Physica A 586 (2022) 126458

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Chaitin's Omega and an algorithmic phase transition

Christof Schmidhuber

Zurich University of Applied Sciences, School of Engineering, Technikumstrasse 9, 8401 Winterthur, CH, Switzerland

ARTICLE INFO

Article history:
Received 8 September 2020
Received in revised form 18 July 2021
Available online 1 October 2021

Keywords:
Chaitin's Omega
Complexity
Turing machine
Algorithmic thermodynamics
Phase transition
String theory

ABSTRACT

We consider the statistical mechanical ensemble of bit string histories that are computed by a universal Turing machine. The role of the energy is played by the program size. We show that this ensemble has a first-order phase transition at a critical temperature, at which the partition function equals Chaitin's halting probability Ω. This phase transition has curious properties: the free energy is continuous near the critical temperature, but almost jumps: it converges more slowly to its finite critical value than any computable function. At the critical temperature, the average size of the bit strings diverges. We define a non-universal Turing machine that approximates this behavior of the partition function in a computable way by a super-logarithmic singularity, and discuss its thermodynamic properties. We also discuss analogies and differences between Chaitin's Omega and the partition function of a quantum mechanical particle, and with quantum Turing machines. For universal Turing machines, we conjecture that the ensemble of bit string histories at the critical temperature has a continuum formulation in terms of a string theory.

1. Introduction

In 1975, G. Chaitin [1] introduced a constant associated with a given universal Turing machine U [2] that is often called the "halting probability" Ω. It is computed as a weighted sum over all prefix-free input programs p for U that halt:

$$\Omega_U = \sum_{\text{halting p(U)}} 2^{-l(p)} = \sum_{l=1}^{\infty} N(l)\, 2^{-l} \tag{1}$$

where p is a "program" (a bit string made up of 0's and 1's), $l(p)$ is its length (the number of bits), and $N(l)$ is the number of prefix-free programs of length l for which U halts. Turing machines and prefix-free bit strings are briefly reviewed in Section 2 and in the Appendix. For a general introduction to information theory, see [3].

Most of the discussion around Ω has focused on its first few digits, which are determined by the function $N(l)$ for small program length l. As every mathematical hypothesis can be translated into a halting problem (the question whether a given program halts for a given Turing machine), many long-standing mathematical problems could be solved if only one could compute Ω digit by digit. Unfortunately, Ω cannot be computable by any halting program, precisely because knowing Ω would imply that one could decide mathematical problems that are known to be undecidable in the sense of Gödel's incompleteness theorem [4]. Moreover, even when the first few digits of Ω are computable for a given universal Turing machine U, they are not universal: they depend on the choice of U.

In this note, we will therefore not be concerned with the contribution of short programs to Ω, nor will we dwell much on the issues of incompleteness and undecidability. Instead, we will focus on the contribution of very long programs to

E-mail address: christof@schmidhuber.ch.

https://doi.org/10.1016/j.physa.2021.126458

Ω, i.e., on the behavior of $N(l) \cdot 2^{-l}$ as $l \to \infty$. More precisely, following [5–7], in a generalization of (1), we consider the statistical mechanical ensemble of bit string histories with partition function

$$Z_U(\beta) = \sum_{\text{halting p(U)}} \exp\{-\beta \cdot l(p)\} \qquad \text{with} \qquad \beta = \frac{1}{kT}, \tag{2}$$

where k is the Boltzmann constant and T the temperature as usual in statistical mechanics (see, e.g., [8] for a review of statistical mechanics and field theory). We will study Z as a function of $\beta = \beta_c + \epsilon$ in the vicinity of the "Chaitin point" $\beta_c = \ln 2$ with $\epsilon \ll 1$ (assuming a binary alphabet; for general Turing machines, β_c is the log of the alphabet size).

We find that the "Chaitin point" $\beta = \beta_c = \ln 2$ corresponds to a critical temperature, at which a first-order phase transition occurs. This phase transition has very curious properties. In particular, the free energy is almost discontinuous: it converges more slowly to its finite critical value than any computable function. However, in a finite universe with limited computation time, this effective discontinuity is invisible. We illustrate this type of transition in a toy model, namely a non-universal Turing machine (the "counting machine") that approximates this behavior of the free energy by a super-logarithmic singularity.

At the critical point, we find that the average size of the output bit strings diverges. This leads us to the fascinating question whether there is a continuum formulation of our bit string ensemble at the Chaitin point in terms of some string theory [9], in which the two-dimensional string world-sheet is spanned by the bit string and the computation time. As outlined in the conclusion, such a potential link between algorithmic information theory and string theory could shed new light on both theories.

To set the stage for investigating this further, we point out formal analogies between ensemble (2) at the critical point and (i) a quantum mechanical relativistic particle, and (ii) the theory of random surfaces. We also interpret the ensemble (2) as a probabilistic Turing machine and show how it can be extended it to a quantum Turing machine.

2. Relation to previous work

The generalization (2) of Chaitin's halting probability has previously been studied by Tadaki [5], who investigated the degree of randomness of the real number $Z_U(\beta)$, written in binary form. The relation with statistical mechanics was pointed out by Calude and Stay [6], who also discuss variants of (2), in which the sum runs over general (as opposed to only prefix-free) programs (the partition function then diverges at $\beta = \ln 2$, instead of converging to Chaitin's Ω). The statistical mechanical approach was formulated more mathematically by Tadaki in [10].

Baez and Stay [7] define the corresponding "algorithmic" versions of the specific heat and other thermodynamic quantities. Moreover, they formally extend the Gibbs factor (2) by including two other terms, corresponding to the logarithm $E(p)$ of the computation time of the Turing machine, and to the expectation value $N(p)$ of the output bit string (interpreted as a natural number in binary form):

$$\exp\{-\beta_1 \cdot l(p) - \beta_2 \cdot E(p) - \beta_3 \cdot N(p)\}. \tag{3}$$

Algorithmic versions of the Carnot cycle are also discussed in [7], where it is pointed out that the partition function has a singularity at $\beta_1 = \ln 2$, $\beta_2 = \beta_3 = 0$. Tadaki [11,12] discuss computational aspects of this "algorithmic phase transition".

Our paper complements this previous work by studying the nature of this algorithmic phase transition from a more physical point of view. In particular, a key question about phase transitions is, whether they are first-order or second-order. As mentioned, we resolve this in Sections 7 and 8 by showing that this one is an exotic first-order transition.

The sum (2) over p can be re-written in terms of a sum over all output bit strings B:

$$Z_U(\beta) = \sum_{\text{B}} \exp\{-\beta \cdot K_\beta(B)\} \tag{4}$$

$$\text{with} \quad K_\beta(B) = -\frac{1}{\beta} \ln \sum_{p(B)} \exp\{-\beta \cdot l(p)\} \quad \to \quad K(B) \quad \text{as} \quad \beta \to \infty. \tag{5}$$

Here, the second sum runs over all programs $p(B)$ that halt and produce B. $K(B)$ is the Kolmogorov complexity, i.e., the length of the shortest program that makes U compute B [13]. For general bit strings, $K(B)$ is not computable by any halting program. At the critical point $\beta = \beta_c$, $\exp\{-\beta_c K_{\beta_c}(B)\}$ is the algorithmic Solomonoff probability [14].

According to Levin's coding theorem [15], $K_{\beta_c}(B) \leq K(B) + c$ with a constant c that does not depend on B. We also have $K(B) < K_{\beta_c}(B)$, because the sum in (5) runs over all programs that produce the output bit string B, not only the shortest one. In recent work, Kolchinsky and Wolpert [16] interpret the differences

$$C(B) = K_{\beta_c}(B) - K(B) \quad , \quad Q(p(B)) = l(p(B)) - K(B)$$

in terms of the minimum heat that is released into the environment by the computation in the sense of [17]. In [16], properties of this generated heat are discussed both for the "coin flipping realization" (1) of the Turing machine U and for other realizations.

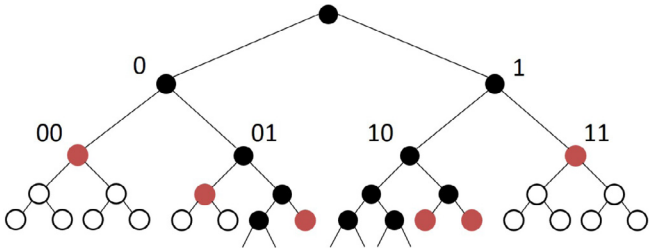

Fig. 1. Tree representation of prefix-free bit strings.

In a separate line of work (see [18–20], and references therein), Manin considers a self-adjoint Hamiltonian H acting on the Hilbert space spanned by the output bit strings $|B\rangle$, viewed as states. If $\beta = it$ is interpreted as imaginary time, the operator $\exp\{iHt\}$ describes a unitary time evolution. H is defined such that $|B\rangle$ are its eigenvectors with eigenvalues $K(B)$. Manin relates this model to error-correcting codes, to Zipf's law, and to renormalization in field theory (I thank D. Murfet for pointing this out to me).

Inspired in part by this work, in Section 9 we formulate ensemble (2) as a probabilistic Turing machine that runs in a different time τ, defined as the position of the head on the read-only program tape. Starting with a blank work tape, the final state is a superposition of output bit strings. While this time evolution is irreversible, and therefore the transfer matrix is not unitary, it can be extended to act on halting states such that it asymptotically approaches Manin's unitary time evolution, as well as generalizations thereof, after an initial phase of heat production.

One of the many other interesting issues that have been addressed in the context of Turing machines and Hamiltonian formulations is that in some cases it is an undecidable problem whether or not the energy spectrum has a vanishing mass gap [21,22].

3. Turing machines and prefix-free programs

We follow Chaitin's definition of a Turing machine, which is reviewed in the Appendix. A bit string y is called a prefix of a bit string x, if x can be written as a concatenation $x = yz$, with a third bit string z. A set of bit strings is called prefix-free, if no bit string is a prefix of another. For the current argument, it is sufficient to think of a Turing machine T as a map ("computation") from a set P of prefix-free input bit strings $p \in P$ ("programs"), for which the computation halts, to the set O of arbitrary output bit strings of any length:

$$T : p \in P \to T(p) \in O$$

The output bit strings are written on a "work tape" that extends infinitely in both directions. The computation manipulates them until it halts. The prefix-free input bit strings are written on a finite read-only "program tape" (see the Appendix for details). The prefix-free input programs p are what we sum over in (2), and whose lengths l play the role of the energy in the Boltzmann factor.

One may represent a set X of prefix-free bit strings by a tree (Fig. 1). The vertices in the $(l+1)$th line (or lth generation) of the graph represent the 2^l binary numbers b_l with l digits. Branching to the left appends a 0, branching to the right appends a 1 at the end of b_l to yield the next generation of b_{l+1}. At each vertex, the corresponding number b_l is either added to the set $X_l \subset X$ of prefix-free programs p_l of size l (red dots) or not (black dots). Black dots are prefixes (parents, grand-parents, ...) of red dots and give birth to two children; we assume that each black dot is the prefix of at least one red dot. Red dots have no children. In the figure, white dots represent bit strings that are never born.

Let n_l be the number of red dots (prefix-free programs) of length l. Let m_l be the number of black dots (prefixes) of length l. Let $w_l = 2^l - n_l - m_l$ be the number of white dots of length l. We define the percentages $Q_l = w_l \cdot 2^{-l}$ of white dots and $P_l = n_l \cdot 2^{-l}$ of red dots in the lth generation and get

$$P_l = Q_{l+1} - Q_l \quad \text{with} \quad \lim_{l\to\infty} Q_l = \sum_{l=1}^{\infty} P_l = \sum_{p_l \in X} 2^{-l} = 1, \tag{6}$$

where the last equation states that "Kraft's inequality is satisfied with equality" (see [3]). As an example of a set of prefix-free programs, consider "Fibonacci coding": a child is a member of X, if its last 2 digits are "1" or – in a slight generalization – if its last N digits are "1". In this case, one easily verifies that P_l falls off exponentially as $l \to \infty$.

For a given Turing machine T, there are two kinds of red dots: $\tilde{n}_l$ halting programs and $n_l - \tilde{n}_l$ non-halting programs. We denote by $h_l = \tilde{n}_l / n_l$ the fraction of programs in the lth generation that halt. Then the partition function (2) can be written as

$$Z_U(\beta) = \sum_{l=1}^{\infty} P_l \cdot h_l \, e^{-\epsilon \cdot l} \quad \text{with} \quad \epsilon = \beta - \beta_c \ , \ \beta_c = \ln 2. \tag{7}$$

4. Universality of the singularity

At the critical point $\beta = \beta_c = \ln 2$, our partition function (7) is Chaitin's Ω(1):

$$Z_U(\beta_c) = \sum_{l=1}^{\infty} P_l \cdot h_l = \Omega < 1$$

For $\beta < \beta_c$, the partition function diverges, as long as $P_l \cdot h_l$ falls off more slowly than exponentially as $l \to \infty$, which is the case for any universal Turing machine U (see below). How exactly does Z_U approach Ω as β approaches β_c from above? Let us first discuss in how far this singularity near $\beta = \beta_c$ is universal, i.e., independent of U.

A universal Turing Machine ("UTM") U is one that can simulate any other Turing machine T_i in the following sense: there is a finite bit string ("translator program") c_i such that for each program p, $U(c_i p) = T_i(p)$. I.e., if p makes T_i compute an output bit string, the concatenation $c_i p$ makes U compute the same output bit string. Let C_i be the finite length of the program c_i. Then the partition function $Z_U(\beta)$ of the UTM contains the partition function $Z_i(\beta)$ of T_i as a subset:

$$Z_U(\beta) \geq e^{-\beta C_i} \cdot Z_i(\beta)$$

This applies to all Turing machines T_i. Thus, as $\epsilon = \beta - \beta_c \to 0$, the Turing machine T_i with the strongest singularity (i.e., with the largest derivative $Z_i'(\epsilon)$ at $\epsilon \sim 0$) dominates the singularity of the partition function $Z_U(\beta)$ at $\beta = \ln 2$. As this applies to all U, we conclude that this singularity is universal, i.e., independent of the choice of the UTM, up to an overall pre-factor 2^{-C_i}.

Our ensemble (2) includes only programs that halt. This makes it intractable, as it is generally an undecidable question whether a given Turing machine halts for a given program. Thus, the factor h_l in (7), Chaitin's Ω, and the partition function $Z_U(\beta)$ are actually not computable by any halting program. These issues around un-decidability and non-computability, fascinating as they may be, will not play a major role here. It is clear from (7) that the strongest singularity in ϵ corresponds to the product $P_l \cdot h_l$ that decays most slowly as $l \to \infty$. Thus, the non-computable factor h_l can only make this singularity weaker. We will therefore first discuss non-universal Turing machines T_i for which all programs halt (i.e. $h_l = 1$), and then return to universal Turing machines towards the end.

As an example of a function P_l that converges more slowly than that from Fibonacci coding, let the N of Fibonacci coding grow with the program length: $N(l) = \text{int}(1 + \lg l)$, where $\lg \equiv \log_2$. In this case, it is not difficult to see that P_l decays like a power of l:

$$P_l \propto l^{-\alpha} \text{ as } l \to \infty \text{ with } \alpha > 1 \quad \Rightarrow \quad 1 - Z(\epsilon) \propto \epsilon^{\alpha - 1} \text{ as } \epsilon = \beta - \beta_c \to 0$$

Next, we present a machine that yields a much stronger, super-logarithmic singularity.

5. The counting machine

We now define a Turing machine T_0 that we call the "counting machine", corresponding to a particular set of prefix-free programs, that always halts. We will then show that its partition function (7) has a computable, super-logarithmic singularity that, for our purposes, serves as a good model of the singularity of UTM's.

Let us first describe the output of the machine T_0. Given any infinite input bit string p on the program tape, T_0 writes a number N of 1's in a row on its otherwise blank work tape and then halts. We call p_N the prefix of p consisting only of those bits of p that have been read by the time the machine halts, i.e., the machine halts on the last bit of p_N. This defines a set P of prefix-free input programs $p_N \in P$. We will construct T_0 such that any number $N \in \mathbb{N}_0$ of 1's appears as the output bit string of exactly one such p_N, namely:

$$\begin{aligned} &\text{for } N < 3: \quad p_0 = 00, \; p_1 = 01\,, \; p_2 = 10 \quad \text{with length} \quad l_N = 2 \\ &\text{for } N = 3: \quad p_3 = 110 \quad \text{with length} \quad l_N = 3 \\ &\text{for } N > 3: \quad p_N = 11 n_2 \ldots n_k N 0 \quad \text{with length} \quad l_N = 6 + n_2 + \cdots + n_k \end{aligned} \tag{8}$$

where n_k is the binary length of N, n_{k-1} is the binary length of n_k, and so on, until a length $n_1 = 3 = 11_2$ is reached. For $N > 2$, p_N begins with "11" and ends with "0". The number of iterations k can be recursively expressed as follows:

$$k(N) = \begin{cases} 0 \text{ if } N < 4 \\ 1 + k(1 + \lg N) \text{ if } N \geq 4 \end{cases} \tag{9}$$

This yields, e.g., $k(4) = k(7) = 1$, $k(8) = k(127) = 2$, $k(128) = 3$, and so on.

Next, we describe how T_0 reconstructs N from p_N. Given an infinite string p on the program tape, T_0 proceeds as follows:

1. T_0 reads the first two digits $n_1 = p_1 p_2$ of p. If $n_1 = 00_2$, T_0 leaves the work tape blank and halts; if $n_1 = 01_2$, T_0 writes 1 on the work tape and halts; if $n_1 = 10_2$, T_0 writes 11 on the work tape and halts. If $n_1 = 11_2$, T_0 reads the next digit p_3 and defines the new integer $m_1 = 3$.

2. If $p_{m_1} = p_3 = 0$, T_0 writes $1^{n_1} = 111$ on the work tape, then halts. If $p_{m_1} = p_3 = 1$, T_0 reads the next $n_1 = 3$ digits $p_{m_1+1}, \ldots, p_{m_1+n_1}$ of p, i.e., p_4, p_5, p_6. T_0 defines $m_2 = m_1 + n_1 = 6$ and $n_2 = p_3p_4p_5$ (the concatenation with p_3 but without p_6)

   ⋮

i. In the ith step, if $p_{m_{i-1}} = 0$, T_0 writes $1^{n_{i-1}}$ on the work tape and halts. If $p_{m_{i-1}} = 1$, T_0 reads in the next n_{i-1} digits. It defines $m_i = m_{i-1} + n_{i-1}$ and the concatenation $n_i = p_{m_{i-1}} \ldots p_{m_i-1}$ and moves on to step $(i + 1)$, until T_0 halts. If T_0 halts in the ith step, then i is related to k of (9) by $k = i - 2$, and $N = n_{i-1}$.

   E.g., in the third step, $p_3 = 1$ and $m_2 = 6$. Suppose, $n_2 = p_3p_4p_5 = 101_2 = 5$. If the sixth digit p_6 of p is 0, T_0 writes a sequence of $n_2 = 5$ 1's on the tape and halts. In this case, $k = 1$ and $N = 5$. However, if $p_6 = 1$, T_0 reads in the next five digits $p_7 \ldots p_{11}$, defines $m_3 = m_2 + n_2 = 11_{10}$ and $n_3 = p_6 \ldots p_{10}$, and moves on to step 4.

As an example, consider the input bit string 11100110100. Then $n_1 = 11_2$, so in step 2, T_0 defines $m_2 = 6, n_2 = 100_2 = 4$. In step 3, since $p_6 = 1$, T_0 sets $m_3 = 10$ and reads in the 4-digit number $n_3 = 1101_2 = 13$. In step 4, since the next digit p_{10} is a 0, T_0 writes $N = 13$ digits 1 in a row and halts. Only the first 10 digits 1110011010 of the input bit string constitute an element of P. More generally, if the counting machine halts in step k, the first m_{k-1} digits of the input string constitute an element of P. The first elements are:

$$P = \{00, 01, 10, 110, 111000, 111010, 111100, 111110, 1110010000, \ldots\}$$

One may verify that any number N of 1's in a row appears as the output bit string of exactly one program $p_N \in P$, as claimed above. It is also clear that P is complete in the sense that it cannot be enlarged by any additional bit string without spoiling its property of being prefix-free. As a result, (6) implies that $Z(\beta_c) = 1$.

Although the counting machine T_0 only produces bit strings that are trivial in the sense that they contain only 1's, variants of the counting machine can be used to make them less trivial in subsequent steps. E.g., one variant may generate all integers m in binary form, such as $m = 20 = 10100_2$, and then overwrite the 1's by repeating m until the bit string ends: "1010010100...". Another variant may generate all integers K, and then create K "kinks" on the bit strings resulting from step 2, by flipping all bits after certain points on the strings. Thus, the counting machine is also a useful tool for systematically generating nontrivial output bit strings of increasing complexity.

More generally, the counting machine T_0 can be used whenever one needs a highly compact specification of large numbers N by prefix-free programs. Of course, other sets of prefix-free programs may give a shorter description of *individual* large numbers, such as $2^{2^{1024}}$, at the expense of the *average* large number.

Appendix A.4 presents a concrete implementation of the counting machine T_0.

6. Super-logarithmic singularity

In this section, we compute how the partition function (7) approaches its critical value as β approaches β_c from above in the case of the counting machine. The counting machine halts for every input program ($h_l = 1$) and therefore has a computable partition function

$$\hat{Z}(\beta) = \sum_{\text{all p}} \exp\{-\beta \cdot l(p)\} = \sum_{k=0}^{\infty} \hat{Z}_k(\beta) \qquad \text{with} \qquad \hat{Z}(\beta_c) = 1, \tag{10}$$

where $\hat{Z}_k(\beta)$ is the contribution from programs p that halt after k iterations, k being defined in (9). Using (8), we expand:

$$\begin{aligned}
\hat{Z}_0(\beta) &= 3e^{-2\beta} + e^{-3\beta} \quad , \quad \hat{Z}_1(\beta) = 4e^{-6\beta} \\
\hat{Z}_2(\beta) &= 8e^{-10\beta} + 16e^{-11\beta} + 32e^{-12\beta} + 64e^{-13\beta} \\
\hat{Z}_k(\beta) &= \sum_{n_2,\ldots,n_k,N} e^{-\beta\cdot(6+n_2+\cdots+n_k)} \\
&\sim \sum_{n_2,\ldots,n_k} 2^{-(6+n_2+\cdots+n_{k-1})} \cdot \frac{1}{2}\, e^{-\epsilon n_k} \quad \text{with} \quad \epsilon = \beta - \beta_c,
\end{aligned} \tag{11}$$

where n_2 runs from 4 to 7, n_{i+1} runs from 2^{n_i-1} to $2^{n_i} - 1$, and N runs from 2^{n_k-1} to $2^{n_k} - 1$. In the last line, we have expanded near $\beta = \beta_c = \ln 2$, and kept only the leading term in ϵ, noting that $n_k \gg n_{k-1}$. For a given k, let Λ_k be the largest possible value of n_k:

$$\Lambda_1 = 3, \quad \Lambda_2 = 7, \quad \Lambda_3 = 127, \quad \Lambda_{k+1} = 2^{\Lambda_k} - 1. \tag{12}$$

If $\Lambda_{k-1} \gg 1/\epsilon$, we can approximate $\hat{Z}_k$ in (11) by 0, since the minimum value of n_k is $\Lambda_{k-1} + 1$. On the other hand, if $\Lambda_k \ll 1/\epsilon$, we can approximate ϵ by 0 in $\hat{Z}_k$. This yields $\hat{Z}_0 = 7/8, \hat{Z}_1 = 1/16$. Noting that there are always $2^{n_i}/2$ possible values for n_{i+1}, in the case $\Lambda_k \ll 1/\epsilon$ we can iteratively perform the sum over $n_2, \ldots, n_k$ for $k > 1$ to obtain

$$\hat{Z}_k = \frac{1}{2} \sum_{n_2,\ldots,n_k} 2^{-6-n_2-\cdots-n_{k-1}} = \frac{1}{4} \sum_{n_2,\ldots,n_{k-1}} 2^{-6-n_2-\cdots-n_{k-2}} = \cdots = \frac{1}{2^{k+3}}$$

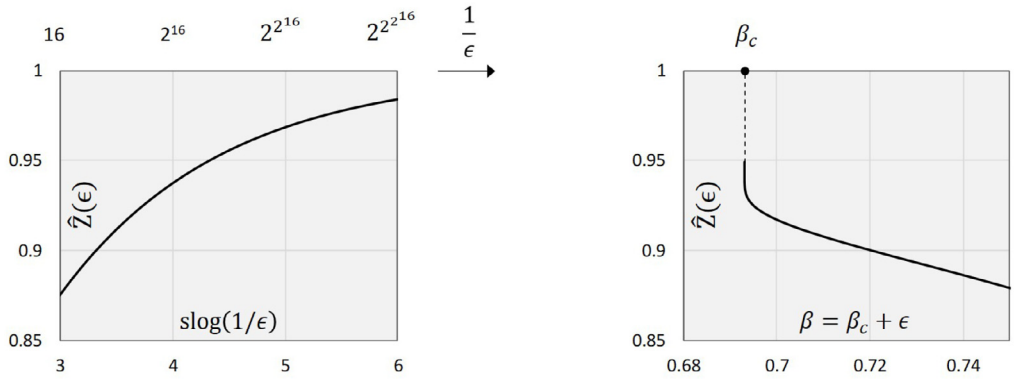

Fig. 2. $\hat{Z}(\epsilon)$ as a function of $1/\epsilon$ (left) and β (right).

We now perform the sum (10) over k and first consider the (rare) case where $1/\epsilon = \Lambda_K$ for some K. In Appendix A.5, it is shown that, in this case, $\hat{Z}_K = 2^{-K-3}$, $Z_{K+1} = 0$ to high accuracy already for $K \geq 4$. Thus,

$$\hat{Z}(\epsilon) = \frac{7}{8} + \frac{1}{16} + \sum_{k=2}^{K} \hat{Z}_k = 1 - 2^{-K-3} \quad \text{with} \quad \frac{1}{\epsilon} = \Lambda_K \tag{13}$$

The singularity in ϵ comes from the dependence of K on ϵ. To continue (13) to general ϵ, we use the "super-logarithm" $\text{slog}_2(x)$ with basis 2 in the so-called "linear approximation":

$$\text{slog}_2(x) = \begin{cases} x - 1 & \text{if } 0 < x \leq 1 \\ \text{slog}_2(\lg(x)) + 1 & \text{if } x > 1 \end{cases} \tag{14}$$

Its integer values are $\text{slog}_2(1) = 0$, $\text{slog}_2(2) = 1$, $\text{slog}_2(4) = 2$, $\text{slog}_2(2^x) = \text{slog}_2(x)+1$. Real values of slog_2 are interpolated from its integer part $\lg^-(x) = \text{int}(\text{slog}_2(x))$ by

$$\text{slog}_2(x) = \lg^-(x) + \lg \ldots \lg x \quad \text{with } 1 + \lg^-(x) \text{ iterations}$$

We can now express $K(\epsilon)$ in terms of the super-logarithm by noting from (12) that $\text{slog}_2(\Lambda_{k+1}) \to \text{slog}_2(\Lambda_k) + 1$ to very high accuracy already for $k > 2$: $\text{slog}_2(\Lambda_1) = 1 + \lg\lg 3 \sim 1.66$, $\text{slog}_2(\Lambda_2) = 2 + \lg\lg\lg 7 \sim 2.57$, $\text{slog}_2(\Lambda_k) = k + 0.57$

$$\begin{aligned} \Rightarrow \quad & K(\epsilon) \sim \text{slog}_2(1/\epsilon) - \phi \quad \text{with} \quad \phi = 0.57\ldots \\ & \hat{Z}(\epsilon) \sim 1 - \lambda \cdot 2^{-\text{slog}_2(1/\epsilon)} = 1 - \lambda \cdot 2^{-\lg^-(1/\epsilon)} \cdot \{\lg \ldots \lg(1/\epsilon)\}^{-1}, \end{aligned} \tag{15}$$

where $\lambda = 2^{\phi-3} \sim 0.186$, and there are $\lg^-(1/\epsilon)$ iterations of the logarithm in the last line.

This continues (13) to any ϵ. Although the continuation (14) of the super-logarithm to real values, and thus the continuation (15) of $\hat{Z}(\epsilon)$, is not unique, different continuations differ only by sub-leading orders in ϵ. Thus, (15) is the leading singularity of the partition function $\hat{Z}(\epsilon)$ at the critical point. This partition function is plotted in Fig. 2. It converges extremely slowly to 1 as $\epsilon \to 0$, and is continuous but "almost" discontinuous.

7. Critical behavior

Armed with the results of Section 5, let us now examine the phase transition for the counting machine near the critical point $\beta = \beta_c + \epsilon$ with $\beta_c = \ln 2$, $\epsilon \ll 1$. The free energy F is:

$$\hat{Z}(\beta) = e^{-\beta F} = \sum_p e^{-\beta l(p)} \quad \Rightarrow \quad F(\beta) = -\frac{1}{\beta} \ln \hat{Z}(\beta) \tag{16}$$

In our ensemble, the program length plays the role of the energy with expectation value

$$\langle l \rangle_\beta = -\partial_\beta \ln \hat{Z}(\beta) \tag{17}$$

The heat capacity is (using $T\partial_T = -\beta\partial_\beta$)

$$C(T) = -T\frac{\partial^2 F}{\partial T^2} \quad \sim \quad -\partial_\beta \ln\langle l \rangle \quad + \quad \text{higher orders in } \epsilon$$

According to the Ehrenfest classification, in a zeroth-order phase transition the free energy $F(T)$ is discontinuous at a critical point $T = T_c$. In a first-order transition, $F(T)$ is continuous but $\partial_T F(T)$ is discontinuous, the gap being the latent

heat. In a second-order transition, $\partial_T F(T)$ is also continuous, but some higher-order derivative of $F(T)$ is discontinuous [8]. In our case,

$$\hat{Z}(\epsilon) = 1 - \lambda \cdot 2^{-\text{slog}_2(1/\epsilon)} \;=\; 1 - \lambda \cdot 2^{-\,\text{lg}^-(1/\epsilon)} \cdot \{\text{lg} \dots \text{lg}\,(1/\epsilon)\}^{-1}$$

where $\lambda \sim 0.186$, lg^- is the integer part of the super-logarithm and we have $\text{lg}^-(1/\epsilon)$ iterations of the logarithm. Thus, in the limit $\epsilon \to 0$, we have, to leading order in ϵ:

$$\begin{aligned} F(\epsilon) &\propto -\lambda \cdot 2^{-\,\text{lg}^-(1/\epsilon)} \cdot \{\text{lg} \dots \text{lg}\,(1/\epsilon)\}^{-1} \\ \langle l \rangle(\epsilon) &\propto [\ \epsilon \cdot \text{lg}\frac{1}{\epsilon} \cdot \text{lg}\,\text{lg}\frac{1}{\epsilon} \cdot \dots \cdot (\text{lg} \dots \text{lg}\frac{1}{\epsilon})^2\ ]^{-1} \\ C(\epsilon) &\propto [\ \epsilon^2 \cdot \text{lg}\frac{1}{\epsilon} \cdot \text{lg}\,\text{lg}\frac{1}{\epsilon} \cdot \dots \cdot (\text{lg} \dots \text{lg}\frac{1}{\epsilon})^2\ ]^{-1} \end{aligned} \tag{18}$$

$F(\epsilon)$ is finite at the critical point. It is continuous, but almost discontinuous. Thus, the phase transition is first-order, but almost zeroth order. We also see that the latent heat is infinite, meaning that the average program size $\langle l \rangle$ diverges at the critical point. The average size N of the output strings also diverges, as $l \sim \text{lg}\,N + \text{lg}\,\text{lg}\,N + \cdots$

To put things into perspective, the age of the universe, as measured in Planck times, is $T_u \sim 2^{200}$. For $\epsilon < 1/T_u$, one needs to consider input bit strings of length $l > T_u$, and therefore computation times $> T_u$, to compute F and $\hat{Z}$ at $\beta = \beta_c + \epsilon$. The super-logarithm of T_u is about 4.6, so for ϵ of order 2^{-200}, $\hat{Z}$ is still about 0.76% away from 1. To get a super-logarithm of 5, we would need a universe of age $2^{65'536}$ Planck times. Even then, $\hat{Z}$ would still be 0.58% away from 1. In this sense, the super-logarithmic singularity (the dashed line in Fig. 2) is invisible at least for all bit string ensembles that can be computed in our universe. In other words, when carried out to any practical degree of accuracy, the calculation of the partition function never reaches the regime corresponding to the dashed line, because $\text{slog}_2(1/\epsilon)$ is effectively cut off at $4.5 - 5$.

8. Singularity for universal turing machines

In the previous section, we have discussed the singularity of the partition function (7) near $\epsilon = 0$ for the non-universal counting machine. How does it compare with the singularity for a universal Turing machine?

Since it is generally an undecidable question whether a given Turing machine halts for a given program, for a UTM the function h_l in (7) and Chaitin's Ω are not computable by any halting program. Neither is the singularity of $Z(\epsilon)$ at the critical point computable. In fact, $Z(\epsilon)$ converges towards Ω more slowly than any computable function.

To see this, let us slightly modify the last step of the counting machine of Section 4: if, in the ith step, $p_{m_{i-1}} = 0$, the modified T_0 switches into a new mode: instead of writing $1^{n_{i-1}}$ on the work tape, it reads the next $\Sigma(n_{i-1})$ digits of the program p from the program tape, where $\Sigma(n)$ is the busy-beaver function. The modified machine $\tilde{T}_0$ writes those digits on the work tape and then halts. Formula (11) thus gets replaced by

$$\tilde{Z}_k(\beta) = \sum_{n_2,\dots,n_k} \sum_{N=0}^{2^{\Sigma(n_k)}} e^{-\beta\cdot(6+n_2+\cdots+n_k+\Sigma(n_k))} \sim \sum_{n_2,\dots,n_k} 2^{-(6+n_2+\cdots+n_k)} \cdot \frac{1}{2}\, e^{-\epsilon\cdot\Sigma(n_k)}$$

$\Sigma(n)$ is known to diverge faster than any computable function as $n \to \infty$. This implies that $\tilde{Z}(\epsilon)$ converges more slowly than any computable function to its critical value 1 for the modified machine $\tilde{T}_0$. Now, any UTM U simulates the modified machine $\tilde{T}_0$, if it is fed with all possible input programs. This implies that, for any UTM, $Z_U(\epsilon)$ converges more slowly than any computable function to its critical value Ω.

The conclusions for UTM's are thus similar as for the counting machine: first of all, at the critical point, the phase transition is first-order but almost zeroth-order, with a divergent latent heat (i.e. average program size). The average size of the output bit strings also diverges, as U simulates T_0, among other machines. This divergent average program size is a pre-condition for a potential continuum limit of (2) at $\beta = \beta_c$, where very long bit strings might be described by continuous strings. This will be further discussed below.

Second, strictly speaking, $Z_U(\epsilon)$ for a UTM is continuous at $\epsilon = 0$, but in practice, its behavior is indistinguishable from a discontinuity. This seems to make it impossible to get a better understanding of the Chaitin point (1) by first studying the ensemble (2) at low temperature (high β), and then taking the limit $\beta \to \beta_c$.

However, as in the case of the counting machine, this discontinuity is invisible in finite computation time. Thus, we can at least interpolate between the low-temperature phase and a "real-world" version of the Chaitin point (1), in which the computation time is limited by the age T_u of the universe. Alternatively, we could limit the length L of the output bit strings B to $L \leq T_u$ (roughly the diameter of the observable universe), and count only the shortest program that produces each B. As B is produced by at least one program of size $L + O(1)$, namely "print B", this also limits the computation time. For most B, this amounts to replacing $K_\beta(B)$ by the Kolmogorov complexity $K(B)$ in (4), (5).

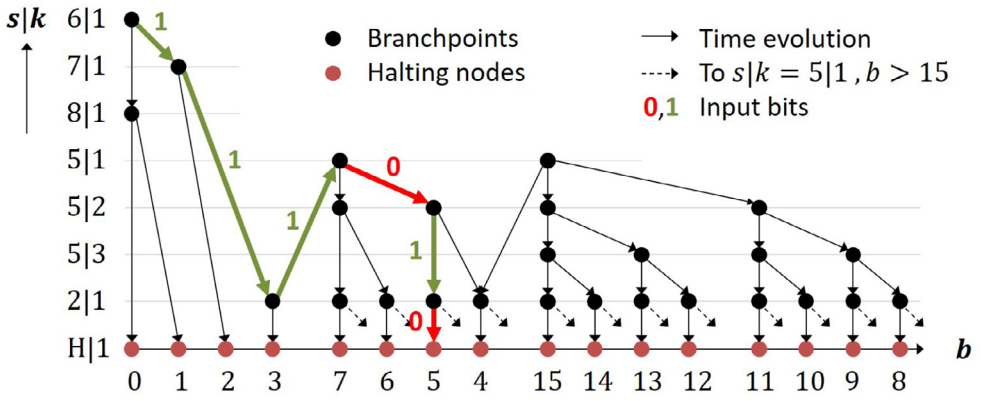

Fig. 3. Paths in configuration space for the counting machine.

9. Analogies with quantum mechanics

In this section, we discuss analogies and differences of the statistical–mechanical model (2) with quantum mechanics, both in the path integral- and the Hilbert space formulation.

So far, we have only considered the input programs p in (2), and not the bit strings b that they produce on the work tape during the computation. Let us now rewrite (2) as a sum over paths in the space of configurations of the Turing machine. At any point in time, a configuration is uniquely specified by the 3-tuple [23]

$$S = \{b, s, k\},$$

where b is the bit string on the work tape, s is the state of the Turing machine, and k is the position of the head. $k = 1$ means that the head sits on the first non-blank bit of the work tape. We define "program time" $\tau \in \{0, 1, 2, \ldots, l\}$ as the position of the head on the read-only program tape. Let us represent a computation by a path $S(\tau)$ over the configurations at the time the τ-th input bit is first read in, i.e, immediately before the path splits. The path ends at $\tau = l$, $b(l) = B$, $s(l) =$"Halt", where B is the output string.

Fig. 3 shows the computation paths and their first 15 output bit strings at the example of the counting machine. Fig. 3 can be thought of as an embedding of the tree graph of Fig. 1 in configuration space. The bit string $b(\tau)$ is plotted on the horizontal axis, written as a decimal number N, except for the halting state "H", in which case N denotes the length of the output bit string 1^N. The states $s(\tau)$ and the position $k(\tau)$ of the work head are denoted by $s|k(\tau)$ and are plotted on the vertical axis (only certain configurations appear). The arrows show the direction of the computation; dashed arrows lead to configurations with $N > 15$ and $s|k = 5|1$. Black dots represent configurations before the paths split ("branchpoints"). The green/red bits shown along the example of the computation path leading to $B = 11111_2$ ($N = 5$) make up the corresponding input program "111010". As an illustrative example, this computation is described in detail in Appendix A.5.

It is instructive to compare with a particle on a Euclidean N-dimensional lattice with coordinates $\vec{x}$. The amplitude for the particle to go from point $\vec{a}$ to $\vec{b}$ is (see, e.g., [24]):

$$G(\vec{a}, \vec{b}) = \sum_{l=0}^{\infty} N_l(\vec{a}, \vec{b}) \cdot \exp\{-\beta \cdot l\}, \tag{19}$$

where $N_l(\vec{a}, \vec{b})$ is the number of paths $\vec{x}(\tau)$ (or "world-lines") of length l on the lattice (measured in lattice spacings) that interpolate between $\vec{x}(0) = \vec{a}$ and $\vec{x}(l) = \vec{b}$. As $l \to \infty$, N_l diverges as $\exp\{\beta_c \cdot l\}$, where the precise value of β_c depends on the lattice. By setting

$$\beta = \beta_c + \epsilon m^2 \quad \text{as} \quad \epsilon \to 0,$$

a continuum limit can be reached, where the model describes a lattice-independent relativistic particle of mass m in N-dimensional Euclidean space with restored Poincaré symmetry. After a Wick rotation, this becomes a quantum mechanical particle in Minkowski space.

At least formally, the (high-dimensional) configuration space of the Turing machine is analogous to the lattice $\vec{x}$ of the particle, and the computation histories are analogous to the world lines $\vec{x}(\tau)$. The input bits on the program tape, which "live" on the paths in Fig. 3, correspond to the random variables $\vec{\eta}(\tau + 1) = \vec{x}(\tau + 1) - \vec{x}(\tau)$, which "live" on the particle's world line and describe its incremental movement on the lattice. Of course, a difference is that the computation of the Turing machine is irreversible, following the arrows in Fig. 3, while the motion of the particle is reversible and symmetric: $G(\vec{a}, \vec{b}) = G(\vec{b}, \vec{a})$.

Both for the particle and for the computation, the average length of the world lines diverges for $\epsilon \to 0$. It is a fascinating question what the computational analog of the continuum limit of the relativistic quantum particle might be. In trying to shed light on this, let us next discuss the corresponding Hilbert space in the case of the computation.

Using the classification of [23], let us begin by regarding the ensemble (2) as a probabilistic Turing machine. It acts on a Hilbert space of states, which are superpositions of the individual configurations S, each of them occurring with probability p_S:

$$|\Psi_\tau\rangle = \sum_S \psi_S(\tau)|S\rangle \quad \text{with} \quad \sum_S p_S = \sum_S |\psi_S|^2 \leq 1, \tag{20}$$

Given a pure state $|S\rangle$ at time τ, the next state at time $(\tau + 1)$ is a superposition of the two possible successor configurations, each of which occurs with probability $e^{-\beta_c} \sim 1/2$. E.g., for the counting machine, if we define l_N as in (8) and $\psi_B(\tau + 1) = \psi_B(\tau)$ for $\tau \geq l$ (i.e., after the computation halts), then, in the limit $\tau \to \infty$, the output state converges to

$$|\Psi_\infty\rangle = \sum_{N=0}^{\infty} \psi_N|N\rangle \quad \text{with} \quad |N\rangle \equiv |1^N, \text{Halt}, 1\rangle, \quad \psi_N = 2^{-l_N/2} \;\Rightarrow\; \sum_N |\psi_N|^2 = 1.$$

The evolution of $|\Psi_\tau\rangle$ from time τ to $\tau+1$ can be described by a transfer matrix T. This evolution is generally not reversible, and for a universal Turing machine not even probability is conserved due to non-halting programs (hence the $\leq$ sign in (20)). Thus, the transfer matrix T is not unitary and the computation generates heat. However, T can be extended such that the time evolution converges to a unitary one within the subspace $\hat{S}$ spanned by output states $|B\rangle$. E.g., an arbitrary phase δ_B can be added in the relation

$$\psi_B(\tau + 1) = e^{i\delta_B}\psi_B(\tau) \quad \text{for} \quad \tau \geq l.$$

An example with applications to error-correction codes is Manin's definition $\delta_B = K(B)$ [19]. It will be interesting to explore other applications for more general Transfer matrices $T = \exp\{i\tau H\}$, with a self-adjoint Hamiltonian H acting on the space $\hat{S}$ of halting states. This turns the probabilistic Turing machine into a quantum Turing machine acting on $\hat{S}$.

Our Hilbert space is spanned by states corresponding to discretized strings of dynamical length L, on which the bits – or, more generally, the letters of an alphabet – live. In an analogy with string theory, the non-blank portion of the work tape corresponds to an unembedded string, while the alphabet, whose letters are written on the tape, can be viewed as an embedding space of this string.

The next step, left for future work, will be to examine how the ground state of a UTM evolves from low temperature ($\beta \to \infty$) to the critical temperature ($\beta = \beta_c$). For $\beta \to \infty$, the system is in an ordered state. E.g., on a bit string of given length L, all letters tend to be identical to minimize the complexity. As $\beta \to \beta_c$, the average length $\langle L\rangle$ diverges, and the string fluctuates all over its embedding space. It is thus a natural conjecture that the computational analog of the continuum limit of the above relativistic quantum particle is a continuous quantum string.

10. Outlook

Let us first briefly summarize our results. We have considered the statistical–mechanical ensemble (2) of prefix-free input programs p of length $l(p)$ for a universal Turing Machine U. Let $p(B_L)$ denote programs that make U compute a given output bit string B_L of length L. Similarly as in (4), (5), we can decompose our Gibbs ensemble (2) as

$$Z(\beta) = \sum_L Z_L(\beta) \quad \text{with} \tag{21}$$

$$Z_L(\beta) = \sum_{B_L} e^{-\beta\cdot K_\beta(B_L)}, \quad \text{where} \quad e^{-\beta\cdot K_\beta(B_L)} = \sum_{p(B_L)} e^{-\beta\cdot l(p(B_L))} \tag{22}$$

is the algorithmic Solomonoff probability of B_L in the limit $\beta = \beta_c \equiv \ln 2$, which we call the Chaitin point, and $K_\beta(B_L)$ is the Kolmogorov complexity of B_L in the limit $\beta \to \infty$. $\beta \sim 1/T \to \infty$ corresponds to low temperature T, and $\beta = \beta_c$ to a critical temperature.

We have shown that ensemble (21) has a first-order phase transition at $\beta = \beta_c$. This phase transition is "almost" zeroth-order, in the sense that the free energy is continuous but almost discontinuous: it converges to its critical value, namely Chaitin's Ω, more slowly than any computable function. To illustrate this behavior, we have constructed a toy model, the "counting machine", whose free energy has a super-logarithmic singularity. We have also argued that, for finite computation time and therefore limited l, the effective discontinuity at the Chaitin point is invisible, and one can smoothly interpolate between the classical limit $\beta \to \infty$ and the critical point $\beta = \beta_c$.

At the Chaitin point, we have shown that the latent heat (the average program size) and the average size of the output bit strings both diverge. This begs the question whether there is a continuum limit, in which long bit strings are described by continuous strings, and the computation histories are represented by string world sheets. If so, it would establish a connection between information theory and string theory, through which these two theories could shed new light on each other.

As preparatory steps to address this question, in Section 9 we have discussed analogies and differences with the continuum limit of a quantum particle on a lattice. We have also interpreted (21) as a probabilistic Turing machine and defined the analog of a Hilbert space of states. The next step will be to study how the ground state of U evolves, as one interpolates from the ordered (i.e., low-complexity) phase at $\beta \to \infty$ to the critical point at $\beta = \beta_c$.

While the ensemble (21) has a first-order transition, when L is summed over, it is conceivable that a universal Turing machine has a second-order transition for $Z_L(\beta)$, when L is very large but fixed. Precedents of statistical mechanical models of dynamic size L that have (i) zeroth- or first-order phase transitions when one sums over L, and (ii) second-order transitions for fixed L, include the Ising model [25] or the sine-Gordon model on a random lattice [26,27] with L lattice sites, where L is a dynamical variable. These systems are indeed described by two-dimensional field theories known as "noncritical string theories".

Why should we care about the order of the phase transition of the ensemble (2)? At second-order phase transitions, statistical mechanical models that are very complex on a microscopic scale often have simple, universal macroscopic descriptions in terms of continuum field theories. E.g., water and steam at a temperature of 374°C and a pressure of 218 atm is described by a scalar ϕ^4 theory, which can be used to compute its critical exponents. At this critical point, almost all microscopic details become irrelevant. E.g., exactly the same critical exponents describe the second-order phase transition of CO_2, of simple lattice gas models, and of any other system in the same universality class.

Ensemble (21), (22) certainly looks intractable on a microscopic scale: the action is highly nonlocal, and not even computable by any halting program. Our hope of better understanding what happens at the Chaitin point therefore rests on the hypothesis that the ensemble of bit strings has a tractable continuum limit at this point, in the sense that it can be modeled by what could be called a "logical quantum field theory": a theory of continuous strings that is independent of the algorithmic details.

Why do we want to better understand the Chaitin point in the first place? It provides a fascinating link between information theory and statistical mechanics, for which many potential applications can be foreseen. Those include the analysis of heat generation in quantum Turing machines, and the generalization of the unitary time evolution of [19] to yield additional applications beyond error-correcting codes and Zipf's law. Moreover, if long bit strings can be described by continuum strings, this may help to simulate string theories and examine their vacua. It may even suggest a platonic answer to the question what strings are made of: they might be purely mathematical objects, made of bits.

CRediT authorship contribution statement

Christof Schmidhuber: Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

I would like to thank my brother Juergen Schmidhuber for arising my interest in information theory. The current work was inspired by his idea of the "Great Programmer" [28]. This research is supported in part by grant no. CRSK-2 190659 from the Swiss National Science Foundation.

Appendix. Review of turing machines

The appendix is organized as follows. Appendix A.1 presents a Turing machine that contains only a work tape and no program tape. An example is given in Appendix A.2. In Appendix A.3, Chaitin's definition of a Turing machine is recalled. Using the example of Appendix A.2 as a building block, we realize the counting machine of Section 4 in Appendix A.4. Appendix A.5 contains a supplementary argument to Section 5.

A.1. A simple turing machine

Our first example of a Turing machine contains a "work tape" that extends infinitely in both directions (see Fig. 4). It consists of cells that are blank, except for a finite, contingent bit string of 0's and 1's (the "input string"). A blank cannot be written between 0's or 1's, so it is not equivalent to a third letter in addition to 0 and 1. Rather, blank areas mark the beginning and end of the string on the work tape. On the first cell of the input string sits a head, which can read, write, and move in both directions. The head can be in one of several states, labeled by 1, 2, 3, …, H. At each step, the machine operates as follows:

1. it reads the bit on the work tape on which the head sits (0, 1 or a blank)

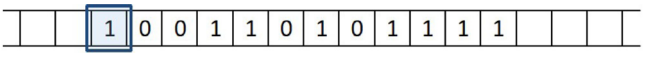

Fig. 4. Work tape of a Turing machine whose head (blue) sits on the first cell of an input string.

Table 1
Table of rules for the simple Turing machine of sub-section A1 (shaded in gray), and its extension to Chaitin's machine of sub-section A3 (white), which also includes a program tape.

Table 1		Current state				Current state & program bit									
Operation	Work Bit	1	2_0	3	4	2_1	5_0	5_1	6_0	6_1	7_0	7_1	8_0	8_1	8_B
Write	0	0	1	0	0	1	-	-	B	1	1	1	B	1	B
on work	1	1	0	1	1	0	0	1	B	1	1	1	B	1	B
tape	B	B	B	B	0	B	B	B	B	1	1	1	B	1	B
Set	0	1	2	3	4	2	-	-	8	7	H	1	H	H	6
the new	1	1	3	4	4	3	5	5	8	7	H	1	H	H	6
state	B	2	H	1	1	5	2	2	8	7	H	1	H	H	6
Move	0	1	-1	-1	-1	-1	-	-	1	1	-1	-1	-1	-1	1
on work	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1
tape	B	-1	1	1	1	1	-1	-1	1	1	-1	-1	-1	-1	1
Move on	0	-	-	-	-	-	-	-	1	1	0	1	0	0	1
program	1	0	0	0	0	0	1	1	1	1	0	1	0	0	1
tape	B	0	0	0	0	0	0	0	1	1	0	1	0	0	1

2. depending on that bit and on its internal state, it writes a 0, 1 or a blank in that cell on the work tape. It may only write a blank if the cell has a blank neighbor, to ensure that the binary string remains contingent
3. it moves the head either one cell to the left or one cell to the right
4. it may or may not change its internal state
5. If and when it reaches the state "H", it halts

A.2. An example

As an example, consider a Turing machine with 5 states 1, 2, 3, 4, H. The first six columns of Table 1 (in gray) define how this particular machine writes 0, 1 or B (B = blank), then moves left (-1) or right ($+1$), then switches to a new state, depending on the input bit it reads (left column) and the state it is in (top row; let us first ignore the subscript in 2_0).

First, let the input string be "01". Fig. 5 (left) shows a two-dimensional graph of the evolution of the bit string, with a new row appended for each time step. The machine

- starts in state 1 on the first bit of the work tape, which is 0.
- Therefore, the 3 yellow fields in Table 1 tell it what to do next: write a 0, remain in state 1, and move right to the next bit. It then sits on an input bit 1 in state 1.
- Therefore, the 3 orange fields tell it what to do next: write a 1, remain in state 1, and move right to the next bit. It then sits on a blank input bit in state 1.
- Therefore, the 3 red fields tell it what to do next: switch to state 2, and move back left to the previous bit. It then sits on an input bit 1 in state 2.
- Therefore, the 3 light blue fields tell it what to do next: overwrite the 1 with 0, switch to state 3, and move left, and so on.

At some point, the machine lands on a blank input bit in state 2, so the 3 medium-blue fields tell it what to do next: write a blank, move one bit to the right and halt. The output string are two 1's in a row. By modifying the input string, other output strings can be produced. The reader may verify that if the input string is the number b in binary code, then the output string of this particular Turing machine always consists of b 1's in a row, with the head halting on the first cell with value "1".

Fig. 5 (center) shows this computation in condensed form for $b = 1001_2 = 9_{10}$. By "condensed", we mean that each time step now corresponds to a new cell, rather than a new row, such that the computation time is the area of the graph. The head of the machine moves along the rows of the graph, and each time it changes direction, a new row is appended. For completeness, Fig. 5 (right) also shows the state of the machine at each point in the computation. The machine moves right along the light gray rows (state 1) and left along the other (blue) rows (states 2, 3, 4). We call these graphs "bit string world sheets".

As an example of a universal Turing Machine (UTM) that can simulate all other Turing machines, consider our brain: given the above table for any Turing machine, we can read it and use it to simulate the machine, as you have just done if you have followed the exercise. Essentially, the table becomes part of the input program, rather than being hard-coded into the Turing machine. For a more specific example of a universal Turing machine, see, e.g., [29]. A UTM is arbitrarily

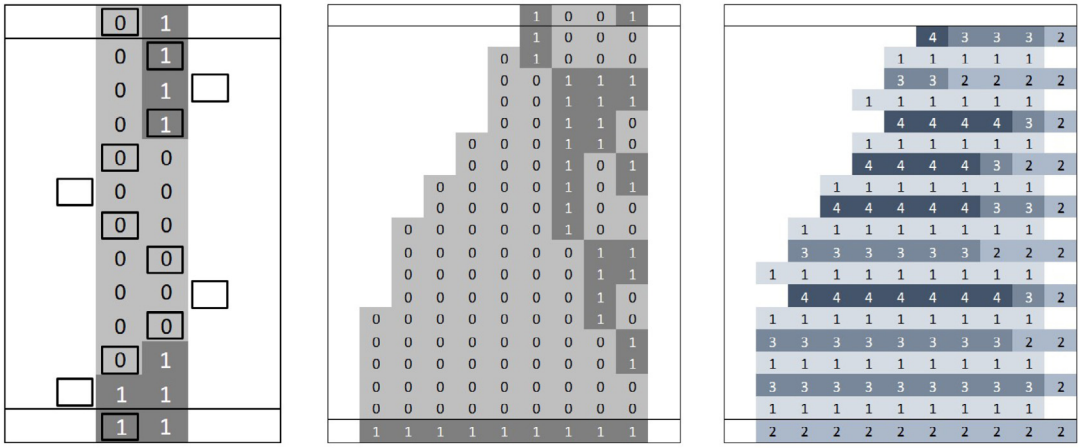

Fig. 5. Time evolution of bit strings on the work tape of the simple Turing machine of sub-sections A.1/A.2. The top line shows the input string, the bottom line shows the output string. Left: time evolution for the input string "01", as described in the text; each line corresponds to one computation step; the frame indicates the position of the head. Center: computation for input string "1001" in condensed form; now, each cell corresponds to one computation step. Right: the same computation, but instead of the bits on the work tape, the states of the machine are shown.

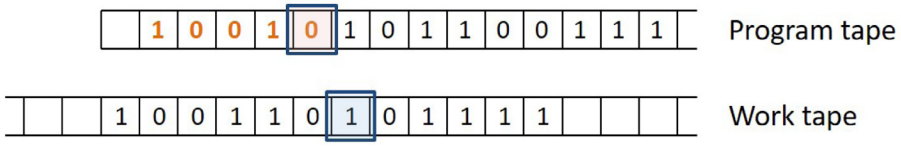

Fig. 6. Chaitin's version of a Turing machine. The program head (red) sits on the program tape and can only move to the right or stay where it is. The head on the work tape (blue) can move either left or right. If the machine halts in the configuration shown here, then the first 5 bits on the program tape (marked in orange) make up the "input program". The same is true if the machine runs forever without moving its program head further to the right.

flexible and can quickly compute strings with one Turing machine that take a long time or are impossible to compute with another machine.

There are many alternative, but equivalent definitions of Turing machines. E.g., one can introduce other symbols in addition to 0 and 1, or more states, or one can work with several parallel work tapes instead of just one.

A.3. Chaitin's machine

In Chaitin's definition, there is a semi-infinite read-only "program tape", in addition to the work tape (see Fig. 6). The program tape begins with a blank cell followed by a bit string of 0's and 1's. On the program tape sits another head, the "program head". Initially, it sits on the blank cell. At each step, the machine performs the following operations in addition to steps 1–5 of Appendix A.1:

- Step 0: it reads the bit on the program tape on which the head sits
- Step 6: it either moves the program head one cell to the right, or leaves it where it is

The machine either halts or runs forever, with or without reading any more program bits. As a result, the set of "input programs", i.e., the bit strings on the program tape from the first to the last bit that has been read by the machine, is prefix-free.

A.4. The counting machine

As an example within Chaitin's framework, we present an implementation of the counting machine of Section 4. We begin with the Turing machine of Appendix A.1, and add a finite read-only program tape, on which the programs of Section 5 are written. We start with a work tape that is initially blank.

We first add three additional states 6, 7, 8, whose role it is to read the first two bits on the program tape and get the machine started (steps 1 and 2 of Section 5). The operations in states 6, 7, 8 depend only on the program bit on which the program head sits, and not on the work bit on which the work head sits. They are defined in the last 7 columns of Table 1,

Input program 111010			Work tape							Program tape						
Step	State	Program bit	-4	-3	-2	-1	0	1	2	0	1	2	3	4	5	6
1	8	B								B	1	1	1	0	1	0
2	6	1									1	1	1	0	1	0
3	7	1					1				1	1	1	0	1	0
4	1	1					1	1			1	1	1	0	1	0
5	2	1				1	1	1			1	1	1	0	1	0
6	5	1				1	1	1			1	1	1	0	1	0
7	2	0				1	0	1			1	1	1	0	1	0
8	2	0		1	1	1	1	1			1	1	1	0	1	0
9	H	0		1	1	1	1	1			1	1	1	0	1	0

Fig. 7. Computation of the output bit string "11111" from the input program "111010". Snapshots of the state of the machine, the work tape, and the program tape are shown, corresponding to steps 1–9 as described in the text. Blue and red frames indicate the positions of the head on the work tape and the program head on the program tape.

where s_ν denotes the machine in state s with its program head sitting on a bit $\nu \in \{0, 1, B\}$. The machine is initially in state 8. It then turns out that in states 6 and 7, the program bit is never blank.

Next, we slightly modify state 2 in Table 1 as follows: if the machine is in state 2, and the head on the work tape sits on a blank, then it switches to state H *only* if the head of the program tape sits on a 0. Otherwise, it switches to a new state 5. In other words, in state 2_0, the machine halts, but in state 2_1, it switches to state 5, as shown in column 7 of Table 1. The operations of the new state 5 are also defined in Table 1. The role of state 5 is to write a new portion from the program tape onto the work tape, thereby over-writing the contingent sequence of 1's. Its operations depend both on the current work bit and on the current program bit (it turns out that the machine is never in state 5 when the program head is on a blank or when the work head is on a 0).

It is straightforward to verify that this machine indeed represents the counting machine of Section 4. Fig. 7 demonstrates this at the example of the input bit string "111010", whose computation path is also shown in Fig. 3. The machine operates as follows:

1. It starts in state 8_B. The program head is on the first (blank) cell of the program tape. The work head is on some cell (we call it the -1th cell) of the blank work tape.
2. The 4 light green fields in Table 1 tell the machine what to do next: leave the work tape blank, switch to state 6, and move 1 cell to the right both on the work tape (landing on a blank) and the program tape (landing on a 1). It is then in state 6_1.
3. The 4 dark green fields in Table 1 tell it what to do next: write a 1 in cell 0 of the work tape, switch to state 7, and move 1 cell to the right on the work tape (landing on a blank) and the program tape (landing on a 1). It is then in state 7_1.
4. The 4 medium-green fields in Table 1 tell it what to do next: write a 1 in cell 1 of the work tape, switch to state 1, and move left on the work tape (landing on a 1) and right on the program tape (landing on a 1).
5. The machine is now in state 1, in which the program bit is irrelevant. The work head sits on the first cell of the string "11", and in the next few steps the machine simulates the simple Turing machine discussed in Appendix A.1/A.2: It writes 3 bits in a row on the work tape (as $11_2 = 3$), then places its work head on the blank cell to the left of this string "111", and switches to state 2_1 (the program bit is still 1).
6. Following the dark blue fields in Table 1, it leaves the work bit blank and moves the work head to the right, onto the first bit of the string "111". If the machine was in state 2_0, it would halt now. However, since it is in state 2_1, it switches to state 5_1.
7. Therefore, the dark blue fields in Table 1 tell the machine what to do next. In the next 3 steps (in states $5_1, 5_2$), it copies the string "101" from the program tape (cells no 3–5) onto the work tape, then switches to state 2_0, with the work head on a 1.
8. In the next few steps the machine again simulates the simple Turing machine of Appendix A.1/A.2 (shaded in gray in the table): It writes 5 bits in a row on the work tape (as $101_2 = 5$), then places its work head on the blank cell to the left of this string "11111", and switches to state 2_0 (the program bit is still 0).
9. Following the medium-blue fields in Table 1, it leaves the work bit blank, moves the work head to the right, onto the first bit of the string "11111", and halts.

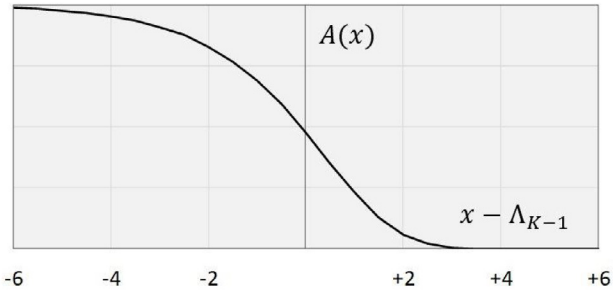

Fig. 8. The function $A(x)$, as described in the text.

A.5. A supplementary argument

In Section 5, we want to evaluate the Kth part of the partition function

$$\hat{Z}_K(\epsilon) = \sum_{n_2,\ldots,n_K} 2^{-(6+n_2+\cdots+n_{K-1})} \cdot \frac{1}{2}\, e^{-\epsilon n_K} \quad \text{in the case} \quad \frac{1}{\epsilon} = \Lambda_K \tag{23}$$

where $K \geq 4$, n_2 runs from 4 to 7, n_{i+1} runs from 2^{n_i-1} to $2^{n_i} - 1$, and Λ_K is the largest possible value of n_K. Specifically, $\Lambda_3 = 127$, $\Lambda_4 = 2^{127} - 1$, and therefore $\Lambda_{K-1} \sim \lg(\Lambda_K) = \lg(1/\epsilon)$ to high accuracy for $K \geq 4$.

Defining $M = 2^{n_{K-1}}$, the sum over n_K yields

$$\hat{Z}_K(\epsilon) = \sum_{n_1,n_2,\ldots,n_{K-1}} 2^{-(6+n_2+\cdots+n_{K-2})} \cdot A(n_{K-1}, \epsilon) \tag{24}$$

$$A(n_{K-1}, \epsilon) = \frac{1}{2M} \sum_{n_K=M/2}^{M-1} e^{-\epsilon \cdot n_K} = \frac{1}{2M\epsilon}(e^{-M\epsilon/2} - e^{-M\epsilon}) \tag{25}$$

$A(x, \epsilon)$ is plotted in Fig. 8. It is a monotonously decaying function with

$$A(x, \epsilon) \to \begin{cases} \frac{1}{4} & \text{for} \quad x \ll \lg \frac{1}{\epsilon} \sim \Lambda_{K-1} \\ 0 & \text{for} \quad x \gg \lg \frac{1}{\epsilon} \sim \Lambda_{K-1} \end{cases} \tag{26}$$

n_{K-1} runs from $2^{n_{K-2}-1}$ to $2^{n_{K-2}} - 1$. Only for the maximal value of n_{K-2} are there a few values of n_{K-1} near Λ_{K-1}, for which A differs significantly from 1/4. Even in this case, the contribution of these differences is

- small (of order 1%) for $K = 4$: for the highest value $n_2 = 7$, n_3 runs from 64 to 127. Only the last few of these n_3 contribute significantly to the difference
- practically zero for $K \geq 5$: e.g., for $K = 5$ and the highest value $n_3 = 127$, n_4 runs from 2^{126} to $2^{127} - 1$. Only a tiny portion of these n_4 contribute to the difference

As long as $K \geq 4$, we can thus approximate A by 1/4 for $x \leq \Lambda_{K-1}$ to obtain $\hat{Z}_K = 2^{-K-3}$ as claimed in Section 5. An analogous argument, not repeated here, shows that we can approximate A by 0 for $x > \Lambda_{K-1}$ to obtain $\hat{Z}_{K+1} = 0$, as long as $K \geq 4$.

References

[1] Chaitin G.J., A theory of program size formally identical to information theory, J. Assoc. Comput. Mach. 22 (1975) 329–340.
[2] A.M. Turing, On computable numbers with an application to the entscheidungsproblem, Proc. Lond. Math. Soc. (2) 42 (1936) 230–265; Correction, on computable numbers with an application to the entscheidungsproblem, Proc. Lond. Math. Soc. (2) 43 (1937) 544–546.
[3] M. Li, P. Vitanyi, An Introduction To Kolmogorov Complexity and Its Applications, third ed., Springer Science+Business Media, 1993.
[4] K. Goedel, Ueber formal unentscheidbare Saetze der Principia Mathematica und verwandter Systeme I, Monatshefte Fuer Math. Phys. 38 (1931) 173–198.
[5] K. Tadaki, A generalization of Chaitin's Halting probability Ω and Halting self-similar sets, 2002, arXiv preprint nlin/0212001.
[6] C.S. Calude, M.A. Stay, Natural halting probabilities, partial randomness, and zeta functions, Inform. and Comput. 204 (11) (2006) 1718–1739.
[7] J.C. Baez, M. Stay, Algorithmic thermodynamics, 2010, arXiv preprint arXiv:1010.2067.
[8] J. Zinn-Justin, Quantum Field Theory and Critical Phenomena, Oxford University Press, 1989.
[9] M.M. Green, J.H. Schwarz, E. Witten, Superstring Theory Vol. 1, 2, Cambridge University Press, 1988.
[10] K. Tadaki, A statistical mechanical interpretation of algorithmic information theory, in: Local Proc. of Computability in Europe 2008 (CiE 2008) (University of Athens, Greece), June 15–20, 2008, pp. 425-434. Electronic Version: http://www.cs.swan.ac.uk/cie08/cie2008-local.pdf.
[11] K. Tadaki, Phase transition between unidirectionality and bidirectionality, in: International Conference on Teaching and Computational Science (203–223), Springer, Berlin, Heidelberg, 2012.

[12] K. Tadaki, Phase transition and strong predictability, in: International Conference on Unconventional Computation and Natural Computation, Springer, Cham, 2014, pp. 340–352.
[13] A.N. Kolmogorov, On tables of random numbers, Sankhyā (1963) 369–376.
[14] Ray J. Solomonoff, A formal theory of inductive inference. Part I, Inf. Control 7 (1) (1964) 1–22; Part II, Inf. Control 7 (2) (1964) 224–254.
[15] Leonid Anatolevich Levin, Laws of information conservation (nongrowth) and aspects of the foundation of probability theory, Probl. Pereda. Inf. 10 (3) (1974) 30–35.
[16] Artemy Kolchinsky, David H. Wolpert, Thermodynamic costs of turing machines, Phys. Rev. Res. 2 (3) (2020) 033312.
[17] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (3) (1961) 183–191.
[18] Yuri I. Manin, Zipf's law and avoidance of excessive synonymy, Cogn. Sci. 32 (7) (2008) 1075–1098.
[19] Yuri I. Manin, Complexity vs energy: theory of computation and theoretical physics, J. Phys. Conf. Ser. 532 (1) (2014).
[20] Yuri I. Manin, Physics in the World of Ideas: Complexity As Energy, ISCS 2014: Interdisciplinary Symposium on Complex Systems, Springer, Cham, 2015.
[21] T.S. Cubitt, D. Perez-Garcia, M.M. Wolf, Undecidability of the spectral gap, Nature 528 (7581) (2015) 207–211.
[22] J. Bausch, T. Cubitt, A. Lucia, D. Perez-Garcia, Undecidability of the spectral gap in one dimension, 2018, arXiv preprint arXiv:1810.01858.
[23] L. Fortnow, One complexity theorist's view of quantum computing, Theoret. Comput. Sci. 292 (3) (2003) 597–610.
[24] A.M. Polyakov, Gauge fields and strings, in: Contemporary Concepts in Physics, vol. 3, harwood academic publishers, 1987, (chapter 9).
[25] D. Gross, A. Migdal, Nonperturbative solution of the ising model on a random surface, Phys. Rev. Lett. 64 (1990).
[26] G. Moore, Gravitational phase transitions and the sine-Gordon model, 1992, arXiv preprint hep-th/9203061.
[27] Christof. Schmidhuber, Exactly marginal operators and running coupling constants in 2-D gravity, Nuclear Phys. B 404 (1993).
[28] Juergen Schmidhuber, A computer scientist's view of life, the universe, and everything, in: C. Freksa, et al. (Eds.), Foundations of Computer Science: Potential - Theory - Cognition, in: Lecture Notes in Computer Science, Springer, 1997.
[29] M.L. Minsky, Size and structure of universal turing machines using tag systems, 1966.

Christof Schmidhuber is a physicist, who received his Ph.D. from CalTech in 1993 with a thesis on superstring theory. After that, he worked at Princeton University, Bern University, and CERN. He then spent 17 years in the alternative investment industry, but recently returned to academia at Zurich University of Applied Sciences. In his research, he is now exploring new and original applications of Statistical Mechanics.