Logical Depth and Physical Complexity

Charles H. Bennett
IBM Research, Yorktown Heights NY 10598, USA

pp. 227-257 in *The Universal Turing Machine– a Half-Century Survey*, edited by Rolf Herken, Oxford University Press (1988)

Abstract

Some mathematical and natural objects (a random sequence, a sequence of zeros, a perfect crystal, a gas) are intuitively trivial, while others (e.g. the human body, the digits of π) contain internal evidence of a nontrivial causal history.

We formalize this distinction by defining an object's "logical depth" as the time required by a standard universal Turing machine to generate it from an input that is algorithmically random (i.e. Martin-Löf random). This definition of depth is shown to be reasonably machine-independent, as well as obeying a slow-growth law: deep objects cannot be quickly produced from shallow ones by any deterministic process, nor with much probability by a probabilistic process, but can be produced slowly.

Next we apply depth to the physical problem of "self-organization," inquiring in particular under what conditions (e.g. noise, irreversibility, spatial and other symmetries of the initial conditions and equations of motion) statistical-mechanical model systems can imitate computers well enough to undergo unbounded increase of depth in the limit of infinite space and time.

1 Introduction

Persons of Turing's genius do not shy away from asking big questions, and hoping to see, in advance of the eventual slow progress of science, the essential outlines of the answers. "What is intelligence?" is one such question that clearly fascinated Turing. "How do complicated structures arise in nature?" is another. On this latter question, seeds planted by Turing have begun to mature to an extent that I think would please him.

Two outgrowths of computability theory, viz. algorithmic information theory and computational complexity theory, have made it possible to formalize satisfactorily the seemingly vague intuitive notion of complexity. Meanwhile, advances in equilibrium and nonequilibrium statistical mechanics have shed considerable light on the conditions, hinted at in Turing's work on morphogenesis [42], required for a simple, initially homogeneous medium to organize itself into structures capable of holding and processing information in the manner of a computer. We begin by showing how universal Turing machines can be used to formalize intuitive notions of complexity, returning in section 5 to the question of self-organization in physics.

The ability of the universal Turing machine to simulate any known algorithmic process, and the belief (commonly called Church's thesis but independently proposed by Turing) that all algorithmic processes can be so simulated, has prompted and justified the use of universal Turing machines to define absolute properties of mathematical objects, beginning with the distinction drawn by Turing [41] between between computable and uncomputable real numbers.

Church's thesis has been found to hold to a considerable extent for the efficiency, as well as the possibility, of computation. A large variety of computational models (roughly, those with at most polynomially growing parallelism) can be simulated by universal Turing machine in polynomial time, linear space, and with an additive constant increase in program size. This stronger Church's thesis has prompted the definition, on the one hand, of robust dynamic complexity classes such as P and $PSPACE$, and on the other hand of a nearly machine-independent algorithmic theory of information and randomness.

The dynamic complexity class P, for example, consists of those 0/1-valued functions of a binary string argument computable in time (i.e. number of machine cycles) bounded by a polynomial in the length of the argument, and includes the same functions regardless of whether the computations are performed by single-tape Turing machines, multi-tape Turing machines, cellular automata, or a wide variety of other models. The class $PSPACE$ is defined analogously, but with the bound on space (i.e. number of squares of tape) instead of time. Diagonal constructions similar to that used to prove the existence of uncomputable functions can be used to define provably hard-to-compute functions, requiring for example exponential time and space to compute. A major open question of dynamic complexity theory is the $P = PSPACE$ question: it is widely conjectured, but not known, that P is a proper subset of $PSPACE$, i.e. that there exist functions computable in

polynomial space but requiring more than polynomial time.

Algorithmic information theory [24], [40], [48] uses a standard universal Turing machine to define the *Kolmogorov complexity* or *information content* of a string x as the length of its *minimal program* x^*, the shortest binary input which causes the standard machine to produce exactly x as output. A string is said to be *compressible* by k bits if its minimal program is $\geq k$ bits shorter than the string itself. A simple counting argument shows that at most a fraction 2^{-k} of strings of length $\leq n$ bits can have this property. This fact justifies calling strings that are incompressible, or nearly so, *algorithmically random*. Like the majority of strings one might generate by coin tossing, such strings lack internal redundancy that could be exploited to encode them concisely, using the given universal Turing machine as decoding apparatus. Because of the ability of universal Turing machines to simulate one another, the property of algorithmic randomness is approximately machine-independent: a string that is incompressible on one machine cannot be compressed, on another machine, by more than the fixed number of bits required to program the first machine to simulate the second.

The relation between universal computer programs and their outputs has long been regarded [40] as a formal analog of the relation between theory and observation in science, with the minimal-sized program representing the most economical, and therefore a priori most plausible, explanation of its output. This analogy draws its authority from the ability of universal computers to execute all formal deductive processes and their presumed ability to simulate all processes of physical causation. Accepting this analogy, one is then led to accept the execution of the minimal program as representing its output's most plausible causal history, and a logically "deep" or complex object would then be one whose most plausible origin, via an effective process, entails a lengthy computation. Just as the plausibility a scientific theory depends on the economy of its assumptions, not on the length of the deductive path connecting them with observed phenomena, so a slow execution time is not evidence against the plausibility of a program; rather, if there are no comparably concise programs to compute the same output quickly, it is evidence of the nontriviality of that output.

A more careful definition of depth should not depend only on the minimal program, but should take fair account of all programs that compute the given output, for example giving two $k+1$ bit programs the same weight as one k-bit program. This is analogous in science to the explanation of a phenomenon by appeal to an ensemble of possible causes, individually unlikely but collectively plausible, as in the kinetic theory of gases. Several

nearly-equivalent definitions of depth are considered in section 3; the one finally advocated defines an object's "s-significant depth" as the least time required to compute it by a program that is itself compressible by no more than s bits. This formalizes the notion that any hypothesis of the object's more rapid origin suffers from s bits of redundancy. Such redundancy fairly measures the "ad-hocness" of a hypothesis, the extent to which it contains unexplained, a-priori-unlikely internal correlations that could be explained by deriving the original hypothesis from a more concise, non-redundant hypothesis.

The notion of logical depth developed in the present paper was first described in [12], and at greater length in [5] and [6]; similar notions have been independently introduced by Adleman [1] ("potential"), Levin and V'jugin [28] ("incomplete sequence"), Levin [29] ("hitting time"), and Koppel [25] ("sophistication"). See also Wolfram's work on "computational irreducibility" [45] and Hartmanis' work on time- and space-bounded algorithmic information [22].

We propose depth as a formal measure of value. From the earliest days of information theory it has been appreciated that information per se is not a good measure of message value. For example, a typical sequence of coin tosses has high information content but little value; an ephemeris, giving the positions of the moon and planets every day for a hundred years, has no more information than the equations of motion and initial conditions from which it was calculated, but saves its owner the effort of recalculating these positions. The value of a message thus appears to reside not in its information (its absolutely unpredictable parts), nor in its obvious redundancy (verbatim repetitions, unequal digit frequencies), but rather in what might be called its buried redundancy—parts predictable only with difficulty, things the receiver could in principle have figured out without being told, but only at considerable cost in money, time, or computation. In other words, the value of a message is the amount of mathematical or other work plausibly done by its originator, which its receiver is saved from having to repeat.

Of course, the receiver of a message does not know exactly how it originated; it might even have been produced by coin tossing. However, the receiver of an obviously non-random message, such as the first million bits of π, would reject this "null" hypothesis, on the grounds that it entails nearly a million bits worth of ad-hoc assumptions, and would favor an alternative hypothesis that the message originated from some mechanism for computing pi. The plausible work involved in creating a message, then, is the amount of work required to derive it from a hypothetical cause involving

no unnecessary, ad-hoc assumptions. It is this notion of the message value that depth attempts to formalize.

Depth may be contrasted with other ways of attributing dynamic complexity to finite objects, for example by regarding them as inputs of a universal computer rather than as outputs. An object would be complex in this sense if it caused the computer to embark on a long (but terminating) computation. Such objects might be called "ambitious" because they describe a lengthy computation that may never have been done. A deep object, on the other hand, contains internal evidence that a lengthy computation has already been done. When considering moderate levels of ambition, e.g. exponential rather than busy-beaver in the size of the objects being considered, it is best to define ambition strictly, as the actual run time of a string when executed as a program. In this case, of course, ambition is not very robust, since a slight change can convert a slow-running program into a fast-running one, or vice versa. In particular, a deep object need not be ambitious. At high levels of ambition, the notion can be given more robustness and interest (cf. [18]), by defining an ambitious string more broadly as one which is very slow-running itself or from which a very slow-running program can easily be computed. In this case, as string's ambition measures the amount of information it contains about the halting problem, i.e. about how to compute very large numbers; and, as will be seen later, deep objects are necessarily ambitious but not conversely.

Another kind of complexity associated with an object would be the difficulty, given the object, of finding a plausible hypothesis to explain it. Objects having this kind of complexity might be called "cryptic": to find a plausible origin for the object is like solving a cryptogram. A desirable (but mathematically unproven) property for small-key cryptosystems is that encryption should be easy, but breaking the system (e.g. inferring the key from a sufficient quantity of intercepted ciphertext) should be hard. If satisfactory small-key cryptosystems indeed exist, then typical cryptograms generated from shallow plaintexts are shallow (because they can be generated quickly from shallow input information) but cryptic (because even when the cryptogram contains sufficient information to uniquely determine the plaintext and key, the job of doing so is computationally infeasible).

One might argue that a message's usefulness is a better measure of value than its mathematical replacement cost, but usefulness is probably too anthropocentric a concept to formalize mathematically.

Related ideas appear in the fiction of Borges [10], e.g. in the story "Pierre Menard, Author of the *Quixote*" about a man who with great effort recon-

structs several chapters of *Don Quixote* without errors and without consulting the original, from an intimate knowledge of the book's historical and literary context. Another story, "The Library of Babel" describes a library whose shelves contain, in seemingly random order, all possible 410-page books on a 25-letter alphabet, and the librarians' attempts to discern which of the books were meaningful.

The tradeoff between conciseness of representation and ease of decoding is illustrated in an extreme form by the information required to solve the halting problem. One standard representation of this information is as an infinite binary sequence K_0 (the characteristic sequence of the halting set) whose i'th bit is 0 or 1 according to whether the i'th program halts. This sequence is clearly redundant, because many instances of the halting problem are easily solvable or reducible to other instances. Indeed, K_0 is far more redundant than this superficial evidence might indicate. Barzdin [2] showed that this information can be compressed to the logarithm of its original bulk, but no concisely encoded representation of it can be decoded in recursivley bounded time.

The most elegantly concise representation of the halting problem is Chaitin's irrational number Ω [11], defined as the halting probability of a universal computer programmed by coin tossing (the computer starts in a standard state, and whenever it tries to read a bit of its program, a coin is tossed to supply that bit). Such a randomly programmed computation is like the old notion of a monkey accidentally typing the works of Shakespeare, but now the monkey sits at a computer keyboard instead of a typewriter. The result is still trivial with high probability, but any nontrivial computation also has a finite probability of being performed, inversely proportional to the exponential of the length of its program. The essential features of Ω [11][3] are

i) The first n bits of Ω suffice to decide approximately the first 2^n cases of the halting problem (more precisely, to decide the fate of all computations using $\leq n$ coin tosses), but there is no faster way of extracting this information than to recursively enumerate halting programs until enough have been found to account for all but 2^{-n} of the total halting probability Ω, a job which requires at least as much time as running the slowest n bit program.

ii) Ω is algorithmically random: like a typical coin toss sequence, its first n bits cannot be encoded in less than $n - O(1)$ bits. [This algorithmic randomness may seem contradictory to, but in fact is a consequence of, Ω's compact encoding of the halting problem. Knowledge of Ω_n (i.e. the

first n bits of Ω) is sufficient to decide the halting of all programs shorter than n bits, and therefore to determine which n-bit strings are compresible and which are not, and therefore to find and print out the lexicographically first incompressible n-bit string x. If Ω_n itself were significantly compressible, then the algorithm just described could be combined with the compressed representation of Ω_n to yield a $< n$ bit program to produce x, a supposedly incompressible n-bit string. This contradiction proves the $O(1)$-incompressibility of Ω.]

iii) although it solves unsolvable problems, Ω does not speed up the solution of solvable problems any more than a random coin-toss sequence would. [This uselessness follows from its algorithmic randomness. Any special ability of Ω to speed up the computation of a computable function would set it apart from random sequences in general, and this atypicality could be exploited to encode Ω more concisely, contradicting its randomness].

In summary, we may say that the Ω contains the same information as K_0, but in such a compressed form that as to be random and useless. Ω is a shallow representation of the deep object K_0.

Descending from the realm of the halting problem to more mundane levels of complexity (e.g. polynomial versus exponential), one is tempted to invoke depth as a measure of complexity for physical objects, but one must proceed cautiously. The applicability of any idea from computational complexity to physics depends on what has been called a "physical Church's thesis," the belief that physical processess can be simulated with acceptable accuracy and efficiency by digital computations. Turing was quite aware of this question, and based his informal justification of his model of computation largely on physical considerations. Granting some form of physical Church's thesis, a nontrivial application of depth to physical systems appears to depend on unproven conjectures at the low end of complexity theory, such as $P \neq PSPACE$. These questions are considered in the last section of the paper.

2 Preliminaries, Algorithmic Information

As usual, a natural number x will be identified with the x'th binary string in lexicographic order (Λ,0,1,00,01,10,11,000...), and a set X of natural numbers will be identified with its characteristic sequence, and with the real number between 0 and 1 having that sequence as its dyadic expansion. The length of a string x will be denoted $|x|$, the n'th bit of an infinite sequence

X will be noted $X(n)$, and the initial n bits of X will be denoted X_n. Concatenation of strings p and q will be denoted pq.

We define the information content (and later the depth) of finite strings using a universal Turing machine U similar to that described by Chaitin [11]. A universal Turing machine may be viewed as a partial recursive function of two arguments. It is universal in the sense that by varying one argument ("program") any partial recursive function of the other argument ("data") can be obtained. In the usual machine formats, program, data and output are all finite strings, or, equivalently, natural numbers. However, it is not possible to take a uniformly weighted average over a countably infinite set. numbers. Since we wish to average uniformly over programs, we adopt a format [16] [27] [11] in which the program is in effect an *infinite* binary sequence, but data and output are finite strings. Chaitin's universal machine has two tapes: a read-only one-way tape containing the infinite program; and an ordinary two-way read/write tape, which is used for data input, intermediate work, and output, all of which are finite strings. Our machine differs from Chaitin's in having some additional auxiliary storage (e.g. another read/write tape) which is needed only to improve the time efficiency of simulations.

We consider only terminating computations, during which, of course, only a finite portion of the program tape can be read. Therefore, the machine's behavior can still be described by a partial recursive function of two string arguments $U(p, w)$, if we use the first argument to represent that portion of the program that is actually read in the course of a particular computation. The expression $U(p, w) = x$ will be used to indicate that the U machine, started with any infinite sequence beginning with p on its program tape and the finite string w on its data tape, performs a halting computation which reads exactly the initial portion p of the program, and leaves output data x on the data tape at the end of the computation. In all other cases (reading less than p, more than p, or failing to halt), the function $U(p, w)$ is undefined. Wherever $U(p, w)$ is defined we say that p is a *self-delimiting program* to compute x from w, and we use $T(p, w)$ to represent the time (machine cycles) of the computation. Often we will consider computations without input data; in that case we abbreviate $U(p, \Lambda)$ and $T(p, \Lambda)$ as $U(p)$ and $T(p)$ respectively.

The self-delimiting convention for the program tape forces the domain of U and T, for each data input w, to be a *prefix set*, that is, a set of strings no member of which is the extension of any other member. Any prefix set S obeys the Kraft inequality

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

Besides being self-delimiting with regard to its program tape, the U machine must be *efficiently universal* in the sense of being able to simulate any other machine of its kind (Turing machines with self-delimiting program tape) with at most an additive constant constant increase in program size and a linear increase in execution time.

Without loss of generality we assume that there exists for the U machine a constant prefix r which has the effect of stacking an instruction to restart the computation when it would otherwise end. This gives the machine the ability to concatenate programs to run consecutively: if $U(p, w) = x$ and $U(q, x) = y$, then $U(rpq, w) = y$. Moreover, this concatenation should be efficient in the sense that $T(rpq, w)$ should exceed $T(p, w) + T(q, x)$ by at most $O(1)$. This efficiency of running concatenated programs can be realized with the help of the auxiliary storage to stack the restart instructions.

Sometimes we will generalize U to have access to an "oracle" A, i.e. an infinite look-up table which the machine can consult in the course of its computation. The oracle may be thought of as an arbitrary 0/1-valued function $A(x)$ which the machine can cause to be evaluated by writing the argument x on a special tape and entering a special state of the finite control unit. In the next machine cycle the oracle responds by sending back the value $A(x)$. The time required to evaluate the function is thus linear in the length of its argument. In particular we consider the case in which the information in the oracle is random, each location of the look-up table having been filled by an independent coin toss. Such a *random oracle* is a function whose values are reproducible, but otherwise unpredictable and uncorrelated.

The following paragraph gives a more formal definition of the functions U and T, which may be skipped by the casual reader.

Let $\{\varphi_i^A(p, w) : i = 0, 1, 2...\}$ be an acceptable Gödel numbering of A-partial recursive functions of two arguments and $\{\Phi_i^A(p, w)\}$ an associated Blum complexity measure, henceforth referred to as time. An index j is called self-delimiting iff, for all oracles A and all values w of the second argument, the set $\{ x : \varphi_j^A(x, w) \text{ is defined } \}$ is a prefix set. A self-delimiting index has efficient concatenation if there exists a string r such that for all oracles A and all strings w, x, y, p, and q, if $\varphi_j^A(p, w) = x$ and $\varphi_j^A(q, x) = y$, then $\varphi_j^A(rpq, w) = y$ and $\Phi_j^A(rpq, w) = \Phi_j^A(p, w) + \Phi_j^A(q, x) + O(1)$. A self-delimiting index u with efficient concatenation is called efficiently universal

iff, for every self-delimiting index j with efficient concantenation, there exists a simulation program s and a linear polynomial L such that for all oracles A and all strings p and w,

$$\varphi_u^A(sp, w) = \varphi_j^A(p, w)$$

and

$$\Phi_u^A(sp, w) \leq L(\Phi_j^A(p, w))$$

. The functions $U^A(p, w)$ and $T^A(p, w)$ are defined respectively as $\varphi_u^A(p, w)$ and $\Phi_u^A(p, w)$, where u is an efficiently universal index.

We now present some definitions and review some elementary facts about algorithmic information.

For any string x, the *minimal program*, denoted x^*, is $\min\{p : U(p) = x\}$, the least self-delimiting program to compute x. For any two strings x and w, the minimal program of x relative to w, denoted $(x/w)^*$, is defined similarly as $\min\{p : U(p, w) = x\}$.

By contrast to its minimal program, any string x also has a *print program*, of length $|x| + O(\log |x|)$, which simply transcribes the string x from a verbatim description of x contained within the program. The print program is logarithmically longer than x because, being self-delimiting, it must indicate the length as well as the contents of x. Because it makes no effort to exploit redundancies to achieve efficient coding, the print program can be made to run quickly (e.g. linear time in $|x|$, in the present formalism).

Extra information w may help, but cannot significantly hinder, the computation of x, since a finite subprogram would suffice to tell U to simply erase w before proceeding. Therefore, a relative minimal program $(x/w)^*$ may be much shorter than the corresponding absolute minimal program x^*, but can only be longer by $O(1)$, independent of x and w.

A string is *compressible* by s bits if its minimal program is shorter by at least s bits than the string itself, i.e. if $|x^*| \leq |x| - s$. Similarly, a string x is said to be compressible by s bits relative to a string w if $|(x/w)^*| \leq |x| - s$.

Regardless of how compressible a string x may be, its minimal program x^* is compressible by at most an additive constant depending on the universal computer but independent of x. [If $(x^*)^*$ were much smaller than x^*, then the role of x^* as minimal program for x would be undercut by a program of the form "execute the result of executing $(x^*)^*$."] Similarly, a relative minimal program $(x/w)^*$ is compressible relative to w by at most a constant number of bits independent of x or w.

The *algorithmic probability* of a string x, denoted $P(x)$, is defined as $\sum\{2^{-|p|} : U(p) = x\}$. This is the probability that the U machine, with a random program chosen by coin tossing and an initially blank data tape, will halt with output x. The *time-bounded algorithmic probability*, $P_t(x)$, is defined similarly, except that the sum is taken only over programs which halt within time t. We use $P(x/w)$ and $P_t(x/w)$ to denote the analogous algorithmic probabilities of one string x relative to another w, i.e. for computations that begin with w on the data tape and halt with x on the data tape.

The *algorithmic entropy* $H(x)$ is defined as the least integer greater than $-\log_2 P(x)$, and the conditional entropy $H(x/w)$ is defined similarly as the least integer greater than $-\log_2 P(x/w)$.

Among the most important properties of the algorithmic entropy is its equality, to within $O(1)$, with the size of the minimal program:

$$\exists c \forall x \forall w H(x/w) \leq |(x/w)^*| \leq H(x/w) + c. \tag{2}$$

The first part of the relation, viz. that algorithmic entropy should be no greater than minimal program size, is obvious, because of the minimal program's own contribution to the algorithmic probability. The second half of the relation is less obvious (cf. [16],[11], and Lemma 2). The approximate equality of algorithmic entropy and minimal program size means that there are few near-minimal programs for any given input/output pair (x/w), and that every string gets an $O(1)$ fraction of its algorithmic probability from its minimal program.

Finite strings, such as minimal programs, which are incompressible or nearly so are called *algorithmically random*. The definition of randomness for finite strings is necessarily a little vague because of the $\pm O(1)$ machine-dependence of $H(x)$ and, in the case of strings other than self-delimiting programs, because of the question of how to count the information encoded in the string's length, as opposed to its bit sequence. Roughly speaking, an n-bit self-delimiting program is considered random (and therefore not ad-hoc as a hypothesis) iff its information content is about n bits, i.e. iff it is incompressible; while an externally delimited n-bit string is considered random iff its information content is about $n + H(n)$ bits, enough to specify both its length and its contents.

For infinite binary sequences (which may be viewed also as real numbers in the unit interval, or as characteristic sequences of sets of natural numbers) randomness can be defined sharply: a sequence X is incompressible, or

algorithmically random, if there is an $O(1)$ bound to the compressibility of its initial segments X_n. This class of infinite sequences was first characterized by Martin-Löf [31]; several equivalent definitions have since been given [27],[12]. Intuitively, an infinite sequence is *random* if it is typical in every way of sequences that might be produced by tossing a fair coin; in other words, if it belongs to no informally definable set of measure zero. *Algorithmically random* sequences constitute a larger class, including sequences such as Ω [11] which can be specified by ineffective definitions. Henceforth, the term "random" will be used in the narrow informal sense, and "incompressible", or "algorithmically random", in the broader, exact sense.

We proceed with a few other useful definitions.

The *busy beaver function* $B(n)$ is the greatest number computable by a self-delimiting program of n bits or fewer.

The *halting set* K is $\{\ x : \varphi_x(x) \text{ converges }\}$. This is the standard representation of the halting problem.

The *self-delimiting halting set* K_0 is the (prefix) set of all self-delimiting programs for the U machine that halt: $\{\ p : U(p) \text{ converges }\}$.

K and K_0 are readily computed from one another (e.g. by regarding the self-delimiting programs as a subset of ordinary programs, the first 2^n bits of K_0 can be recovered from the first $2^{n+O(1)}$ bits of K; by encoding each n-bit ordinary program as a self-delimiting program of length $n + O(\log n)$, the first 2^n bits of K can be recovered from the first $2^{n+O(\log n)}$ bits of K_0.)

The *halting probability* Ω is defined as $\sum \{\ 2^{-|p|} : U(p) \text{ converges }\}$, the probability that the U machine would halt on an infinite input supplied by coin tossing. Ω is thus a real number between 0 and 1.

The first 2^n bits of K_0 can be computed from the first n bits of Ω, by enumerating halting programs until enough have halted to account for all but 2^{-n} of the total halting probability. The time required for this decoding (between $B(n - O(1))$ and $B(n + H(n) + O(1))$ grows faster than any computable function of n. Although K_0 is only slowly computable from Ω, the first n bits of Ω can be rapidly computed from the first $2^{n+H(n)+O(1)}$ bits of K_0, by asking about the halting of programs of the form "enumerate halting programs until (if ever) their cumulative weight exceeds q, then halt", where q is an n-bit rational number.

In the following, we will often be dealing with a prefix set S of strings having some common property, e.g. the set of all self-delimiting programs to

compute x in time t. Two useful lemmas relate the *measure* of such a set

$$\mu[S] = \sum_{p \in S} 2^{-|p|}$$

to the compressibility of its individual members.

Lemma 1: If S is a prefix set of strings whose total measure $\mu[S] = \sum_{x \in S} 2^{-|x|}$ is at least 2^{-m}, and y is an arbitrary string (or the empty string), then at least some members of S must not be compressible by more than m bits relative to $y : \forall_y \exists_{x \in S} |(x/y)^*| \geq |x| - m$.

Proof: Suppose on the contrary that for some S and y, $\mu[S] \geq 2^{-m}$ and $|(x/y)^*| < |x| - m$ for all $x \in S$. Then the set $\{(x/y)^* : x \in S\}$, also a prefix set, would have measure greater than 1, because each of its members is more than m bits shorter than the corresponding member of S. But the Kraft inequality forbids any prefix set from having measure greater than 1; the lemma follows. ∎

Lemma 2: If a prefix set S of strings, having total measure $\mu(S) = \sum_{x \in S} 2^{-|x|}$ less than 2^{-m}, is computable by a self-delimiting program of s bits; then every member of S is compressible by at least $m - s - O(1)$ bits.

Proof Sketch: This lemma can be proved using, for each S, a special-purpose self-delimiting computer C_S designed to compress each member of S by exactly m bits, in other words, to produce each output in S in response to some m-bit-shorter input which is not the prefix or extension of any other input. This special computer is then simulated by the general purpose U machine, at the cost of expanding all its programs by a constant amount (equal to the number of bits required to describe S and m), to obtain the desired result.

The existence of C_S is guaranteed by a more general result [11] (theorem 3.2) proving the existence of a special purpose computer C satisfying any consistent, recursively-enumerable list of requirements of the form $< x(k), n(k) >$ $(k = 0, 1, 2...)$, where the k'th requirement $< x(k), n(k) >$ asks that a self-delimiting program of length $n(k)$ be "assigned" to the output string $x(k)$. The special purpose computer may be thought of more abstractly as a partial recursive function whose range is $\{x(k)\}$ and whose domain is a prefix set. The requirements are called consistent if they obey the Kraft inequality $\sum_k 2^{-n(k)} \leq 1$, and the computer is said to satisfy

them if there are precisely as many programs of length n with output x as there are pairs $< x, n >$ in the list of requirements. In [11] it is shown in more detail how a straightforward "greedy" allocation rule, which attempts to satisfy each requirement in order of recursive enumeration by the first available string of the requested length that is not the prefix or extension of any previously allocated string, works so well that it only fails if the list of requirements would violate the Kraft inequality.

In the present application, the requirements are for uniform compression of each member of the set S, i.e. $n(k) = |x(k)| - m$. If S is computable by a self-delimiting program of s bits, then the feasible degree of compression m can be computed from the same program that computes S. Thus the U simulating the special purpose C_S machine requires an additional program length of s bits; and each member of S is compressible by $k - s - O(1)$ bits. ∎

3 Depth of Finite Strings

We begin by considering finite objects (e.g. strings or natural numbers), where the intuitive motivation for depth is clearer, even though mathematically sharper (polynomially or recursively invariant) results can be obtained with infinite sequences.

We consider several candidates for the best definition of depth:

Tentative Definition 0.1: A string's depth might be defined as the execution time of its minimal program.

The difficulty with this definition arises in cases where the minimal program is only a few bits smaller than some much faster program, such as a print program, to compute the same output x. In this case, slight changes in x may induce arbitrarily large changes in the run time of the minimal program, by changing which of the two competing programs is minimal. Analogous instability manifests itself in translating programs from one universal machine to another. This instability emphasizes the essential role of the quantity of buried redundancy, not as a measure of depth, but as a certifier of depth. In terms of the philosophy-of-science metaphor, an object whose minimal program is only a few bits smaller than its print program is like an observation that points to a nontrivial hypothesis, but with only a low level of statistical confidence.

To adequately characterize a finite string's depth one must therefore consider the amount of buried redundancy as well as the depth its burial. A string's depth at significance level s might thus be defined as that amount of time complexity which is attested by s bits worth of buried redundancy. This characterization of depth may be formalized in several ways.

Tentative Definition 0.2: A string's depth at significance level s be defined as the time required to compute the string by a program no more than s bits larger than the minimal program.

This proposed definition solves the stability problem, but is unsatisfactory in the way it treats multiple programs of the same length. Intuitively, 2^k distinct $(n+k)$-bit programs that compute same output ought to be accorded the same weight as one n-bit program; but, by the present definition, they would be given no more weight than one $(n+k)$-bit program. Multiple programs can be fairly taken into account by the next definition.

Tentative Definition 0.3: A string's depth at signicifcance level s depth might be defined as the time t required for the string's time-bounded algorithmic probability $P_t(x)$ to rise to within a factor 2^{-s} of its asymptotic time-unbounded value $P(x)$.

This formalizes the notion that for the string to have originated by an effective process of t steps or fewer is less plausible than for the first s tosses of a fair coin all to come up heads.

It is not known whether there exist strings that are deep according to def 0.2 but not def 0.3, in other words, strings having no small fast programs, even though they have enough large fast programs to contribute a significant fraction of their algorithmic probability. Such strings might be called deterministically deep but probabilistically shallow, because their chance of being produced quickly in a probabilistic computation (e.g. one where the input bits of U are supplied by coin tossing) is significant compared to their chance of being produced slowly. The question of whether such strings exist is probably hard to answer because it does not relativize uniformly. Deterministic and probabilistic depths are not very different relative to a random coin-toss oracle A (this can be shown by a proof similar to that [4] of the equality of random-oracle-relativized deterministic and probabilistic polynomial time complexity classes); but they can be very different relative to an oracle B deliberately designed to hide information from deterministic computations (this parallels Hunt's proof [23] that deterministic and probabilistic polynomial time are unequal relative to such an oracle).

Although def 0.3 satisfactorily captures the informal notion of depth, we propose a slightly stronger definition for the technical reason that it appears to yield a stronger slow growth property (Theorem 1 below).

Definition 1 (Depth of Finite Strings): Let x and w be strings and s a significance parameter. A string's *depth* at significance level s, denoted $D_s(x)$, will be defined as $\min\{T(p) : (|p| - |p^*| < s) \wedge (U(p) = x)\}$, the least time required to compute it by a s-incompressible program. At any given significance level, a string will be called t-deep if its depth exceeds t, and t-shallow otherwise.

The difference between this definition and the previous one is rather subtle philosophically and not very great quantitatively. Philosophically, def. 1 says that each *individual* hypothesis for the rapid origin of x is implausible at the 2^{-s} confidence level, whereas the previous definition 0.3 requires only that a weighted average of all such hypotheses be implausible. The following lemma shows that the difference betwen def. 1 and def. 0.3 is also small quantitatively.

Lemma 3.: There exist constants c_1 and c_2 such that for any string x, if programs running in time $\leq t$ contribute a fraction between 2^{-s} and 2^{-s+1} of the string's total algorithmic probability, then x has depth at most t at significance level $s + c_1$ and depth at least t at significance level $s - \min\{H(s), H(t)\} - c_2$.

Proof: The first part follows easily from the fact that any k-compressible self-delimiting program p is associated with a unique, $k - O(1)$ bits shorter, program of the form "execute the result of executing p^*". Therefore there exists a constant c_1 such that if all t-fast programs for x were $s + c_1$-compressible, the associated shorter programs would contribute more than the total algorithmic probability of x. The second part of the lemma follows because, roughly, if fast programs contribute only a small fraction of the algorithmic probability of x, then the property of being a fast program for x is so unusual that no program having that property can be random. More precisely, the t-fast programs for x constitute a finite prefix set, a superset S of which can be computed by a program of size $H(x)+\min\{H(t), H(s)\}+O(1)$ bits. (Given x^* and either t^* or s^*, begin enumerating all self-delimiting programs that compute x, in order of increasing running time, and quit when either the running time exceeds t or the accumulated measure of programs so far enumerated exceeds $2^{-(H(x)-s)}$). Therefore there exists a constant c_2 such that, by Lemma 1, every member of S, and thus every t-fast program

for x, is compressible by at least $s - \min\{H(s), H(t)\} - O(1)$ bits. ∎

The ability of universal machines to simulate one another efficiently implies a corresponding degree of machine-independence for depth: for any two efficiently universal machines of the sort considered here, there exists a constant c and a linear polynomial L such that for any t, strings whose $(s+c)$-significant depth is at least $L(t)$ on one machine will have s-significant depth at least t on the other.

Depth of one string relative to another may be defined analogously to definition 1 above, and represents the plausible time required to produce one string, x, from another, w.

Definition 1.1 (Relative Depth of Finite Strings): For any two strings w and x, the *depth of x relative to w* at significance level s, denoted $D_s(x/w)$, will be defined as $\min\{T(p,w) : (|p|-|(p/w)^*| < s) \wedge (U(p,w) = x)\}$, the least time required to compute x from w by a program that is s-incompressible relative to w.

Depth of a string relative to its length is a particularly useful notion, allowing us, as it were, to consider the triviality or nontriviality of the "content" of a string (i.e. its bit sequence), independent of its "form" (length). For example, although the infinite sequence 000... is intuitively trivial, its initial segment 0^n is deep whenever n is deep. However, 0^n is always shallow relative to n, as is, with high probability, a random string of lenth n.

In order to adequately represent the intuitive notion of stored mathematical work, it is necessary that depth obey a "slow growth" law, i.e. that fast deterministic processes be unable to transform a shallow object into a deep one, and that fast probabilistic processes be able to do so only with low probability.

Theorem 1 (Slow Growth Law): Given any data string x and two significance parameters $s_2 > s_1$, a random program generated by coin tossing has probability less than $2^{-(s_2-s_1)+O(1)}$ of transforming x into an excessively deep output, i.e. one whose s_2-significant depth exceeds the s_1-significant depth of x plus the run time of the transforming program plus $O(1)$. More precisely, there exist positive constants c_1, c_2 such that for all strings x, and all pairs of significance parameters $s_2 > s_1$, the prefix set $\{q : D_{s_2}(U(q,x)) > D_{s_1}(x) + T(q,x) + c_1\}$ has measure less than $2^{-(s_2-s_1)+c_2}$.

Proof: Let p be a s_1-incompressible program which computes x in time $D_{s_1}(x)$, and let r be the restart prefix mentioned in the definition of the U

machine. Let Q be the prefix set $\{q : D_{s_2}(U(q,x)) > T(q,x) + D_{s_1}(x) + c_1\}$, where the constant c_1 is sufficient to cover the time overhead of concatenation. For all $q \in Q$, the program rpq by definition computes some deep result $U(q,x)$ in less time than that result's own s_2-significant depth, and so rpq must be compressible by s_2 bits. The sum of the algorithmic probabilities of strings of the form rpq, where $q \in Q$, is therefore

$$\sum_{q \in Q} P(rpq) < \sum_{q \in Q} 2^{-|rpq|+s_2} = 2^{-|r|-|p|+s_2}\mu(Q). \tag{3}$$

On the other hand, since the self-delimiting program p can be recovered from any string of the form rpq (by deleting r and executing the remainder pq until halting occurs, by which time exactly p will have been read), the algorithmic probability of p is at least as great (within a constant factor) as the sum of the algorithmic probabilities of the strings $\{rpq : q \in Q\}$ considered above:

$$P(p) > \mu(Q) \cdot 2^{-|r|-|p|+s_2-O(1)}.$$

Recalling the fact that minimal program size is equal within a constant factor to the $-\log$ of algorithmic probability, and the s_1-incompressibility of p, we have $P(p) < 2^{-(|p|-s_1+O(1))}$, and therefore finally

$$\mu(Q) < 2^{-(s_2-s_1)+O(1)},$$

which was to be demonstrated. ∎

An analogous theorem (with analogous proof) also holds for the improbability of rapid growth of depth relative to an arbitrary fixed string w:

Theorem 1.1 (Relative Slow Growth Law): Given data strings x and w and two significance parameters $s_2 > s_1$, a random program generated by coin tossing has probability less than $2^{-(s_2-s_1)+O(1)}$ of transforming x into an excessively deep output, one whose s_2-significant depth relative to w exceeds the s_1-significant depth of x relative to w plus the run time of the transforming program plus $O(1)$. More precisely, there exist positive constants c_1, c_2 such that for all strings x and w, and all pairs of significance parameters $s_2 > s_1$, the prefix set $\{q : D_{s_2}(U(q,x)/w) > D_{s_1}(x/w) + T(q,x) + c_1\}$ has measure less than $2^{-(s_2-s_1)+c_2}$.

Examples of Shallow Objects:

The trivial string 000... has already been mentioned. It is shallow in the sense that there exists a low-order polynomial L (linear for the machine

model we are using) and a significance parameter s such that for all n, $D_s(0^n/n) < L(n)$. Here s represents the size and $L(n)$ the running time of a fixed program that for any n computes 0^n from n. Similarly, there exist s and L such that for all n and k, a random string x of length n produced by coin tossing satisfies $D_{s+k}(x/n) < L(n)$ with probability greater than $1-2^{-k}$. In this case the shallowness of x is shown by a fast near-incompressible program that simply copies x verbatim off the program tape, using the data n to decide when to stop. This shallowness of random sequences applies not only to those generated by coin tossing, but to ineffectively definable sequences such as the halting probability Ω, whose algorithmic randomness implies that there is a significance level s at which $D_s(\Omega_n/n)$ increases only linearly with n. As emphasized in the introduction, Ω's concentrated information about the halting problem does not make it deep, because the information is encoded so concisely as to appear random.

Examples of Very Deep Objects:

Very deep strings can be constructed by diagonalization, for example, by programs of the form

"Find all n-bit strings whose algorithmic probability, from computations halting within time T (a large number), is greater than 2^{-n}, and print the first string not in this set."

This program runs very slowly, using time about $T\cdot 2^T$ to evaluate the algorithmic probabilities by explicitly simulating all T-bit coin toss sequences, but eventually it outputs a specific string $\chi(n,T)$ guaranteed to have T-fast algorithmic probability less than 2^{-n}, even though the string's relatively concise description via the above algorithm guarantees a slow algorithmic probability of at least $2^{-H(n)-H(T)+O(1)}$. We can then invoke Lemma 3 (relating depth to the rise time of algorithmic probability) to conclude that $\chi(n,T)$ has depth at least T at significance level

$$n - H(T) - \min\{H(n - H(n) - H(T)), H(T)\} - O(1),$$

which (taking into account that in non-vacuous cases $H(T) < n$) is at least $n - H(T) - O(\log n)$.

Because the halting set K has the ability to speed up any slow computation, deep objects such as the diagonal strings considered above are rapidly computable from K. Therefore, by the slow growth law, the halting set must be deep itself. More precisely, by arguments similar to those used in Barzdin's paper [2] on the compressibility of the halting problem, it can

be shown that for any $c < n$, the initial 2^n bits of K have depth between $B(n - c - O(\log n)$ and $B(n - c + O(\log n))$ at significance level 2^c.

It is not hard to see that the busy beaver function provides an approximate upper bound on the depth of finite strings. Given a string x, its length n, and a value of the significance parameter s, all of which can be encoded in a self-delimiting program of size $n - s + O(\log n)$, one can compute the depth $D_s(x)$, which must therefore be less than $B(n - s + O(\log n))$ by the definition of the busy beaver function.

Very deep objects, because they contain information about their own depth, are necessarily ambitious in the broad sense (cf. introduction) of containing information about how to compute large numbers. On the other hand, ambitious objects need not be deep. For example, Ω is ambitious but shallow.

On the Efficient Generation of Depth The diagonal method mentioned above for calculating deep strings suffers from exponential overhead, taking more than 2^T time to generate an object of depth T. One naturally wonders whether there are ways of generating depth with an efficiency more closely approaching the maximum allowed by the slow growth law: depth T in time T.

One way of doing so would be simply to generate a string of *length* T, say a string of T zeros. Time $O(T)$ is clearly both necessary and sufficient to generate this string, but it would be more satisfying to find an example of an efficiently generated object deeper than its own bulk. Only then would the object contain evidence of being the visible part of a larger invisible whole.

Unfortunately it appears that nontrivial and efficient production of depth may depend on plausible but unproven assumptions at the low end of complexity theory. Motivated by the finding that many open questions of complexity theory can be easily shown to have the answers one would like them to have in the relativized context of a random oracle (e.g. $P^A \neq NP^A \neq PSPACE^A$), Bennett and Gill [4] informally conjectured that pseudorandom functions that "behave like" random oracles exist absolutely, and therefore that all "natural" mathematical statements (such as $P \neq NP$) true relative to a random oracle should be true absolutely. Their attempt to formalize the latter idea (by defining a broad class of statements to which it was supposed to apply) was unsuccessful, [26], but the former idea has been formalized quite successfully in the notion of a cryptographically strong pseudorandom function (CSPSRF) [9] [47] [21] [30]. A CSPSRF is a 0/1-valued polynomial time computable function G of two variables with the

property that, if the first variable (s, "the seed") is chosen randomly from among strings of a given length, the resulting function of the second variable $G_s(x)$ cannot be distinguished from a random oracle $A(x)$ in time polynomial in the length of the seed. "Cannot be distinguished" means that there is no probabilistic or deterministic test by which an adversary, ignorant of the seed s, but knowing the algorithm for G and having the ability to evaluate $G_s(x)$ for arbitrary x, could distinguish the pseudorandom oracle G_s from a truly random oracle A, except with low probability, or by using time more than polynomial in the length of the seed. Exponential time of course would be sufficient to distinguish the two oracles with certainty, by searching exhaustively for a seed which exactly reproduced the behavior of one oracle (the pseudorandom one) but not the other.

Below we show how a random oracle, and hence a cryptographically strong pseudorandom function, would permit deep strings to be generated efficiently.

Given a 0/1-valued random oracle A it is routine to construct a random function $\xi_A(x)$ which maps strings randomly onto strings of the same length. The statistical structure of such random length-preserving functions is well known and facilitates the construction, from a standard starting string such as 0^n, of the deep objects considered below.

Let $\xi_A^k(0^n)$ denote the k'th forward image of 0^n, where $k < 2^{n/2}$, under the length-preserving function derived from random oracle A. This string, the target of a chain of k pointers through the random structure of the ξ function, is readily computed if enough time ($O(k \cdot n^2)$ in our model) is allowed to evaluate the ξ function k times; but, for typical random oracles, if less time is allowed, the probability of finding the correct target is only $O(1/2^n)$, representing pure luck. This statement is true for typical oracles; for a small minority of oracles, of measure $O(k^2/2^n)$, the chain of pointers starting from 0^n would begin to cycle in fewer than k iterations, permitting the target to be found more quickly. Returning to the case of a typical oracle, Lemma 3, relating depth to the time-dependence of algorithmic probability, can be used to show that the target string $\xi_A^k(0^n)$ has, at significance level $n - O(\log n)$), depth proportional to the time $O(k \cdot n^2)$ actually used by the straightforward algorithm for computing it.

This result holds, with high probability, in the relativized world containing a random oracle. If one assumes the existence of CSPSRF, then a similar result holds in the real world: using a pseudorandom length-preserving function ξ_{G_s} derived from a CSPSRF with seed s randomly chosen among strings

of length n, one obtains target strings of the form $\xi^k_{G_s}(0^n)$ which can be generated in polynomial time in $k \cdot n$, but with high probability (in the choice of the seed s) have depth exceeding a smaller polynomial in $k \cdot n$. If this were not the case, the pseudorandom G_s could be distinguished from a random A in polynomial time by demonstrating a greater fast algorithmic probability of the target string in the former case.

Another sort of deep object definable with the help of the ξ function are preimages of 0^n, i.e. members of the set $\{x : \xi(x) = 0^n\}$. The number of preimages is binomially distributed, approaching a Poisson distribution for large n, so that a given string (such as 0^n) has no preimages approximately $1/e$ of the time, one preimage $1/e$ of the time, and m preimages $e^{-1}/m!$ of the time. Preimages of 0^n, when they exist, are deep because for random ξ they cannot be found except by exhaustive search. In the terminology of NP problems, a preimage is "witness" for membership of 0^n in the range of ξ_A, a set which, relative to a random oracle A belongs to $NP^A - P^A$.

Remarks on the Transitivity of Depth

The slow growth law says that deep objects cannot quicky be produced from shallow ones. It is natural to wonder whether this property can be extended to a transitive law for relative shallowness; in other words, if x is shallow relative to w, and y is shallow relative to x, does it follow that y is shallow relative to w?

The answer is no, as can be seen from the following example: Let w be a random string (produced e.g. by coin tossing), x be the empty string, and y be the bitwise exclusive-or of w with some deep string d. Then y is also random and shallow, and so shallow relative to x, as x is relative to w; however y is deep relative to w, since d can easily be regenerated from y and w. Therefore simple transitivity does not hold.

A more cumulative sort of transitivity can be shown to hold: for all w, x, and y, if x is shallow relative to w, and y is shallow relative to the ordered pair $< w, x >$, then y is indeed shallow relative to w, at least within logarithmic error terms in the significance parameter such as those in Lemma 3. In particular, cumulative transitivity holds when w is empty: if x is shallow and y is shallow relative to x, then y is shallow.

Scalar Measures of Depth

One may well wonder whether, by defining some sort of weighted average run time, a string's depth might be expressed by a single number, unqualified by a significance parameter. This may be done, at the cost of imposing a

somewhat arbitrary rate of exchange between the two conceptually very different quantities run time and program size. Proceeding from tentative def. 0.3 above, one might try to define a string's average depth as the average run time of all computations contributing to its algorithmic probability, but this average diverges because it is dominated by programs that waste arbitrarily much time. To make the average depth of x depend chiefly on the fastest programs of any given size that compute x, one can use the "harmonic mean," or reciprocal mean reciprocal, run time in place of a straight average. The *reciprocal mean reciprocal depth* of a string x may thus be defined as

$$D_{rmr}(x) = \frac{\sum\{2^{-|p|} : (U(p) = x)\}}{\sum\{(2^{-|p|}/T(p)) : (U(p) = x)\}}. \tag{4}$$

In this definition, the various computations that produce x act like parallel resistors, the fast computations in effect short-circuiting the slow ones. [The ratio of rmr depth to algorithmic probability, called "hitting"time, was introduced by Levin [29] to measure the difficulty of solving NP-type problems by an optimal search algorithm; related ideas are explored in [1]]. Due to the short-circuiting of slower programs, no matter how small, by the print program, rmr depth doesn't allow strings to have depth more than exponential in their length; however, it does provide a simple quantitative measure of a string's nontriviality. Among efficiently universal machines, it is machine-independent to within a polynomial depending on the machines.

The denominator of the above formula for rmr depth is like the numerator, except that it penalizes each program according to the logarithm of its run time. By using a more slowly growing penalty function, the inverse busy beaver function, one obtains another unparameterized depth measure which may be more suitable for very deep objects.

$$D_{bb}(x) = \min\{s + k : D_s(x) < B(k)\}. \tag{5}$$

This depth measure is closely related to the quantity called sophistication by Koppel [25].

4 Depth of Infinite Sequences

In attempting to extend the notion of depth from finite strings to infinite sequences, one encounters a familiar phenomenon: the definitions become sharper (e.g. recursively invariant), but their intuitive meaning is less clear, because of distinctions (e.g. between infintely-often and almost-everywhere

properties) that do not exist in the finite case. We present a few definitions and results concerning depth of infinite objects.

An infinite sequence X is called *strongly deep* if at every significance level s, and for every recursive function f, all but finitely many initial segments X_n have depth exceeding $f(n)$.

It is necessary to require the initial segments to be deep almost everywhere rather than infinitely often, because even the most trivial sequence has infinitely many deep initial segments X_n (viz. the segments whose *lengths* n are deep numbers).

It is not difficult to show that the property of strong depth is invariant under truth-table equivalence [38] (this is the same as Turing equivalence in recursively bounded time, or via a total recursive operator), and that the same notion would result if the initial segments were required to be deep in the sense of receiving less than 2^{-s} of their algorithmic probability from $f(n)$-fast programs. The characteristic sequence of the halting set K is an example of a strongly deep sequence.

A weaker definition of depth, also invariant under truth-table equivalence, is perhaps more analogous to that adopted for finite strings:

An infinite sequence X is *weakly deep* if it is not computable in recursively bounded time from any algorithmically random infinite sequence.

As remarked above, computablility in recursively bounded time is equivalent to two other properties, viz. truth-table reducibility and reducibility via a total recursive operator. These equivalences are not hard to demonstrate. We will call this reducibility by the traditional name of truth-table reducibility, even though the other two characterizations may be more intuitive.

By contrast to the situation with truth-table reducibility, Gacs has recently shown [20] that every sequence is computable from (i.e. Turing reducible to) an algorithmically random sequence if no bound is imposed on the time. This is the infinite analog of far more obvious fact that every finite string is computable from an algorithmically random string (e.g. its minimal program).

Every strongly deep sequence is weakly deep, but by intermittently padding K with large blocks of zeros, one can construct a weakly deep sequence with infinitely many shallow initial segments.

Truth table reducibility to an algorithmically random sequence is equivalent to the property studied by Levin et. al. of being random with respect to some recursive measure. Levin calls sequences with this property "proper" [48] or "complete" [28][29] sequences (we would call them strongly shallow),

and views them as more realistic and interesting than other sequences because they are the typical outcomes of probabilistic or deterministic effective processes operating in recursively bounded time. Deep sequences, requiring more than recursively bounded time to generate, and especially ineffectively defined sequences such as K or Ω, he regards as unnatural or pathological by comparison. We take a somewhat opposing view, regarding objects of recursively unbounded depth as perhaps useful analogs for the less deep objects that may be found in nature.

V'jugin [44] has shown that weakly deep sequences arise with finite probability when a universal Turing machine (with one-way input and output tapes, so that it can act as a transducer of infinite sequences) is given an infinite coin toss sequence for input. These sequences are necessarily produced very slowly: the time to output the n'th digit being bounded by no recursive function, and the output sequence contains evidence of this slowness. Because they are produced with finite probability, V'jugin sequences can contain only finite information about the halting problem. This contrasts with the finite case, where deep strings necessarily contain information about K. It is not known whether all strongly deep strings contain infinite information about K.

5 Depth and Complexity in Physics

Here we argue that logical depth is a suitable measure of subjective complexity for physical as well as mathematical objects, and consider the effect of irreversibility, noise, and spatial symmetries of the equations of motion and initial conditions on the asymptotic depth-generating abilities of model systems. Many of the ideas mentioned here are treated at greater length in [36] and [8], and [7].

"Self-organization" suggests a spontaneous increase of complexity occuring in a system with simple, generic (e.g. spatially homogeneous) initial conditions. The increase of complexity attending a computation, by contrast, is less remarkable because it occurs in response to special initial conditions. This distinction has been highlighted recently by the discovery of models (e.g. classical hard spheres moving in an appropriate periodic array of obstacles [13][33]) which are computationally universal on a subset of initial conditions, but behave in a quite trivial manner for more general initial conditions.

An important question, which would have interested Turing, is whether

self-organization is an asymptotically qualitative phenomenon like phase transitions. In other words, are there physically reasonable models in which complexity, appropriately defined, not only increases, but increases without bound in the limit of infinite space and time? A positive answer to this question would not explain the natural history of our particular finite world, but would suggest that its quantitative complexity can legitimately be viewed as an approximation to a well-defined qualitative property of infinite systems. On the other hand, a negative answer would suggest that our world should be compared to chemical reaction-diffusion systems (e.g. Belousov-Zhabotinsky), which self-organize on a macroscopic, but still finite scale, or to hydrodynamic systems (e.g Benard) which self-organize on a scale determined by their boundary conditions. A thorough understanding of the physical prerequisites for qualitative self-organization may shed some light on the difficult issue of the extent to which our world's observed complexity is conditioned by the posterior existence of sentient observers.

The suitability of logical depth as a measure of physical complexity depends on the assumed ability ("physical Church's thesis") of Turing machines to simulate physical processes, and to do so with reasonable efficiency. Digital machines cannot of course integrate a continuous system's equations of motion exactly, and even the notion of computability is not very robust in continuous systems (e.g. a computable, differentiable function can have a non-computable derivative [34]) but for realistic physical systems, subject throughout their time development to finite perturbations (e.g. electromagnetic and gravitational) from an uncontrolled environment, it is plausible that a finite-precision digital calculation can approximate the motion to within the errors induced by these perturbations. Empirically, many systems have been found amenable to "master equation" treatments in which the dynamics is approximated as a sequence of stochastic transitions among coarse-grained microstates [43]. Presumably, many mundane hydrodynamic and chemical systems could be efficiently simulated by discrete stochastic models using this approach, if the mesh size were made fine enough and the number of states per site large enough.

To see how might depth be used to measure physical complexity, consider an infinite hard shere gas at equilibrium. The intuitive triviality of this system can be formalized by observing that the coarse-grained state of a typical region in the gas (say a specification, at p digits precision, of the positions and velocities of all the particles in a region of diameter l) has depth bounded by a small polynomial (in lp). Since the gas is at equilibrium, its depth does not increase with time. Now consider the same gas with a

nonequilibrium initial condition, e.g. a periodic modulation of the density. The depth of a local region within the gas would now increase with time, representing the duration of the plausible evolution connecting the present configuration with the significantly nonrandom initial condition.

Now let the evolution of the gas be subject to noise. The regional depth would increase for a while as before, but eventually the significance parameter characterizing this depth would fall to near zero, as the noise gradually obliterated the region's correlation with the system's nonrandom initial condition. Thus, in the realistic presence of noise, the hard sphere gas with nonequilibrium initial condition is not self-organizing.

We do not attempt to survey the vast range of mathematical models used in physics, within which computationally complex or self-organizing behavior might be sought, but instead concentrate somewhat arbitrarily on cellular automata, in the broad sense of discrete lattice models with finitely many states per site, which evolve according to a spatially homogeneous local transition rule that may be deterministic or stochastic, reversible or irreversible, and synchronous (discrete time) or asynchronous (continuous time, master equation). Such models (cf. the recent review by Wolfram [46]) cover the range from evidently computer-like (e.g. deterministic cellular automata) to evidently material-like (e.g. Ising models) with many gradations in between.

In general it appears that reversibility, noise, asynchrony, and spatial reflection-symmetry of the dynamical law hinder computation, whereas their opposite properties facilitate it. Generic values of model parameters (e.g. coupling constants, transition probabilities) and generic initial conditions also tend to hinder computation, whereas special parameters and special initial conditions facilitate it. A further variable is the complexity of the (finite) rule, in particular the number of states per site. Complex rules, of course, give more scope for nontrivial behavior, but are less physically realistic.

Various tradeoffs are possible, and not all the favorable properties need be present at once to obtain nontrivial computation. The billiard ball cellular automaton of Margolus [33], for example, though simple, reversible, and reflection-symmetric, is computationally universal. The 3-dimensional error-correcting automaton of Gacs and Reif [19], on the other hand, is computationally universal in the presence of generic noise, but is irreversible and lacks reflection symmetry. Gacs' one-dimensional automaton [17] has similar qualitative properties, but is also enormously complex.

More of the favorable properties need to be invoked to obtain "self-organization," i.e. nontrivial computation from a spatially homogeneous

initial condition. In [8] we described a rather artifical system (a cellular automaton which is stochastic but noiseless, in the sense that it has the power to make purely deterministic as well as random decisions) which undergoes this sort of self-organization. It does so by allowing the nucleation and growth of domains, within each of which a depth-producing computation begins. When two domains collide, one conquers the other, and uses the conquered territory to continue its own depth-producing computation (a computation constrained to finite space, of course, cannot continue for more than exponential time without repeating itself). To achieve the same sort of self-organization in a truly noisy system appears more difficult, partly because of the conflict between the need to encourage fluctuations that break the system's translational symmetry, while suppressing fluctuations that introduce errors in the computation.

Irreversibility seems to facilitate complex behavior by giving noisy systems the generic ability to correct errors. Only a limited sort of error-correction is possible in microscopically reversible systems such as the canonical kinetic Ising model. Minority fluctuations in a low-temperature ferromagnetic Ising phase in zero field may be viewed as errors, and they are corrected spontaneously because of their potential energy cost. This error correcting ability would be lost in nonzero field, which breaks the symmetry between the two ferromagnetic phases, and even in zero field it gives the Ising system the ability to remember only one bit of information. This limitation of reversible systems is recognized in the Gibbs phase rule, which implies that under generic conditions of the external fields, a thermodynamic system will have a unique stable phase, all others being metastable. Irreversible noisy systems escape this law [7], being able to store information reliably, and perform reliable computations [19], even when the noise is biased so as to break all symmetries.

Even in reversible systems, it is not clear why the Gibbs phase rule enforces as much simplicity as it does, since one can design discrete Ising-type systems whose stable phase (ground state) at zero temperature simulates an aperiodic tiling [37] of the plane, and can even get the aperiodic ground state to incorporate (at low density) the space-time history of a Turing machine computation. Even more remarkably, one can get the structure of the ground state to diagonalize away from all recursive sequences [32]. These phenomena have been investigated from a physical viewpoint by Radin and Miekisz [35]; it is not known whether they persist at finite temperature, or in the presence of generic perturbations of the interaction energies.

Instead of inquiring, as earlier, into the asymptotic time-dependence of

depth in infinite systems, we can ask how much depth can be generated by a discrete system of size n, given unlimited time. The answer depends on open questions in computational complexity theory. Assuming the existence of cryptographically strong pseudorandom functions which require only polynomial time and space to compute, such a system, if it is capable of universal computation, can generate states exponentially deep in n; the production of greater depth (except accidentally, with low probability) is forbidden by the system's Poincaré recurrence. On the other hand, if the dismal state of affairs $P = PSPACE$ holds, only polynomially deep states can be produced.

6 Acknowledgements

The development of these ideas is the outcome of thoughtful discussions and suggestions, extending over two decades, from Ray Solomonoff, Gregory Chaitin, Rolf Landauer, Dexter Kozen, Gilles Brassard, Leonid Levin, Peter Gacs, Stephen Wolfram, Geoff Grinstein, Tom Toffoli, Norman Margolus, and David Griffeath, among others.

References

[1] L. Adleman, "Time, Space, and Randomness", MIT Report LCS/TM-131, April 1979.

[2] Ja. M. Barzdin "Complexity of Programs to Determine Whether Natural Numbers not Greater than n Belong to a Recursively Enumerable Set". *Sov. Math. Doklady* **9**, 1251-1254 (1968).

[3] C.H. Bennett, "On Random and Hard-to-Describe Numbers", IBM Report RC 7483 (1979), and Martin Gardner, "Mathematical Games" *Sci. Amer* 20-34 (November 1979) based on this report.

[4] C.H. Bennett and J. Gill, "Relative to a Random Oracle A, $P^A \neq NP^A \neq co{-}NP^A$ with Probability 1", *SIAM J. Comput.* **10**, 96-113 (1981).

[5] C.H. Bennett, "On the Logical Depth of Sequences and Their Reducibilities to Random Sequences" unpublished manuscript (1982).

[6] C.H. Bennett, "Information, Dissipation, and the Definition of Organization", in *Emerging Syntheses in Science*, David Pines ed. (Santa Fe Institute, Santa Fe, NM) 297-313 (1985).

[7] C.H. Bennett and G. Grinstein *Phys.Rev.Lett* **55**, 657-660, 1985.

[8] C.H. Bennett, "On the Nature and Origin of Complexity in Discrete, Homogeneous, Locally-Interacting Systems", *Foundations of Physics* **16** 585-592 (1986).

[9] M. Blum and S. Micali "How to Generate Cryptographically Strong Sequences of Pseudo Random Bits," *SIAM J. on Computing*, **13** 850-864 (1984).

[10] J.-L. Borges, *Labyrinths: Selected Stories and Other Writings*, D.A. Yates and J.E. Irby editors, (New Directions, New York 1964)

[11] G. Chaitin, "A Theory of Program Size Formally Identical to Information Theory", *J. Assoc. Comput. Mach.* **22**, 329-340 (1975). Cf also G.J. Chaitin *Algorithmic Information Theory*, (Cambridge Univ. Press, Cambridge, England 1987)

[12] G. Chaitin, "Algorithmic Information Theory", *IBM J. Res. Develop.* **21**, 350-359, 496, (1977). Cf also *Information, Randomness, and Incompleteness—Papers on Algorithmic Information Theory*, (World Scientific Press, Singapore 1987).

[13] E. Fredkin and T. Toffoli, *Internat. J. of Theo. Phys.*, **21**, 219 (1982).

[14] Gardner, Martin, "Mathematical Games" *Scientific American* 20-34 (November 1979)

[15] M. Garey and D. Johnson, *Computers and Intractability, a Guide to NP Completeness* (Freeman, 1979).

[16] P. Gacs, "On the Symmetry of Algorithmic Information", *Soviet Math Dokl.* **15**, 1477 (1974).

[17] P. Gacs, Technical Report No 132, Computer Science Dept. U. of Rochester (1983), to appear in J. of Computer and System Science.

[18] P. Gacs, "On the relation between descriptional complexity and probability" *Theo. Comp. Sci.* **22**, 71-93 (1983).

[19] P. Gacs and J. Reif, *Proc. 17th ACM Symposium on the Theory of Computing*, 388-395 (1985).

[20] P. Gacs, "Every Sequence is Reducible to a Random Sequence" *Info. and Control* **70**, 186-192 (1986).

[21] O. Goldreich, S. Goldwasser, and S. Micali "How to Construct Random Functions" *Proc. 25th IEEE Symp. on Found. of Computer Science*, (1984).

[22] J. Hartmanis, "Generalized Kolmogorov Complexity and the Structure of Feasible Computations", *Proc. 24'th IEEE Symp. on Found. of Computer Science*, 439-445 (1983).

[23] J. W. Hunt, "Topics in Probabilistic Complexity", PhD dissertation Stanford Univ. Electrical Engineering (1978).

[24] A.N. Kolmogorov, "On Tables of Random Numbers," *Sankhya, the Indian Journal of Statistics* **A25**, 369-376 (1963), "Three Approaches to the Quantitative Definition of Information," *Problems in Information Transmission* **1** 1-7 (1965).

[25] M. Koppel, this book

[26] S. Kurtz "On the Random Oracle Hypothesis" *Info. and Control* (1983).

[27] L. A. Levin "On the Notion of a Random Sequence" *Sov. Math. Doklady* **14**, 1413-1416 (1973).

[28] L. A. Levin and V. V. V'jugin, "Invariant Properties of Informational Bulks", *Springer Lecture Notes in Computer Science* **53**, 359-364 (1977).

[29] L. A. Levin "Randomness Conservation Inequalities: Information and Independence in Mathematical Theories," *Info. and Control* **61**, 15-37 (1984); preliminary draft *MIT Technical Report MIT/LCS/TR-235* (1980).

[30] L. Levin "One-Way Functions and Pseudorandom Generators" *ACM Symposium on Theory of Computing*, (1985).

[31] P. Martin-Löf, "Definition of Random Sequences", *Information and Control* **9**, 602-619 (1966).

[32] Dale Myers, "Nonrecursive Tilings of the Plane II," *J. Symbolic Logic* **2**, 286-284 (1974).

[33] N. Margolus, *Physica* **10D**, 81-95 (1984).

[34] J. Myhill, *Michigan Math J.* **18**, 97-98 (1971).

[35] Charles Radin, *J. Math. Phys.* **26**, 1342 (1985) and *J. Stat. Phys.* **43**, 707 (1986); Jacek Miekisz and Charles Radin, "The Unstable Chemical Structure of Quasicrystalline Alloys" preprint 1986

[36] Bennett, C.H., "Information, Dissipation, and the Definition of Organization", in *Emerging Syntheses in Science*, David Pines ed., Addison-Wesley (1987).

[37] Raphael M. Robinson, *Inv. Math.* **12**, 177-209 (1971).

[38] Hartley Rogers, Jr., "Theory of Recursive Functions and Effective Computability" McGraw-Hill (New York, 1967) pp. 141, 255.

[39] C.P. Schnorr "Process Complexity and Effective Random Tests" *J. Comput and Syst. Scis.* **7**, 376-388, (1973).

[40] R. J. Solomonoff, "A Formal Theory of Inductive Inference," *Inform. and Control* **7**, 1-22 (1964).

[41] A. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proc. London Math. Soc.* **42**, 230-265 and **43**, 544-546 (1936); "Computability and λ-definability," *Journal of Symbolic Logic* **2**, 153-163 (1937).

[42] A. Turing, "A Chemical Theory of Morphogenesis" (1952)

[43] N. van Kampen in *Fundamental Problems in Statistical Mechanics*, edited by E.G.D. Cohen (North-Holland, 1962).

[44] V.V. V'jugin, "On Turing invariant Sets" Sov. Math. Doklady (1976)

[45] Stephen Wolfram *Scientific American* (ca 1985).

[46] Stephen Wolfram *Theory and Applications of Cellular Automata* (World Scientific Press 1986).

[47] A. Yao, "Theory and Applications of Trapdoor Functions" *Proc. 23rd IEEE Symp. on Found. of Computer Science*, 80-91 (1982).

[48] A.K. Zvonkin and L.A. Levin, "The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms", *Russ. Math. Surv.* **256** 83-124 (1970).