theorem

Shannon's coding theorem [27] says that the minimal average code string length is about equal to the entropy of the source string set. The coding theorem plays an important role in Shannon's information theory [27, 28, 16]. In what follows we will briefly present an algorithmic version of Shannon's coding theorem.

Self-delimiting Turing machines have a prefix-free domain. Prefix-free sets S satisfy Kraft's inequality, [16]:

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

The following (more general) converse result, known as *Kraft-Chaitin theorem* (see [4]), is frequently used to build self-delimiting Turing machines:

Given a computable list of "requirements" $(n_i, s_i), (s_i$ are strings, $n_i \geq 1)$ such that $\sum_i 2^{-n_i} \leq 1$, we can effectively construct a self-delimiting Turing machine T and a computable one-to-one enumeration $x_0, x_1, x_2, \ldots$ of strings x_i of length n_i such that $T(x_i) = s_i$, for all i, and $T(x)$ is undefined if $x \notin \{x_i \mid i \geq 1\}$.

Let Σ^* be the set of all binary strings. A function $P : \Sigma^* \to [0,1]$ such that $\sum_x P(x) \leq 1$ is called a *semi-distribution* over the strings. In case $\sum_x P(x) = 1$, P is a *distribution*. A semi-distribution P is semi-computable from below (above) in case the set $\{(x,r) \mid x \in \Sigma^*,\ r \in \mathbf{Q},\ P(x) > r\}$ $(\{(x,r) \mid x \in \Sigma^*,\ r \in \mathbf{Q},\ P(x) < r\})$ is computably enumerable ($\mathbf{Q}$ is the set of rationals). A semi-distribution P is computable if it is semi-computable from below and from above. For example, the probability[6] that the self-delimiting Turing machine T produces the output x,

$$P_T(x) = \sum_{T(u)=x} 2^{-|u|},$$

is a semi-distribution semi-computable from below. The function $P(x) = 2^{-2|x|-1}$ is a computable distribution.

A *prefix-code* for strings is a one-to-one function $C : \Sigma^* \to \Sigma^*$ such that $C(\Sigma^*)$ is prefix-free. If $C(x) = u$, then u is a code for x. The injectivity of C implies unique decodability.

For every self-delimiting Turing machine T and string x such that $P_T(x) > 0$, we denote by

$$x_T^* = \min\{u \mid T(u) = x\},$$

[6] See more about the underlying probability space in [4].

where the minimum is taken according to the quasi-lexicographical ordering of strings ($\lambda < 0 < 1 < 00 < 01 < 10 < 11 < 000 < \cdots$); x_T^* is called the *minimal (canonical) program* of x with respect to T. For every surjective self-delimiting Turing machine T, $C_T(x) = x_T^*$ is a prefix-code; universal machines are surjective.

The *average code-string length* of a prefix-code C with respect to a semi-distribution P is

$$L_{C,P} = \sum_x P(x) \cdot |C(x)|,$$

the *minimal average code-string length* with respect to a semi-distribution P is

$$L_P = \inf \{L_{C,P} \mid C \text{ prefix-code}\},$$

and the *entropy* of a semi-distribution P is

$$\mathcal{H}_P = -\sum_x P(x) \cdot \log P(x).$$

Shannon's classical *(noiseless) coding theorem* [27, 16] can be expressed in the language of semi-distributions as follows:

The following inequalities hold true for every semi-distribution P:

$$\mathcal{H}_P - 1 \leq \mathcal{H}_P + \left(\sum_x P(x)\right) \log \left(\sum_x P(x)\right) \leq L_P \leq \mathcal{H}_P + 1.$$

If P is a distribution, then $\log(\sum_x P(x)) = 0$, so we get the classical inequality $\mathcal{H}_P \geq L_P$, cf. [16]. However, this inequality is not true for every semi-distribution. For example, take $P(x) = 2^{-2|x|-3}$ and $C(x) = x_1x_1 \ldots x_nx_n01$. It follows that $L_P \leq L_{C,P} = \mathcal{H}_P - \frac{1}{4}$.

Under which conditions, given a semi-distribution P, can we find a (universal) self-delimiting Turing machine T such that $H_T(x)$ is equal, up to an additive constant, to $-\log P(x)$, i.e. the complexity is equal up to an additive constant to entropy? An answer is given by the following general result proved in [8]:

Assume that P is a semi-distribution such that $P(x) > 0$, for every x, and there exist a computably enumerable set $S \subset \Sigma^* \times \{0, 1, \ldots\}$ and a constant $c \geq 0$ such that the following two conditions are satisfied for every $x \in \Sigma^*$:

(i) $\sum_{(x,n) \in S} 2^{-n} \leq P(x)$,

(ii) *if $P(x) > 2^{-n}$, then $(x, m) \in S$, for some $m \leq n + c$.*

Then, there exists a machine T (depending upon S) such that for all x,

$$-\log P(x) \leq H_T(x) \leq (1 + c) - \log P(x).$$

The above result makes no direct computability assumptions on P. To get sharper consequences we will introduce the halting probability of a self-delimiting Turing machine T, Ω_T[7], and the minimal (canonical) programs with respect to T. First, in analogy with (1) we define

$$\Omega_T = \sum_{\{x \mid T(x)\,\text{halts}\}} 2^{-|x|}.$$

Specialising P we show that minimal programs are almost optimal for P. Minimal programs of universal machines are almost optimal for every semi-computable semi-distribution P:

Assume that P is a semi-distribution semi-computable from below. Then, there exists a machine T (depending upon P) such that for all x,

$$-\log P(x) \le H_T(x) \le 2 - \log P(x).$$

Consequently, minimal programs for T are almost optimal: the code C_T satisfies the inequalities:

$$0 \le L_{C_T,P} - \mathcal{H}_P \le 2.$$

When the semi-distribution P is given, an optimal prefix-code can be found for P. However, that code may be far from optimal for a different semi-distribution. For example, let C be a prefix-code such that $|C(x)| = 2^{|x|+2}$, for all x. Let $\alpha > 0$ and consider the distribution

$$P_\alpha(x) = (1 - 2^{-\alpha})\, 2^{-(\alpha+1)|x|}.$$

If $\alpha \le 1$, then

$$L_{C,P_\alpha} - \mathcal{H}_{P_\alpha} = \infty,$$

but if $\alpha > 1$, then

$$L_{C,P_\alpha} - \mathcal{H}_{P_\alpha} < \infty.$$

So, C is asymptotical optimal for every distribution P_α with $1 < \alpha$, but C is far away from optimality if $0 < \alpha \le 1$. Clearly, P_α is computable provided α is computable.

Minimal programs are asymptotical optimal for every semi-distribution semi-computable from below:

Let P be a semi-distribution semi-computable from below, and U a universal self-delimiting Turing machine. Then, there exists a constant c_P (depending upon P) such that

$$0 \le L_{C_U,P} - \mathcal{H}_P \le 1 + c_P.$$

The next result establishes a tight relation between complexity (H_T) and entropy ($-\log P_T$):

Let T be a machine and $c \ge 0$. The following statements are equivalent:

(a) *for all x, $H_T(x) \le (1 + c) - \log P_T(x)$,*

(b) *for all non-negative n, if $P_T(x) > 2^{-n}$, then $H_T(x) \le n + c$.*

In particular we get the *algorithmic coding theorem* (Chaitin–Gács):

There exists a constant $c \ge 0$ such that for all strings x,

$$|H_U(x) + \log P_U(x)| \le 1 + c.$$

[7] The reader may recall the number Ω_U introduce in section 1.

6 Algorithmic randomness and incompleteness

Defining randomness is very tricky. There are many proposals, among them the algorithmic one which equates randomness with incompressibility, and then proves other natural properties of "algorithmic randomness": stochasticity, unpredictability, etc. Algorithmic randomness comes into two forms, for finite bit-strings and for infinite sequences.

An infinite sequence $\mathbf{x} = x_1 x_2 \ldots, x_n \ldots$ is *algorithmically random* if there exists a positive constant $c > 0$ such that $H(x_1 x_2 \ldots, x_n) \geq n - c$. Chaitin's theorem (see [13]) states

The sequence of bits of Ω_U (i.e. the sequence $\omega_1 \omega_2 \ldots \omega_n \ldots$ in (2)) is algorithmically random.

We say that the real Ω_U is algorithmically random.

Two questions come naturally: a) are there any other "natural" algorithmically random sequences?, b) Ω_U is not only algorithmically random, but also computably enumerable, that is, Ω_U is the limit of a computable increasing sequence of rationals; are there other computably enumerable and algorithmically random numbers?

The answer to the first question is affirmative while the second question has a negative answer.

Let $\mathrm{bin} : \{1, 2, \ldots\} \to \Sigma^*$ be the bijection which associates to every $n \geq 1$ its binary expansion without the leading 1,

n	n_2	$\mathrm{bin}(n)$	$\lvert\mathrm{bin}(n)\rvert$
1	1	λ	0
2	10	0	1
3	11	1	1
4	100	00	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$

If $A \subset \Sigma^*$, then we define $\Upsilon[A] = \{n \geq 1 \mid \mathrm{bin}(n) \in A\}$. In other terms, the binary expansion of n is $n_2 = 1\mathrm{bin}(n)$. The *zeta number* of the Turing machine M,[8] denoted ζ_M, is defined by

$$\zeta_M = \sum_{n \in \Upsilon[\mathrm{dom}(M)]} \frac{1}{n}.$$

In [10] one proves the following result:

The zeta number ζ_U of a universal self-delimiting Turing machine U is algorithmically random.

In fact, the above theorem is true for a larger class of Turing machines. A *convergent Turing machine* is a Turing machine V whose zeta number is finite, $\zeta_V < \infty$.[9] Every self-delimiting Turing machine is convergent, but the converse is not true. The universality theorem holds true for convergent Turing machines as well. We can now state a more general result, [10]:

[8] M is not necessarily self-delimiting; of course, ζ_M could be infinite.

[9] Clearly, $\Omega_V < \infty$ iff $\zeta_V < \infty$, so convergence can be equally defined in terms of zeta or Omega.

The zeta number ζ_V of a universal convergent Turing machine V is algorithmically random.

The answer to the second question is provided by the following theorem (cf. [7, 22], see also [4]):

A real $\alpha \in (0,1)$ is computably enumerable and algorithmically random iff there exists a universal self-delimiting Turing machine U such that $\alpha = \Omega_U$.

Algorithmic randomness is intimately related to incompleteness in Gödel's sense. Here are two results:

Chaitin's theorem [13]*: Every Gödelian theory cannot determine more than* finitely many *digits of Ω_U.*

Solovay's theorem [29]*: Fix a Gödelian theory. We can construct universal self-delimiting Turing machines U such that the theory cannot determine* any *digit of Ω_U.*

Generalised Solovay's theorem [5]*: Fix a Gödelian theory and a universal self-delimiting Turing machine U. Assume that $\Omega_U = 0.11\cdots 10\cdots$. Then, we can effectively construct a universal self-delimiting Turing machine U' such that $\Omega_{U'} = \Omega_U$ and the theory can determine* at most the digits of Ω_U before the first 0*.*

7 Algorithmic randomness and halting

In this section we answer the question: Can a program stop at an algorithmically random time?

First we introduce yet another complexity measure, the natural complexity, cf. [11]. The *natural complexity* of the string x (with respect to the Turing machine[10] M) is $\nabla_M(x) = \min\{n \geq 1 \mid M(\mathrm{bin}(n)) = x\}$. Using ∇ the universality theorem has the following form:

One can effectively construct a (universal) Turing machine V such that for every machine M, there is a constant $\varepsilon > 0$ (depending on V and M) such that $\nabla_V(x) \leq \varepsilon \cdot \nabla_M(x)$, for all strings x.

We fix the universal Turing machine V and write ∇ instead of ∇_V. A binary string x is *algorithmically random* if $\nabla(x) \geq 2^{|x|}/|x|$.[11] One can prove (see [15, 4]) that algorithmically random strings have many properties one naturally associate with randomness, among them strong uncomputability:

No Turing machine is capable of enumerating an infinity of algorithmically random strings.

[10] Not necessarily self-delimiting.

[11] In the language of the complexity H, the string x is algorithmically random if $H(x) \geq |x| - \log|x|$. Algorithmically randomness for strings is a matter of degree, so we can set various bounds on the complexity; see [4].

Most binary strings of a given length n are algorithmically random because they have high density:

$$\text{density}(n) = \#\{x \in \Sigma^* : |x| = n, \nabla(x) \geq 2^n/n\} \cdot 2^{-n} \geq 1 - 1/n,$$

hence

$$\lim_{n\to\infty} \text{density}(n) = 1.$$

We are interested in the properties of the exact times programs stop. A time t will be called *algorithmically random* if $\text{bin}(t)$ is algorithmically random. In [12] one proves the following result:

Let V be a universal Turing machine. One can effectively compute a constant c (depending on V) such that the following is true: if an N-bit program p has not stopped on V by the time $2^{2N+2c+1}$, where $N \geq 2$, then $V(p)$ cannot exactly stop at any algorithmically random time $t \geq 2^{2N+2c+1}$.

In other words, given V and a program p of length N we can compute the time $\theta_{V,N} = 2^{2N+2c+1}$ with the following property: either $V(p)$ stops before the time $\theta_{V,N}$, or if it has not stopped by that time, then either $V(p)$ will never stop or $V(p)$ will stop at a non-algorithmically random time $t \geq \theta_{V,N}$. Because non-algorithmically random times have effectively zero density, "chances" that an N-bit program p that has not stopped on V by the time $\theta_{V,N}$ will eventually stop effectively approach zero:

For every length N, we can effectively compute a threshold time $\theta_{V,N}$ (which depends on V and N) such that if a program of length N runs for $\theta_{V,N}$ steps without halting, then the density of times greater than $\theta_{V,N}$ at which the program can stop has effective zero density. More precisely, if an N-bit program runs for $T > \max\{\theta_N, 2^{2+5\cdot 2^k}\}$ steps, then the density of times at which the program can stop is less than 2^{-k}.

8 Incompleteness and uncertainty

Gödel's hostility to any suggestion regarding possible connections between his incompleteness theorem and physics, particularly, Heisenberg's uncertainty relation, is well-known: J. Wheeler was thrown out of Gödel's office for asking the question "Professor Gödel, what connection do you see between your incompleteness theorem and Heisenberg's uncertainty principle?"

Still, there is a huge interest in the relations between these two statements. For example, Hawking's view (see [20]) is that

"a physical theory is self-referencing, like in Gödel's theorem ... Theories we have so far are both inconsistent and incomplete".[12]

In [12] a relation between incompleteness and uncertainty is established. To present it we will use the natural complexity $\nabla = \nabla_U$ induced by a universal self-delimiting Turing machine U; recall that $H = H_U$. One can see that

$$2^{H(x)} \leq \nabla(x) < 2^{H(x)+1},$$

[12] It is worth noting that a theory which is inconsistent is not necessarily complete, although in many cases this is true.

hence, $\Delta(x) = 2^{H(x)}$, the uncertainty in the value $\nabla(x)$, is the difference between the upper and lower bounds given.

Finally let $\Delta_s = 2^{-s}$. The property of $\Omega = \Omega_U$ to be algorithmically random can be expressed in the following way:

$$\Delta_s \cdot \Delta(\omega_1 \ldots \omega_s) \geq 1, \tag{3}$$

In (3), an *uncertainty relation*, the complexity measures the uncertainty in the total information. One can prove that the relation (3) *implies* Chaitin's theorem (presented at the end of section 6), hence, Gödel's incompleteness.

Of course, this is a formal approach and much more is required to check its "physical" base (see more in [12]).

References

1. K. Ambos-Spies, C. S. Calude, W. Merkle, L. Staiger. *On Universal Computably Enumerable Prefix Codes*, in preparation.
2. Y. Bar-Hillel (ed.). *Language and Information: Selected Essays on Their Theory and Application*, Addison-Wesley, Reading, Mass, 1964.
3. J. Barwise and J. Seligman. *Information Flow: The Logic of Distributed Systems*, Cambridge University Press, Cambridge, 1997.
4. C. S. Calude. *Information and Randomness: An Algorithmic Perspective*, 2nd Edition, Springer-Verlag, Berlin, 2002.
5. C. S. Calude. Chaitin Ω numbers, Solovay machines and incompleteness, *Theoret. Comput. Sci.* 284 (2002), 269–277.
6. C. S. Calude, Elena Calude, M. J. Dinneen. A new measure of the difficulty of problems, *Journal for Multiple-Valued Logic and Soft Computing* 10 (2006), 1–21.
7. C. S. Calude, P. Hertling, B. Khoussainov, Y. Wang. Recursively enumerable reals and Chaitin Ω numbers, *Theoret. Comput. Sci.* 255 (2001), 125–149.
8. C. S. Calude, H. Ishihara, T. Yamaguchi. Minimal programs are almost optimal, *International Journal of Foundations of Computer Science* 12, 4 (2001), 479–489.
9. C. S. Calude, H. Jürgensen. Is complexity a source of incompleteness? *Advances in Applied Mathematics* 35 (2005), 1–15.
10. C. S. Calude, M. A. Stay. Natural halting probabilities, partial randomness, and Zeta functions, *Information and Computation*, 204 (2006), 1718–1739.
11. C. S. Calude, M. A. Stay. From Heisenberg to Gödel via Chaitin, *International Journal of Theoretical Physics* 44, 7 (2005), 1053–1065.
12. C. S. Calude, M. A. Stay. Most Programs Stop Quickly or Never Halt, *CDMTCS Research Report* 284, 2006, 17 pp.
13. G.J. Chaitin. A theory of program size formally identical to information theory, *J. Assoc. Comput. Mach.* 22 (1975), 329–340.
14. G.J. Chaitin. *Information, Randomness and Incompleteness, Papers on Algorithmic Information Theory*, World Scientific, Singapore, 1987. (2nd ed., 1990)
15. G. J. Chaitin. *Algorithmic Information Theory*, Cambridge University Press, Cambridge, 1987.
16. T. M. Cover and J. A. Thomas. *Elements of Information Theory*, Wiley, New York, 1991.
17. K. J. Devlin. *The Millennium Problems*, Basic Books, New York, 2002.
18. K. J. Devlin. *Logic and Information*, Cambridge University Press, Cambridge, 1991.
19. L. Floridi. What is the philosophy of information? *Metaphilosophy* 33(1-2) (2002), 123–145.

20. S. W. Hawking. Gödel and the End of Physics, *Dirac Centennial Celebration*, Cambridge, UK, July 2002, `http://www.damtp.cam.ac.uk/strtst/dirac/hawking/`.
21. J. Kohlas. *Information Algebras: Generic Structures for Inference*, Springer-Verlag, London, 2003.
22. A. Kučera, T. A. Slaman. Randomness and recursive enumerability, *SIAM J. Comput.*, 31, 1 (2001), 199–211.
23. J. C. Lagarias. The $3x + 1$ problem and its generalizations, *Amer. Math. Monthly* 92 (1985), 3–23.
24. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, 2000.
25. T. Oliveira e Silva. Goldbach Conjecture verification, `http://www.ieeta.pt/~tos/goldbach.html`, June 5, 2006.
26. T. Oliveira e Silva. Computational verification of the 3x+1 conjecture, `http://www.ieeta.pt/~tos/3x+1.html`, May 26, 2006.
27. C. E. Shannon. A mathematical theory of communication, *Bell System Technical Journal* 27(1948), 379–423, 623–656.
28. C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, Illinois, 1949 (paperback edition 1963; special fiftieth anniversary edition in 1999).
29. R.M. Solovay. A version of Ω for which ZFC can not predict a single bit, in C.S. Calude, G. Păun (eds.). *Finite Versus Infinite. Contributions to an Eternal Dilemma*, Springer-Verlag, London, 2000, 323–334.
30. T. Stonier. *Information and Meaning: An Evolutionary Perspective*, Springer, Heidelberg, 1997.
31. `http://www.claymath.org/millennium/Riemann_Hypothesis/`.