CDMTCS
Research
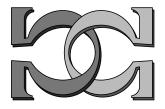**Report**
Series

Truth and Light: Physical Algorithmic Randomness

Michael A. Stay
University of Auckland

Centre for Discrete Mathematics and
Theoretical Computer Science

Department of Computer Science
The University of Auckland
New Zealand

Truth and Light: Physical Algorithmic Randomness

Michael A. Stay

July 2005

Supervisor:

Cristian Calude

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE

Abstract

This thesis examines some problems related to Chaitin's Ω number. In the first section, we describe several new minimalist prefix-free machines suitable for the study of concrete algorithmic information theory; the halting probabilities of these machines are all Ω numbers. In the second part, we show that when such a sequence is the result given by a measurement of a system, the system itself can be shown to satisfy an uncertainty principle equivalent to Heisenberg's uncertainty principle. This uncertainty principle also implies Chaitin's strongest form of incompleteness.

In the last part, we show that Ω can be written as an infinite product over halting programs; that there exists a "natural," or base-free formulation that does not (directly) depend on the alphabet of the universal prefix-free machine; that Tadaki's generalized halting probability is well-defined even for arbitrary univeral Turing machines and the plain complexity; and finally, that the natural generalized halting probability can be written as an infinite product over primes and has the form of a zeta function whose zeros encode halting information. We conclude with some speculation about physical systems in which partial randomness could arise, and identify many open problems.

Acknowledgements

Contents

Introduction

Truth and light are intimately related. The parable of Maxwell's demon beautifully illustrates that information about a system allows one to extract energy from the system, that truth can be exchanged for light and vice-versa.

The bits of the halting probability of a universal prefix-free machine, Chaitin's Ω, are pure information [20, 28]; the first few thousand bits of any of the machines we discuss here contain, in principle, the answers to all of the Clay Millenium problems [33], including such important ones as Riemann's hypothesis.

This thesis explores some of the properties of the algorithmic complexity, both plain and prefix-free, of halting probabilities. Since there are only two prefix-free languages we know of in the literature, in chapter one, we propose a few minimalist prefix-free machines suitable for study. In chapters two and three, we go on to examine some formal relationships between physical quantities and random sequences.

In chapter two, we consider the case where a completely random sequence describes a delta function with respect to some continuous basis—an eigenstate of a quantum system. We show by information-theoretic arguments that the accuracy to which the delta function is measured and the complexity of the initial prefix of the string are noncommuting observables, and that such an uncertainty relation leads to a particularly concise proof of Chaitin's strongest incompleteness result, that there are infinitely many unprovable true statements.

In chapter three, we show that Ω can be written as an infinite product as well as an infinite sum; that there exists a base-independent, or "natural," formulation of Ω; that the concept of partial randomness extends to plain complexity; and that when we combine these three concepts, we find that the natural generalized halting probability of a universal Turing machine is a zeta function. This zeta function is identical to Riemann's zeta function with the exception that the product is taken over "halting primes."

We conclude with some observations about quantum systems and several open questions. Specifically, we note the interpretation of Riemann's zeta function as a partition function of a quantum system, and find that the natural generalized halting probability can be interpreted as the limiting expectation value of a computably enumerable series of observables on this system.

1

Very simple prefix-free machines for concrete AIT

In 1948, Shannon published his seminal paper on *information theory* [58]. In Shannon's model, there is a sender, a receiver, and a (probably noisy) channel, or pipe. The sender transmits strings of bits over the channel, and the receiver records them on arrival. To let the receiver know when to stop listening, the message must have a certain pre-arranged structure. The message may end with a binary equivalent of "over and out," or perhaps the length of the message is sent first. What matters is that as soon as the last bit is sent, the reciever can calculate that that bit was the end of the message; such structures are known as *instantaneous codes*.

Not all instantaneous codes are created equal: some can send the same information with far fewer bits than others. The sender and receiver can agree on a *dictionary* that associates common strings with short encodings and infrequent strings with longer ones. If the message is typical, this will save considerably on the message size. If the message is atypical, there's more information in it, more that is unexpected. The shortest possible message size is related to the amount of information the string contains relative to the chosen dictionary, the string's *complexity*.

In the mid-1960's, Kolmogorov [48], Solomonoff [59], and Chaitin [19] independently proposed the idea of using programs to describe the complexity of strings. This gave birth to the field of *algorithmic information theory* (AIT) [22]. The Kolmogorov complexity $K_U(s)$ of a string s is the length of the shortest program in the programming language U whose output is s. Solomonoff proposed a weighted sum over the programs, but it was dominated by the shortest program, so his approach and Kolmogorov's are roughly equivalent. Chaitin added the restriction that the programs themselves must be codewords in an instantaneous code, giving rise to *prefix-free AIT*. In this model, complexities become true probabilities, and Shannon's information theory applies directly.

Given a prefix-free domain, there is the natural distribution $P(x) = 2^{-|x|}$, where $|x|$ is the length of x in bits. One can then ask the question, "What is the probability, given this distribution of inputs, that a program will output *some* string and halt?" Chaitin discovered that the bits of this "halting

probability", after an initial computable prefix, are pure information [28]: the length of the shortest program that computes the first n bits of the halting probability and stops is at least $n - c$ bits long. To calculate one more bit, you have to add at least one more bit to your program; there is no description of the strings of bits shorter than the strings themselves, modulo some fixed constant. The bits also contain all the information about whether each program will halt or not; Chaitin called this halting probability an Omega number.

With the advent of accessible computers, Chaitin proposed studying *concrete* AIT—theorems about specific programming languages, with positive integers as error terms rather than the phrase "some fixed constant." To study concrete prefix-free AIT, Chaitin proposed two universal languages, or machines: one was a variant of LISP [28]; the other an 11-instruction "register machine" [24]. We know of no other universal prefix-free machines in the literature. LISP is a complex language, and the register machine, while small, is far from minimal. To foster the study of concrete AIT, we propose a few new minimalist prefix-free machines.

1.1   Definitions

Fix an alphabet Σ. The set of all finite strings of elements of Σ is denoted Σ^*. A Turing machine M is a partial recursive function $M : \Sigma^* \to \Sigma^*$, where "partial" means that M may be undefined on some inputs; this handles the cases where the program runs forever.

A Turing machine M is called a *prefix-free machine* if its domain

$$\mathrm{dom}(M) \equiv \{x \in \Sigma^* : M(x) \text{ halts}\}$$

is prefix-free, i.e. for all $x, y \in \mathrm{dom}(M)$, either x is a prefix of y, y is a prefix of x, or x is identical to y.

For any $x \in \Sigma^*$ and prefix-free machine M, the program-size complexity of x with respect to M is defined as

$$H_M(x) \equiv \min\{|w| : M(w) = x\}.$$

A prefix-free machine U is called *universal* with respect to a set of machines $\mathbf{S}$ if $H_U(x) \le H_M(x) + O(1)$ for any $x \in \Sigma^*$ and for any $M \in \mathbf{S}$.

It is helpful to consider a prefix-free machine in Shannon's original sender-pipe-receiver model. Borrowing terminology from concurrent programming, the pipe is a shared resource. The input to the machine is held by the sender, a producer. The sender tries to put its bits into the pipe; it blocks if there are more bits to send and the pipe is full. When there are no more bits to send, the sender halts. The prefix-free machine is the receiver, a consumer. From time to time it tries to get bits out of the pipe, and blocks if the pipe is empty. The entire computation is said to halt if the sender halts, the prefix-free machine halts, and the pipe is empty. *Codewords* are those inputs x for which the computation halts, *i.e.* $x \in \mathrm{dom}(M)$.

This model makes it easy to see why the domain of a prefix-free machine is prefix-free: any extension of a codeword would cause the sender to block, a condition called *overflow*; any prefix of it would cause the prefix-free machine to block, a condition called *underflow*. A universal prefix-free machine usually reads in a self-delimiting program description, the *prefix*, and then simulates that program acting on the remainder of the input. If the input does not contain a proper program description, the universal machine blocks; we call this condition a *syntax error*.

A *blank-endmarker machine* (BEM) B is a prefix-free machine in which one symbol of Σ—the blank endmarker, hereafter denoted $\diamond$—is reserved. All but the last symbol of the codeword are taken from the

alphabet $(\Sigma - \{\diamond\})$, and the codeword is terminated with $\diamond$. B will request input until it reads the symbol $\diamond$, after which no more input will be requested. It is this blank endmarker that allows the codewords to be prefix-free: removing the endmarker will cause an underflow; any symbols following an endmarker will cause an overflow.

We may construct a BEM M' from an arbitrary Turing machine M defined on the alphabet Σ as follows. First, define the alphabet $\Sigma' \equiv \Sigma \cup \{\diamond\}$ over which M' operates. Next, define $\mathrm{dom}(M') \equiv \{x \diamond | x \in \mathrm{dom}(M)\}$, i.e. we append the symbol $\diamond$ to each string on which M halts. No prefix or extension of a codeword is in the domain of M', since every codeword in the domain has exactly one $\diamond$ as the last symbol.

A *universal BEM* is a BEM defined on the alphabet Σ' that is universal with respect to all such BEMs.

Theorem 1. *A universal BEM B_U exists.*

Proof. Let 0^n denote the concatenation of n 0 symbols. Then B_U can read in in the self-delimiting prefix $0^n 1$ and simulate B_n on the remainder of the input. $\square$

Theorem 2. *No BEM is universal with respect to all prefix-free machines defined on Σ'.*

Proof. For a BEM to be universal over a set S, it must be able to represent the domain of machines in S with only a constant increase in the length of the codewords. However, this is impossible.

Consider the following prefix-free machine C: inputs are concatenations of a self-delimiting program p and a string $x \in \Sigma'^*$. The program p, when executed, outputs the length of the string, $n = |x|$. The machine C first reads p, then executes it to get n, and then reads x. Finally, C outputs the string x.

The number of n-symbol strings $|\Sigma'|^n = (|\Sigma| + 1)^n$. On the other hand, the number of $(n+c)$-symbol strings available to a BEM is only $|\Sigma|^{n+c}$, because it may only use the symbol $\diamond$ once at the end of the codeword. Since $(|\Sigma| + 1)^n$ grows faster than $|\Sigma|^{n+c}$, there is no constant c such that $|\Sigma|^{n+c} \geq (|\Sigma| + 1)^n$ for all n. $\square$

We define the relation "universal-with-respect-to" and denote it $\succeq$. Let $\Sigma_n \equiv \{0, 1, \ldots, (n-1)\}$ and $\Sigma'_n \equiv \{0, 1, \ldots, (n-1), \diamond\}$. For all $n \geq 2$ we have the following:

Theorem 3. *Let A_n be prefix-free machine that is universal with respect to all prefix-free machines defined over Σ_n, B_n be a BEM that is universal with respect to all BEMs defined over Σ'_n, and C_n be a prefix-free machine that is universal with respect to all prefix-free machines defined over Σ'_n. Then*

1. $C_n \succeq B_n \succeq A_n$, but

2. $B_n \not\succeq C_n$

3. and $A_n \not\succeq B_n$.

Proof. The first part of (1) holds because BEMs are prefix-free machines and C_n was chosen to be universal with respect to that set.

The second part of (1) holds because B_n can simulate A'_n, the BEM constructed from A_n: it reads a self-delimiting program for A_n and begins simulating it. If the program requests an input and B_n reads $\diamond$, then B_n loops forever, simulating the underflow condition. If the program halts, then B_n reads one more symbol, x; if $x \neq \diamond$, then B_n loops forever, simulating overflow. If the program loops forever on its own, then so does B_n. Thus the domain of B_n is the same as that of A_n modulo the final $\diamond$.

Item (2) is theorem 2.

Finally, (3) holds because A_n is not allowed to use $\diamond$. It must use a self-delimiting description of the input string, and the shortest self-delimiting version of a string x grows like $|x| + log^*|x| + O(1)$ [8], which violates the error bound for universality. $\square$

1.2   Some minimalist machines

In this section, we review four languages that greatly influenced our designs, and point out why these are not universal prefix-free machines.

1.2.1   Lambda calculus

Lambda calculus formed the basis of Church's 1936 negative answer [31, 32] to Hilbert's *Entscheidungsproblem* (decision problem): is there an algorithm for deciding whether first-order statements are universally valid? He showed first that stating the equivalence of two lambda terms was a first-order predicate, and then that there is no recursive (or computable) function that can compute whether two terms are equivalent. Turing independently proved the same result the same year [68], and when he heard of Church's result, was quickly able to show that his machines compute the same class of functions as Church's lambda terms.

Everything in lambda calculus is a function; there are no built-in types, data structures, branching instructions, or constants, and the only operation is functional composition, or *application*. Functions take functions as input and return functions as output. To denote his functions, Church used a slightly different notation than most mathematicians are used to: rather than

$$f(x, y, z) = \langle \text{definition of } f \text{ in terms of } x, y, z \rangle,$$

Church wrote*

$$f = \lambda xyz.\langle \text{definition of } f \text{ in terms of } x, y, z \rangle.$$

Application is denoted by concatenation or parentheses, and is left-associative: $f = \lambda xy.y(xx)y$ is the same as $f(x, y) = (y(x(x)))(y)$.

There is a *universal basis* consisting of the two functions (or *combinators*)

$$S = \lambda xyz.xz(yz) \quad \text{and} \quad K = \lambda xy.x$$

That is, the function represented by a lambda term may also be represented by a combination of these combinators. For example, the identity function $I = SKK$:

$$
\begin{aligned}
SKKv &= (\lambda xyz.xz(yz))KKv \\
&= Kv(Kv) \\
&= (\lambda xy.x)v(Kv) \\
&= v
\end{aligned}
$$

In fact, there is an algorithm called *lambda abstraction* that reduces any lambda term to a combination of S and K, and I combinators† that eliminates the need for any variables. To abstract away all the variables in a term, begin with the innermost variable v and apply the following rules, then repeat for the remaining variables.

1. abstract$(XY) = S(\text{abstract}(X))(\text{abstract}(Y))$

2. abstract$(v) = I$

---

*Strictly speaking, even the equals operator is just syntactic sugar: lambda terms are *anonymous* functions.

†The combinator I is included merely for convenience; it can, of course, be replaced by SKK.

3. if a term X does not depend on v, then abstract$(X) = KX$

For example, the reverse-application combinator is $\lambda xy.yx$. Abstracting λy yields $\lambda x.SI(Kx)$; abstracting λx yields $S(K(SI))(S(KK)I)$.

A term is said to be in *normal form* if the variable to be applied has not been bound to a value. For example, $\lambda xy.y$ is a normal form, but $(\lambda xy.y)S$ gets reduced to the normal form $\lambda y.y$, and $(\lambda y.y)K$ gets reduced to the normal form K. Reducing a term to normal form is equivalent to a Turing machine reaching a halting state. Some lambda terms do not have normal forms; these correspond to computations that never finish. For example, the term $(\lambda x.xx)(\lambda x.xx)$ reduces to itself; it is the lambda-calculus equivalent of an infinite loop.

The output of a lambda calculus computation is the normal form of the term, if it exists. Since normal forms can be enumerated *à la* Gödel, there are bijections from normal forms to natural numbers and to binary strings. For the purposes of this paper, where we define machines to be partial recursive functions from binary strings to binary strings, it is convenient to choose the latter.

Lambda calculus' alphabet consists of parentheses, the symbols for the lambda operator and the dot, and symbols for variables. Though one rarely needs more than twenty-six variables, the formalism allows for subscripts; therefore, digits for subscripts and an end-of-subscript marker are also included. Let k be the number of symbols in the alphabet.

Since there infinitely many prefix-free machines that halt on each string x, lambda calculus needs to be able to encode an arbitrary string with only a constant overhead in order to be a universal prefix-free machine. When a bit string increases by one symbol, the number of representable strings increases by a factor of k. However, because of the well-formedness requirement that parentheses balance in a lambda term, the number of codewords with one more symbol increases by a smaller factor. Any encoding of strings necessarily suffers from a slight expansion, and so lambda calculus fails to reach the constant overhead bound.

There are more requirements for a well-formed lambda term that we are ignoring here. We perform an exact analysis of a less restrictive case where we have only one parenthesis symbol and one combinator in the next section; we'll see that it, too, fails to be a universal prefix-free machine.

1.2.2 Iota

Iota [3] is a minimalist language created by Chris Barker. The universal basis $\{S, K\}$ suffices to produce every lambda term, but it is not necessary. There are one-combinator bases, known as *universal combinators*. Iota is a very simple universal combinator, $\lambda f.fSK$, denoted 0. To make Iota unambiguous, there is a prefix operator, 1, for application. Valid programs are preorder traversals of full binary trees. In the tables that follow, brackets $[\cdot]$ denote taking the semantics of the argument.

Syntax	Semantics
$F \to 1F_0F_1$	$[F_0]([F_1])$
$F \to 0$	$\lambda f.fSK$

Fokker [40] proposed a different universal combinator, $\lambda f.fS(\lambda xyz.x)$, which is slightly larger, but recovers S and K with fewer applications.

Like lambda calculus, there will be a normal form of the Iota term if the program halts. As before, because normal forms are denumerable, we can make a bijection between binary strings and normal forms, ordering both lexically, and output the matching string. Another alternative, advocated by Ben Rudiak-Gould [56], is to restrict the output to a subset of normal forms, such as lists of booleans. Any

program whose normal form is not in this subset is defined to output the empty string, while the list of booleans is converted directly to a binary string.

We illustrate the execution of two simple Iota programs below:

$$
\begin{aligned}
100 &= (\lambda f.fSK)(\lambda f.fSK) \\
&= (\lambda f.fSK)SK \\
&= SSKK \\
&= SK(KK) \\
&= I
\end{aligned}
$$

$$
\begin{aligned}
1010100 &= 1010I \\
&= (\lambda f.fSK)((\lambda f.fSK)I) \\
&= (\lambda f.fSK)(ISK) \\
&= (\lambda f.fSK)(SK) \\
&= SKSK \\
&= KK(SK) \\
&= K
\end{aligned} \tag{1}
$$

Notice that at step (1) we performed an application within an internal branch. Iota is confluent: it does not matter in which order the applications are carried out, because there are no side-effects.

Iota is not quite a universal prefix-free machine because of the requirement that codewords be preorder traversals of a full binary tree. There are C_n full binary trees with $n + 1$ leaves, where C_n is the nth Catalan number, giving a codeword of length $2n + 1$; asymptotically, $C_n \sim \frac{4^n}{\sqrt{\pi n^3}}$. Thus, if we increase the length of a codeword by two bits, the number of representable strings only increases by $2 - 3\lg(\frac{n}{n-1})$ bits. It is asymptotically close to being a universal prefix-free machine, but doesn't quite make it. Again, any encoding of strings within Iota will necessarily suffer from a slight expansion, and will not satisfy the $O(1)$ error requirement.

1.2.3   Zot

Zot [4] is a continuized form of Iota. Here, 1 is a combinator rather than an operator; in Barker's words, it is treated "lexically" rather than "syncategoremically." The initial continuation is the trivial one, and the current continuation is applied to each combinator in turn. This allows the program to get access to each bit of input individually. It also makes Zot a nice Gödel numbering, since every blank-terminated binary string is a valid Zot codeword and every computable function is represented.

Syntax	Semantics
$F \to FB$	$[F]([B])$
$F \to \diamond$	$\lambda c.cI$
$B \to 0$	$\lambda c.c(\lambda f.fSK)$
$B \to 1$	$\lambda cL.L(\lambda lR.R(\lambda r.c(lr)))$
$B \to \odot$	$\lambda c.c(O)(P)$, where $O = \lambda abcde.KI$
	and P is an output monad.

Barker also includes an operator which I have denoted $\odot$, which allows the program to interact with an output monad. It is not strictly necessary: we can ignore the $\odot$ operator and, like Iota, consider the normal form of the current continuation to be the output.

Zot is a BEM, and therefore not a universal prefix-free machine.

1.2.4 Binary lambda calculus

Binary lambda calculus (BLC) [63] is a language created by John Tromp in resonse to Chaitin's claim [27] that "Lambda calculus is even simpler and more elegant than LISP, but it's unusable. Pure lambda calculus with combinators S and K, it's beautifully elegant, but you can't really run programs that way, they're too slow." Tromp noted that "There is however nothing intrinsic to λ calculus or CL that is slow; only such choices as Church numerals for arithmetic can be said to be slow, but one is free to do arithmetic in binary rather than in unary," and proposed BLC specifically for studying concrete AIT.

Rather than follow Church's original notation, Tromp used de Bruijn [34] notation, which eliminates the need to use the variable name in the both lambda prefix and in the body of a term. Instead, n refers to the variable bound by the nth enclosing λ.

Syntax	Semantics
$F \rightarrow 01F_0F_1$	$[F_0]([F_1])$
$F \rightarrow 00F$	$\lambda[F]$
$F \rightarrow 1^{n+1}0$	n

Any remaining bits are converted to a *nil*-terminated list of combinators K and KI to which the program is applied. The list is constructed using the pairing cobinator $P = \lambda xyz.zxy$, which has the property that $PXYK = X$ and $PXY(KI) = Y$. Thus K and KI behave like booleans with respect to P.

Tromp explicitly states that the normal form of the resulting BLC term is the output.

Prefix-free BLC uses a rather nonstandard approach. Instead of defining a computer whose domain is prefix-free, Tromp redefines the way output is handled: a program is prefix free if and only if

$$U(p:z) = \langle x_p, z \rangle, \tag{*}$$

where $p:z$ is a lambda term encoding a list whose first few members are the bit string p, followed by the tail z, where z may be infinite. Since z is potentially infinite and is arbitrary, U cannot output z without processing all of the bits of p and returning z as part of the output. This guarantees that no prefix or extension of p has the right form, and thus the set of such p is prefix-free.

Normal BLC is a BEM, and therefore not a universal prefix-free machine; prefix-free BLC is a BEM combined with the definition (*), but does not define a prefix-free machine whose domain matches the set $\{p \mid (*) \text{ holds}\}$.

We would like to construct universal prefix-free machines from universal BEMs. The first step is to get rid of the blank endmarker. In the next section, we describe an algorithm to extract the (possibly empty) subset of the domain of a BEM that does not depend on reading the blank endmarker to know when to halt.

1.3   Eliminating the blank endmarker

Some BEMs have the property that their behavior after the final read request does not depend on the result of that request. In a BEM B, the last request is always for the $\diamond$ symbol, but in these cases, the program does not need the $\diamond$ to know when to halt. We can use this property to define a prefix-free machine whose domain is a prefix-free set of programs such that if $x \in \mathrm{dom}(C)$, then $x\diamond \in \mathrm{dom}(B)$.

We construct C in the following way:

1. C simulates B up to the point where the first read request is made; if no read requests are made, then C loops forever. After a read request, no read is actually performed; rather,

2. C simulates the behavior of B for all the possibilites for that symbol up to the point where one of the branches is about to halt or make another read request.

3. At that point, the read for the previous symbol is actually performed; if there are no more symbols, then C blocks; otherwise the C selects the appropriate branch and the abandons other branches' simulations.

4. C continues executing that branch until it halts or makes a read request. If the selected branch halts, then C halts (although if there are symbols remaining in the pipe, the sender will block and the computation will fail to halt). If the selected branch performs a read request, C goes to step (2).

In this way, C only needs to simulate $|\Sigma|$ concurrent branches at a time, and if B does not need to read $\diamond$ to know when to halt, then that read is never actually performed by C. The domain of C is prefix free, since C would underflow on any string more than one symbol shorter than a codeword of B and overflow on any extension of a codeword of C.

A prefix-free machine constructed in this way is universal if it can simulate either of the universal machines that Chaitin proposed with only an additive constant increase in the size of the input. The machine constructed in this way from BLC is universal.

1.4   Church's lambda operator as a curried interpreter

In this section, we take a small aside to introduce a concept used in one of the proposed prefix-free machines below, as well as to introduce a bit of new notation.

Church's lambda operator can be seen as an interpreter. It reads three self-delimiting parameters from the input stream—*var*, *body*, and *replacement*—builds data structures representing the application of the functions and operators that those parameters describe, and performs alpha and beta reduction on the data structures, calling itself recursively. If the process of alpha- and beta-reduction reaches a point where it cannot continue, it decodes the data structure into the normal form of the lambda term and outputs it as a string of symbols.

Since an interpreter is merely a function, we can curry it: the function λ takes an input *var* and returns a new function λ'; likewise, the function λ' takes a single input *body* and returns a function λ''; the function λ'' applied to the input *replacement* yeilds the normal form, if one exists.

Since curried functions take exactly one input, the application operator becomes strictly binary, and the tree of applications is a full binary tree. Programs may be written as preorder traversals of the tree to avoid parentheses. We adopt the backtick (') as a prefix application operator throughout the rest of the paper, which behaves identically to Iota's 1 operator; for example, the lambda term $S = \lambda xyz.xz(yz)$ will be written

$$\text{``}\lambda x \text{ ``}\lambda y \text{ ``}\lambda z \text{ ``}x\ z\ \text{`}y\ z$$

In Appendix A, we include the Javascript source code for a source-to-source filter from this dialect of lambda calculus to Javascript. It supports lazy evaluation and can be trivially modified to work with any eager language with first-class functions, such as Perl.

1.5   A very simple prefix-free machine

Below, we present a Chaitin-universal combinator for use in Iota. Input requests R go via a monad that binds the requests together in lazy-evaluation order. R reads a bit and evaluates to K or '$K\ I$ if the bit is 0 or 1, respectively.

The definition of the combinator is optimized, like Fokker's, for the number of applications to recover K, S, and R. Codewords are concatenations of programs (preorder traversals of the application tree) and (possibly empty) input.

$$
\begin{aligned}
A &= \text{`}K\ \text{`}K\ R \\
B &= \text{`}K\ \text{`}K\ \text{`}K\ \text{`}K\ \text{`}K\ \text{`}K\ \text{`}K\ K \\
C &= \text{``}\lambda\ x\ \text{`}x\ B \\
0 &= \text{``}\lambda\ x\ \text{````}x\ C\ A\ \text{`}K\ I\ S
\end{aligned}
$$

$$
\begin{aligned}
100 &= K \\
10100 &= S \\
1010100 &= R
\end{aligned}
$$

1	first program to loop due to a syntax error
00 = 0 with input 0	first to loop due to overflow
110101000 ='R 0 with no input	first to loop due to underflow
1101010000 = 'R 0 with input 0 = 'K 0	first to halt with nonempty input

This language is prefix-free. Prefices of programs are not traversals of full binary trees, so they loop due to a syntax error. In any halting program, there will be a finite number of applications of the R operator, and thus a finite number of bits appended to the end of the program. The execution of any prefix of that codeword will block due to underflow, and any extension will block due to overflow.

It is also universal with respect to all prefix-free machines defined over $\{0,1\}$. Chaitin's universal machine can be simulated by this one with a program that reads in a parenthesis-balanced LISP S-expression and evaluates it: S and K are universal over the lambda terms, and R provides the bits for Chaitin's *readBit* function.

1.6   Extending a universal combinator

One may also construct a Chaitin-universal combinator 0 from any universal combinator U by taking

$$0 = \text{``}Pair \text{ ``}\lambda x \text{ ``}\lambda y \text{ ``}\lambda z\ U\ R,$$

where

$$Pair \quad = ``\lambda x ``\lambda y ``\lambda z ``z \; x \; y.$$

Programs written for U can be converted to use 0 by replacing U with 100. For example, taking Iota's combinator $U =``\lambda f ``f \; S \; K$, we have that

$$
\begin{aligned}
1100100 \quad &= `U \; U & &= I \\
11001100100 \quad &= `U \; `U \; U & &= `S \; K \\
110011001100100 \quad &= `U \; `U \; `U \; U & &= K \\
1100110011001100100 \quad &= `U \; `U \; `U \; `U \; U & &= S \\
1011001100100 \quad &= `0 \; `U \; `U \; U & &= R
\end{aligned}
$$

1.7 Keraia, continuized binary lambda calculus with a 6-bit UTM

Keraia is a BEM that uses a straightforward encoding of the curried λ introduced in section 1.4. We begin with three examples:

I =	◇		110	0	0								
	Interpret		``λ	x	x								
K =	◇		110	0	110	10100	0						
	Interpret		``λ	x	``λ	y	x						
S =	◇		110	10100	110	11000	110	0	11	10100	0	1	11000	0
	Interpret	``λ	x	``λ	y	``λ	z	``	x	z	`	y	z

The leftmost leaf represents the curried λ function; the first right subtree *var* (if it exists) represents a variable; the second right subtree *body* (if it and *var* exist) represents the applications of curried lambda and variables; the third right subtree *replacement* (if it and the previous two exist) represents the replacement pattern.

Keraia uses a greedy algorithm while marking variables: it traverses *body* marking occurrences of *var*, then recursively parses *body* to mark the rest of the variables. Next, it performs α-reduction and β-reduction until *body* has reached normal form. Any remaining leaves are replaced by the 0 combinator. Finally, *Keraia* performs lambda-abstraction and returns a combinator.

Syntax	Semantics
$F \to FB$	$`[F] \; [B]$
$F \to \diamond$	$[0]$
$B \to 0$	$``\lambda c \; `c \; Interpret$
$B \to 1$	$``\lambda c ``\lambda A \; `A ``\lambda a ``\lambda B \; `B ``\lambda b \; `c ``Pair \; a \; b$

While Zot's 1 combinator applies the left branch to the right, Keraia's 1 merely *Pair*s the branches. *Interpret* reads in the data structure created by the *Pair*ing, and then interprets it. The behavior of the function *Interpret* is only specified on combinators of the form constructed by applications of the combinators 0 and 1.

Like Zot, Keraia has a very simple self-interpreting UTM, with the same meaning: 111000 is the encoding of "apply the identity operator to what follows."

In Appendix B, we give Javascript source code for an implementation of Keraia that bootstraps off of the curried lambda dialect from section 1.4.

1.8   Keraia as a universal prefix-free machine

Rather than use a continuized set of combinators to get a universal Turing machine, we can get a universal prefix-free machine with a few small modifications. First, we treat 1 lexically, interpreting the first full binary tree traversal as the program description; the remaining bits are given to the sender to push through the pipe. Also, rather than replace remaining leaves with the 0 combinator in the last step, we replace them with R operator. As always, syntax errors (incomplete tree traversals), overflow, and underflow cause the machine to loop indefinitely.

For example, the codeword 1110100101001100001 splits into a self-delimiting program (all the bits but the last) and the input bit 1 (the last bit). The execution proceeds as follows:

$$
\begin{aligned}
11101001010011000 \quad &= \text{'''}\lambda x \text{ 'R } x \text{ } I \\
&= \text{'R } I \\
&= \text{''}K \text{ } I \text{ } I \qquad (R \rightarrow \text{'K } I \text{ upon reading the bit 1}) \\
&= I
\end{aligned}
$$

This modification of Keraia is a universal prefix-free machine, since every Lisp S-expression has an equivalent lambda term that is directly encodable, and the R operator behaves identically to Chaitin's *readBit* operator.

1.9   Conclusion

Algorithmic information theory has much to say both about physics and philosophy. It would be nice to experiment with tiny concrete models, but until now, there were only two programming languages that were universal prefix-free machines. We have given examples of two new universal prefix-free machines, a modification to universal combinators that allows them to be Chaitin-universal, and an algorithm for constructing a prefix-free machine from a BEM by removing the blank-endmarker and extracting the prefix-free subset of those words.

The complexity of n bits of a Chaitin Omega number is $n - c$; Calude *et al.* [13] computed the first 64 bits of an Omega number, the halting probability of Chaitin's register machine. We know, now, that c for that machine is at least 64. The machines proposed above are ideally suited for similar computations.

Chaitin published an exponential Diophantine equation with one parameter n which has infinitely many solutions if and only if the nth bit of Ω_C (i.e. the halting probability of a particular universal prefix-free machine C) is one [24]. These machines should make it possible to producing a smaller instance.

2

From Heisenberg to Gödel via Chaitin

2.1 Introduction

Are there any connections between uncertainty and incompleteness? We don't know of any reaction of Heisenberg to this question. However, Gödel's hostility to any suggestion regarding possible connections between his incompleteness theorem and physics, particularly, Heisenberg's uncertainty relation, is well-known.* One of the obstacles in establishing such a connection comes from the different nature of these two results: uncertainty is a quantitative phenomenon while incompleteness is prevalently qualitative.

In recent years there have been a lot of interest in the relations between computability and incompleteness and physics. Opinions vary considerably, from the conclusion that the impact on Gödel and Turing incompleteness theorems to physics is a red herring (see [17, 18]), to Hawking's view that "a physical theory is self-referencing, like in Gödel's theorem. ... Theories we have so far are both inconsistent and incomplete" (cf. [43]). A very interesting analysis of the possible impact of Gödel's incompleteness theorems in physics was written by Barrow [5, 6]; the prevalence of physics over mathematics is argued by Deutsch [36]; for Svozil [61, 62], Heisenberg's incompleteness is pre-Gödelian-Turing and finite. Other relevant papers are Geroch and Hartle [41], Peres [54], and Peres and Zurek [55].

In this note we do *not* ask whether Gödel's incompleteness has any bearing on Heisenberg's uncertainty, but the converse: Does uncertainty imply incompleteness? We will show that we can get a positive answer to this question: algorithmic randomness can be recast as a "formal uncertainty principle" which implies Chaitin's information-theoretic version of Gödel's incompleteness.

2.2 Outline

We begin with overviews of the relevant ideas first discovered by Heisenberg, Gödel, and Chaitin.

---

*J. Wheeler was thrown out of Gödel's office for asking the question "Professor Gödel, what connection do you see between your incompleteness theorem and Heisenberg's uncertainty principle?", cf. Chaitin's account cited in Barrow [5], p. 221.

Next, we show that random reals, of which Chaitin Omega numbers are just an example, satisfy a "formal uncertainty principle", namely

$$\Delta_s \cdot \Delta_C(\omega_1 \ldots \omega_s) \geq \varepsilon, \tag{2.1}$$

where ε is a fixed positive constant.

The two conjugate coordinates are the random real and the binary numbers describing the programs that generate its prefixes. Then, the uncertainty in the random real given an n-bit prefix is 2^{-n}, and the uncertainty in the size of the shortest program that generates it is, to within a multiplicative constant, 2^n.

The Fourier transform is a lossless transformation, so all the information contained in the delta function $\delta_{\Omega(x)} = 1$ if $x = \Omega$, $\delta_{\Omega(x)} = 0$, otherwise, is preserved in the conjugate. Therefore, if you need n bits of information to describe a square wave convergent on the delta function, there must be n bits of information in the Fourier transform of the square wave. Since both the information in the transformed square wave and the shortest program describing the square wave increase linearly with n, there is an equivalence between the two.

We show that the formal uncertainty principle is a true uncertainty principle–that is, the terms are bounded by the standard deviations of two random variables with particular probability distributions. We note that for many self-delimiting Turing machines C, the halting probability Ω_C is computable; in these cases, there are quantum systems with observables described by these probability distributions, and our uncertainty relation is equivalent to Heisenberg's.

Finally, (2.1) implies a strong version of Gödel's incompleteness, Chaitin's information-theoretic version [20, 21] (see also the analysis in [35, 8]). Chaitin's proof relied on measure theory; we present here a new proof via a complexity-theoretic argument.

2.3   Heisenberg

In 1925 Heisenberg developed the theory of matrix mechanics; it was his opinion that only observable quantities should play any role in a theory. At the time, all observations came in the form of spectral absorption and emission lines. Heisenberg, therefore, considered the "transition quantities" governing the jumps between energy states to be the fundamental concepts of his theory. Together with Born, who realized Heisenberg's transition rules obeyed the rules of matrix calculus, he developed his ideas into a theory that predicted nearly all the experimental evidence available.

The next year, Schrödinger introduced what became known as wave mechanics, together with a proof that the two theories were equivalent. Schrödinger argued that his version of quantum mechanics was better in that one could visualize the behavior of the electrons in the atom. Many other physicists agreed with him.

Schrödinger's approach disgusted Heisenberg; in a letter to Pauli (see [53]), he called Schrödinger's interpretation "crap". Publicly, however, he was more restrained. In [44] he argued that while matrix mechanics was hard to visualize, Schrödinger's interpretation of wave mechanics was self-contradictory, and concluded that something was still missing from the interpretation of quantum theory.

In 1927 Heisenberg published "Über den Anschaulichen Inhalt der Quantentheoretischen Kinematik und Mechanik" (see [45]) to provide the missing piece. First, he gave his own definition of visualization: "We believe we have gained intuitive understanding of a physical theory, if in all simple cases, we can grasp the experimental consequences qualitatively and see that the theory does not lead to any contradictions." In this sense, matrix mechanics was just as intuitive as wave mechanics. Next, he argued that terms like

"the position of a particle" can only make sense in terms of the experiment that measures them.

To illustrate, he considered the measurement of an electron by a microscope.[†] The accuracy is limited by the wavelength of the light illuminating the electron; one can use as short a wavelength as one wishes, but for very short wavelengths, the Compton effect is non-negligible. He wrote, (see [45], p.174–175),

> At the instant of time when the position is determined, that is, at the instant when the photon is scattered by the electron, the electron undergoes a discontinuous change in momentum. This change is the greater the smaller the wavelength of the light employed, i.e., the more exact the determination of the position. At the instant at which the position of the electron is known, its momentum therefore can be known only up to magnitudes which correspond to that discontinuous change; thus, the more precisely the position is determined, the less precisely the momentum is known, and conversely.

Heisenberg estimated the uncertainty to be on the order

$$\delta_p \cdot \delta_q \sim h,$$

where h is Planck's constant.

Kennard (see [47]) was the first to publish the uncertainty relation in its exact form. He proved in 1927 that for all normalized state vectors $|\Psi\rangle$,

$$\Delta_p \cdot \Delta_q \geq \hbar/2,$$

where Δ_p and Δ_q are standard deviations of momentum and position, i.e.

$$\Delta_p^2 = \langle\Psi|p^2|\Psi\rangle - \langle\Psi|p|\Psi\rangle^2; \Delta_q^2 = \langle\Psi|q^2|\Psi\rangle - \langle\Psi|q|\Psi\rangle^2.$$

Thus, assuming quantum mechanics is an accurate description of reality, the formalism is compatible with Heisenberg's principle.

2.4   Gödel

In 1931 Gödel published his (first) incompleteness theorem in [42] (see also [38, 39]). According to the current terminology, he showed that *every formal system which is (1) finitely specified, (2) rich enough to include the arithmetic, and (3) consistent, is incomplete.* That is, there exists an arithmetical statement which (A) can be expressed in the formal system, (B) is true, but (C) is unprovable within the formal system.

All conditions are necessary. Condition (1) says that there is an algorithm listing all axioms and inference rules (which could be infinite). Taking as axioms all true arithmetical statements will not do, as this set is not finitely listable. A "true arithmetical statement" is a statement about non-negative integers which cannot be invalidated by finding any combination of non-negative integers that contradicts it. Condition (2) says that the formal systems has all the symbols and axioms used in arithmetic, the symbols for 0 (zero), S (successor), $+$ (plus), $\times$ (times), $=$ (equality) and the axioms making them work (as for example, $x + S(y) = S(x + y)$). Condition (2) cannot be satisfied if you do not have individual terms for $0, 1, 2, \ldots$; for example, Tarski [67] proved that the plane Euclidean geometry, which refers to

---

[†]Heisenberg might have been so concerned with uncertainty because in 1923 he almost failed his Ph.D. exam when Sommerfeld asked about (optical) limitations to the resolution of the microscope.

points, circles and lines, is complete.[‡] Finally (3) means that the formal system is free of contradictions.

Like uncertainty, incompleteness has provoked a lot of interest (and abuse).

2.5   Chaitin

Chaitin has obtained three types of information-theoretic incompleteness results (scattered through different publications, [20, 21, 23, 26]; see also [29, 30]). The strongest form concerns the computation of the bits of a Chaitin Omega number Ω_U, the halting probability of a self-delimiting universal Turing machine U (see also the analysis in [35, 8]). A self-delimiting Turing machine U is a normal Turing machine C which processes binary strings into binary strings and has a prefix-free domain, that is, if $C(x)$ is defined and y is either a proper prefix or an extension of x, then $C(y)$ is not defined. The self-delimiting Turing machine U is universal if for every self-delimiting Turing machine C there exists a fixed binary string p (the simulator) such that for every input x, $U(px) = C(x)$: either both computations $U(px)$ and $C(x)$ stop and, in this case they produce the same output or both computations never stop. The Omega number introduced in [20]

$$\Omega_U = 0.\omega_1\omega_2\ldots\omega_n\ldots \tag{2.2}$$

is the halting probability of U; it is one of the most important concepts in algorithmic information theory (see [8]).

In [20] Chaitin proved the following result: *Assume that X is a formal system satisfying conditions (1), (2) and (3) in Gödel's incompleteness theorem. Then, for every self-delimiting universal Turing machine U, X and values of only finitely scattered bits of Ω_U, and one can give a bound on the number of bits of Ω_U which X determine.* This is a form of incompleteness because, with the exception of finitely many n, any true statement of the form "the nth bit of Ω_U is ω_n" is unprovable in X.

For example, we can take X to be ZFC[§] under the assumption that it is arithmetically sound, that is, any theorem of arithmetic proved by ZFC is *true*. Solovay [60] *has constructed a specific self-delimiting universal Turing machine S (called Solovay machine) such that ZFC cannot determine any bit of Ω_S.* In this way one can obtain *constructive* versions of Chaitin's theorem. For example, *if ZFC is arithmetically sound and S is a Solovay machine, then the statement "the 0th bit of the binary expansion of Ω_S is 0" is true but unprovable in ZFC.* In fact, one can effectively construct arbitrarily many examples of true and unprovable statements of the above form, cf. [10].

2.6   Rudiments of Algorithmic Information Theory

In this section we will present some basic facts of algorithmic information theory in a slightly different form which is suitable for the results appearing in the following section.

We will work with binary strings; the length of the string x is denoted by $|x|$. For every $n \geq 0$ we denote by $B(n)$ the binary representation of the number $n + 1$ without the leading 1. For example, $0 \mapsto \lambda$ (the empty string), $1 \mapsto 0$, $2 \mapsto 1$, $3 \mapsto 00, \ldots$ The length of $B(n)$ is almost equal to $\log_2(n)$; more precisely, it is $\lfloor \log_2(n+1) \rfloor$. The function B is bijective and we denote by N its inverse. The string x is length-lexicographically less than the string y if and only if $N(x) < N(y)$.

We need first the Kraft-Chaitin theorem: *Let $n_1, n_2, \ldots$ be a computable sequence of non-negative*

[‡]This result combined with with Gödel's completeness theorem implies decidability: there is an algorithm which accepts as input an arbitrary statement of plane Euclidean geometry, and outputs "true" if the statement is true, and "false" if it is false. The contrast between the completeness of plane Euclidean geometry and the incompleteness of arithmetic is striking.

[§]Zermelo-Fraenkel set theory with choice.

integers such that

$$\sum_{i=1}^{\infty} 2^{-n_i} \leq 1. \tag{2.3}$$

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

Let C be a self-delimiting Turing machine. The program-size complexity induced by C is defined by $H_C(x) = \min\{|w| \mid C(w) = x\}$ (with the convention that strings not produced by C have infinite complexity). One might suppose that the complexity of a string would vary greatly between choices of self-delimiting Turing machine. However, because of the universality requirement, the complexity difference between C and C' is at most the length of the shortest program for C' that simulates C. Therefore, the complexity of a string is fixed to within an additive constant. This is known as the "invariance theorem" (see [8]), and is usually stated: *For every self-delimiting universal Turing machine U and self-delimiting Turing machine C there exists a constant $\varepsilon > 0$ (which depends upon U and C) such that for every string x,*

$$H_U(x) \leq \varepsilon + H_C(x).$$

For our aim it is more convenient to define the complexity measure $\nabla_C(x) = \min\{N(w) \mid C(w) = x\}$, the smallest integer whose binary representation produces x via C. Clearly, for every string x,

$$2^{H_C(x)} - 1 \leq \nabla_C(x) \leq 2^{H_C(x)+1} - 2.$$

Therefore we can say that $\Delta_C(x)$, our uncertainty in the value $\nabla_C(x)$, is the difference between the upper and lower bounds given, namely $\Delta_C(x) = 2^{H_C(x)}$.

The invariance theorem can now be stated as follows: *for every self-delimiting universal Turing machine U and self-delimiting Turing machine C there exists a constant $\varepsilon > 0$ (which depends upon U and C) such that for every string x,*

$$\Delta_U(x) \leq \varepsilon \cdot \Delta_C(x).$$

Let $\Delta_s = 2^{-s}$. Chaitin's theorem (see [20]) stating that the bits of Ω_U in (2.2) form a random sequence can now be presented as a "formal uncertainty principle": *for every self-delimiting Turing machine C there is a constant $\varepsilon > 0$ (which depends upon U and C) such that*

$$\Delta_s \cdot \Delta_C(\omega_1 \ldots \omega_s) \geq \varepsilon. \tag{2.4}$$

The inequality (2.4) is an uncertainty relation as it reflects a limit to which we can simultaneously increase both the accuracy with which we can approximate Ω_U and the complexity of the initial sequence of bits we compute; it relates the uncertainty of the output to the size of the input. When s grows indefinitely, Δ_s tends to zero in contrast with $\Delta_C(\omega_1 \ldots \omega_s)$ which tends to infinity; their product is not only bounded from below, but increases indefinitely (see also (2.6)). From a complexity viewpoint (2.4) tells us that there is a limit ε up to which we can uniformly compress the initial prefixes of the binary expansion of Ω_U.

How large can be ε in (2.4)? For example, $\varepsilon = 1$ when $C = U_0$ is a special universal self-delimiting Turing machine:

$$\Delta_s \cdot \Delta_{U_0}(\omega_1 \ldots \omega_s) \geq 1. \tag{2.5}$$

If U is universal and satisfies (2.4), then a universal machine U_0 satisfying (2.5) can be defined by $U_0(0^\varepsilon x) = U(x)$ (so requiring that any input to U_0 not starting with ε zeros causes the machine to go into an infinite loop).

In fact, in view of the strong complexity-theoretic characterization of random sequences (see [20, 8]) a stronger form of (2.4) is true: *for every positive integer N there is a bound M (which depends upon U, C and N) such that for all $s \geq M$ we have:*

$$\Delta_s \cdot \Delta_C(\omega_1 \ldots \omega_s) \geq N. \tag{2.6}$$

The constant N appearing in (2.4) can be made arbitrarily large in case s is large enough; the price paid appears in the possible violation of the inequality for the first $s < M$ bits.

Is (2.4) a 'true' uncertainty relation? We prove that the variables Δ_s and Δ_C in (2.4) are the standard deviations of two measurable observables in suitable probability spaces.

For Δ_s we consider the space of all real numbers in the unit interval which are approximated to exactly s digits. Consider the probability distribution $Prob(v) = P_C(v)/\Omega_C^s$, where $P_C(x) = \sum_{C(y)=x} 2^{-|y|}$ and $\Omega_C^s = \sum_{|x|=s} P_C(x)$.

Now fix the first s digits of Ω_U, $\omega_1 \omega_2 \ldots \omega_s$ and define

$$\alpha = 2^{-s/2} \cdot (Prob(\omega_1 \omega_2 \ldots \omega_s))^{-1/2} \cdot (1 - Prob(\omega_1 \omega_2 \ldots \omega_s))^{-1/2}.$$

The random variable X on a reals approximated by the first s digits $v = v_1 v_2 \ldots v_s$ is defined by the delta function $X(v) = \alpha$ if $v = \omega_1 \omega_2 \ldots \omega_s$ and $X(v) = 0$ otherwise. Then the expectation values of X and X^2 are $\langle X \rangle = \alpha \cdot Prob(\omega_1 \omega_2 \ldots \omega_s)$ and $\langle X^2 \rangle = \alpha^2 \cdot Prob(\omega_1 \omega_2 \ldots \omega_s)$, so the standard deviation is $\sigma_X = \Delta_s$.

For Δ_C we consider

$$\beta = (\Delta_C(\omega_1 \omega_2 \ldots \omega_s))^{1/2} \cdot (Prob(\omega_1 \omega_2 \ldots \omega_s))^{-1/2} \cdot (1 - Prob(\omega_1 \omega_2 \ldots \omega_s))^{-1/2},$$

and the same space but the random variable $Y(\omega_1 \omega_2 \ldots \omega_s) = \beta$ and $Y(v) = 0$ if $v \neq \omega_1 \omega_2 \ldots \omega_s$. Then, the expectation values of Y and Y^2 are $\langle Y \rangle = \beta \cdot Prob(\omega_1 \omega_2 \ldots \omega_s)$ and $\langle Y^2 \rangle = \beta^2 \cdot Prob(\omega_1 \omega_2 \ldots \omega_s)$, so the standard deviation is $\sigma_Y = \Delta_C(\omega_1 \omega_2 \ldots \omega_s)$.

Hence the relation (2.4) becomes:

$$\sigma_X \cdot \sigma_Y = \Delta_s \cdot \Delta_C(\omega_1 \omega_2 \ldots \omega_s) \geq \varepsilon,$$

so for U_0 satisfying (2.5) we have:

$$\sigma_X \cdot \sigma_Y \geq 1.$$

2.7   From Heisenberg to Chaitin

Since self-delimiting universal Turing machines are strictly more powerful than non-universal ones, the inequality holds for the weaker computers as well. In many of these cases, the halting probability of the machine is computable, and we can construct a quantum algorithm to produce a set of qubits whose state is described by the distribution.

To illustrate, we consider a quantum algorithm with two parameters, C and s, where C is a Turing machine for which the probability of producing each s-bit string is computable. We run the algorithm to compute that distribution on a quantum computer with s ouput qubits; it puts the output register into a superposition of spin states, where the probability of each state $|v\rangle$ is $P_C(v)/\Omega_C^s$. Next, we apply the Hamiltonian operator $H = \beta |\omega_1 \ldots \omega_s\rangle \langle \omega_1 \ldots \omega_s|$ to the prepared state. A measurement of energy will give β with probability $P = Prob(\omega_1 \omega_2 \ldots \omega_s)$ and zero with probability $1 - P$. The expectation value

for energy, therefore, is exactly the same as that of Y, but with units of energy, i.e.

$$\Delta_C(\omega_1\omega_2\ldots\omega_s)[J] \cdot \Delta_s \geq \varepsilon[J],$$

where $[J]$ indicates Joules of energy.

Now define

$$\Delta_t \equiv \frac{\sigma_Q}{|d\langle Q\rangle/dt|},$$

where Q is any observable that does not commute with the Hamiltonian; that is, Δ_t is the time it takes for the expectation value of Q to change by one standard deviation. With this definition, the following is a form of Heisenberg's uncertainty principle:

$$\Delta_E \cdot \Delta_t \geq \hbar/2.$$

We can replace Δ_E by $\Delta_C(\omega_1\omega_2\ldots\omega_s)$ by the analysis above; but what about Δ_t? If we choose a time scale such that our two uncertainty relations are equivalent for a single quantum system corresponding to a computer C and *one* value of s, then the relation holds for C and *any* value of s:

$$\Delta_C(\omega_1\omega_2\ldots\omega_s)[J] \cdot \Delta_s \frac{\hbar}{2\varepsilon}[J^{-1} \cdot Js] \geq \frac{\hbar}{2}[Js].$$

In this sense, we claim that Heisenberg's uncertainty relation is equivalent to (2.4). We cannot say whether (2.4) is physical for universal self-delimiting Turing machines; to do so requires deciding the Church-Turing thesis for quantum systems.

The uncertainty principle now says that getting one more bit of Ω_U requires (asymptotically) twice as much energy. Note, however, that we have made an arbitrary choice to identify energy with complexity. We could have chosen to create a system in which the position of a particle corresponded to the complexity, while momentum corresponded to the accuracy of C's estimate of Ω_U. In that case, the uncertainty in the position would double for each extra bit. Any observable can play either role, with a suitable choice of units.

If this were the only physical connection, one could argue that the result is merely an analogy and nothing more. However, consider the following: let ρ be the density matrix of a quantum state. Let R be a computable positive operator-valued measure, defined on a finite dimensional quantum system, whose elements are each labeled by a finite binary string. Then the statistics of outcomes in the quantum measurement is described by R: $R(\omega_1\ldots\omega_s)$ is the measurement outcome and $tr(\rho R(\omega_1\ldots\omega_s))$ is the probability of getting that outcome when we measure ρ. Under these hypotheses, Tadaki's inequality (1) (see [64], p. 2), and our inequality (2.4) imply the existence of a constant τ (depending upon R) such that for all ρ and s we have:

$$\Delta_s \cdot \frac{1}{tr(\rho R(\omega_1\ldots\omega_s))} \geq \tau.$$

In other words, there is no algorithm that, for all s, can produce

1. an experimental setup to produce a quantum state and

2. a POVM with which to measure the state such that

3. the probability of getting the result $\omega_1\omega_2\ldots\omega_s$ is greater than $1/(\tau 2^s)$.

Finally, it is interesting to note that a Fourier transform of the wave function switches between an "Omega space" and a "complexity space". We plan on examining this relationship further in a future paper.

2.8   From Chaitin to Gödel

In this section we prove that the uncertainty relation (2.4) implies incompleteness.

We start with the following theorem: *Fix a universal self-delimiting Turing machine U. Let $x_1 x_2 \ldots$ be a binary infinite sequence and let F be a strictly increasing function mapping positive integers into positive integers. If the set $\{(F(i), x_{F(i)}) \mid i \geq 1\}$ is computable, then there exists a constant $\varepsilon > 0$ (which depends upon U and the characteristic function of the above set) such that for all $k \geq 1$ we have:*

$$\Delta_U(x_1 x_2 \ldots x_{F(k)}) \leq \varepsilon \cdot 2^{F(k)-k}. \tag{2.7}$$

To prove (2.7) we consider for every $k \geq 1$ the strings

$$w_1 x_{F(1)} w_2 x_{F(2)} \ldots w_k x_{F(k)}, \tag{2.8}$$

where each w_j is a string of length $F(j) - F(j-1) - 1, F(0) = 0$, that is, all binary strings of length $F(k)$ where we have fixed bits at the positions $F(1), \ldots, F(k)$.

It is clear that $\sum_{i=1}^{k} |w_i| = F(k) - k$ and the mapping $(w_1, w_2, \ldots, w_k) \mapsto w_1 w_2 \ldots w_k$ is bijective, hence to generate all strings of the form (2.8) we only need to generate all strings of length $F(k) - k$.

Next we consider the enumeration of all strings of the form (2.8) for $k = 1, 2, \ldots$. The lengths of these strings will form the sequence

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

which is computable and satisfies the inequality (2.3) as

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

Hence, by Kraft-Chaitin theorem, for every string w of length $F(k) - k$ there effectively exists a string z_w having the same length as w such that the set $\{z_w \mid |w| = F(k) - k, k \geq 1\}$ is prefix-free. Indeed, from a string w of length $F(k) - k$ we get a unique decomposition $w = w_1 \ldots w_k$, and z_w as above, so we can define $C(z_w) = w_1 x_{F(1)} w_2 x_{F(2)} \ldots w_k x_{F(k)}$; C is a self-delimiting Turing machine. Clearly,

$$\Delta_C(w_1 x_{F(1)} w_2 x_{F(2)} \ldots w_k x_{F(k)}) \leq \nabla_C(w_1 x_{F(1)} w_2 x_{F(2)} \ldots w_k x_{F(k)}) \leq N(z_w) \leq 2^{F(k)-k+1} - 1,$$

for all $k \geq 1$. In particular, $\Delta_C(x_1 \ldots x_{F(k)}) \leq 2^{F(k)-k+1} - 1$, so by the invariance theorem we get the inequality (2.7).

It is easy to see that under the hypothesis of the above theorem the uncertainty relation (2.4) is violated, so the sequence $x_1 x_2 \ldots x_n \ldots$ is not random. Indeed, if the sequence were random, then the formal uncertainty principle (2.4) will hold true, hence for each $k \geq 1$, we would have the following contradictory pair of inequalities:

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

We are now able to deduce Chaitin's information-theoretic incompleteness theorem from the uncertainty relation (2.4). Assume by absurdity that ZFC can determine infinitely many digits of $\Omega_U = 0.\omega_1 \omega_2 \ldots$. Then, we could enumerate an infinite sequence of digits of Ω_U, thus contradicting the above

theorem.

In particular, there exists a bound N such that ZFC cannot determine more than N scattered digits of $\Omega_U = 0.\omega_1 \omega_2 \ldots$.

2.9   Conclusion

We have shown that uncertainty implies algorithmic randomness which, in turn, implies incompleteness. Specifically, the complexity-theoretic characterization of the randomness of the halting probability of a universal self-delimiting Turing machine U, Chaitin Omega number Ω_U, can be recast as a "formal uncertainty principle": an uncertainty relation between the accuracy of one's estimate of Ω_U and the complexity of the initial bit string. This relation implies Chaitin's information-theoretic version of Gödel's incompleteness.

The uncertainty relation applies to all self-delimiting Turing machines C. For the class of machines whose halting probabilities Ω_C are computable, we have shown that one can construct a quantum computer for which the uncertainty relation describes conjugate observables. Therefore, in these particular instances, the uncertainty relation is equivalent to Heisenberg's.

There is an important distinction between "quantum randomness" and our formal uncertainty principle. They are separate concepts. In the Copenhagen interpretation, the random collapse of the wavefunction is a postulate. In the Bohmian interpretation, where there are real particles with real (though non-Newtonian) trajectories, randomness comes from our ignorance about the system; the velocity of any particle depends instantaneously on every other particle. In one case the interpretation is probabilistic, while in the other, it is completely deterministic. We cannot distinguish between these. Our result concerns a different source of randomness.

Like Heisenberg's uncertainty principle, our formal uncertainty principle is a general one; they both apply to *all* systems governed by the wave equation, not just quantum waves. We could, for example, use sound waves instead of a quantum system by playing two pure tones with frequencies f and $f + \Delta_C(\omega_1 \ldots \omega_s)$. Then Δ_s corresponds to the complementary observable, the length of time needed to perceive a beat. The (algorithmic) randomness we are concerned with seems to be pervasive in physics, even at the classical level. We may speculate that uncertainty implies randomness not only in mathematics, but also in physics.

3

Omega and zeta functions

3.1   Introduction

Chaitin [20] defined the halting probability of a universal prefix-free machine U defined on $\{0,1\}^*$ to be

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

This real is computably enumerable and the bits of Ω_U form a random sequence with respect to the prefix-free complexity H:

$$\exists c \,\forall m > 1 \; H(\Omega_U[m]) \geq m - c,$$

where $\Omega_U[m]$ is the string formed by the first m bits of Ω_U.

We show below that Ω can be written as an infinite *product* over halting programs; that there exists a "natural," or base-free formulation that does not (directly) depend on the alphabet of the universal prefix-free machine; that Tadaki's generalized halting probability is well-defined even for arbitrary univeral Turing machines and the plain complexity; and finally, that the natural generalized halting probability can be written as an infinite product over primes and has the form of a zeta function whose zeros encode halting information. We conclude with some speculation about physical systems in which partial randomness could arise, and identify many open problems.

3.2   Omega as an infinite product

Given a universal prefix-free machine U, construct a program t that is known not to halt, with no halting prefices or extensions. For instance, in the prefix-free variant of Iota from chapter 1.5, the program

encoding the lambda term

$$t = (\lambda x.xx)(\lambda x.xx)$$
$$= 1111010011101001001001110100100100111010011101001001001110100100100$$

has no normal form, and no extension of it halts, since it does not read any bits. Then $\mathrm{dom}(U) \cup \{t\}$ is also prefix free. Now consider the set of strings

$$S = \left(\prod_{p \,\in\, \mathrm{dom}(U)} p^* \right) \cdot t = \sum n \cdot t$$

where products are ordered finite concatenation, sums are set union, and each n is an ordered finite concatenation of elements of $\mathrm{dom}(U)$.

The set S is clearly prefix-free, since each element has exactly one copy of t at the very end, and no element of $\mathrm{dom}(U)$ is a prefix or extension of t. It corresponds to the domain of a prefix-free machine M that repeatedly reads in programs for U and executes them until it reads t, in which case it halts.

By the Kraft inequality,

$$\Omega_M = \sum_{nt \,\in\, S} 2^{-|nt|} = 2^{-|t|} \prod_{p \,\in\, \mathrm{dom}(U)} \frac{1}{1 - 2^{-|p|}} < 1.$$

Theorem 4. *The bits of Ω_M form a c.e. random sequence.*

Proof. Consider the set of strings
$$S' = \sum_{p \,\in\, \mathrm{dom}(U)} p \cdot t \subset S;$$

these are the programs for M that consist of a single halting program for U and the terminator t. Given m bits of Ω_M, we can clearly compute the halting status of all $p \in \mathrm{dom}(M)$ such that $|p| \leq m$; but these include the programs in S', so it also tells us the halting status of all $p \in \mathrm{dom}(U)$ such that $|p| < m - |t|$. We can then pick a string not in the set of outputs of such programs, and halt. Thus, there exists a computable function $\Psi : X \to X$ with the property that

$$H(\Psi(\Omega_M[m])) > m - |t|.$$

By the universality of U,
$$H(\Omega_M[m]) + c_\Psi \geq H(\Psi(\Omega_M[m])) > m - |t|$$

and
$$H(\Omega_M[m]) > m - (c_\Psi + |t|),$$

which completes the proof. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

3.3   Natural halting probability

Consider the quasi-lex ordering $b : \mathbb{N}^+ \to \{0,1\}^*$—that is, $b(p)$ is the string formed by removing the leading 1 from the binary expansion of p. Unlike $B(p)$ in section 2, this map is undefined on 0 and maps $1 \to \lambda, 2 \to 0, 3 \to 1, 4 \to 00$, etc. Given a universal prefix-free machine U defined over $X = \{0,1\}^*$ we

define

$$\zeta_U = \sum_{p \in \mathbb{N} \mid b(p) \in \mathrm{dom}(U)} 1/p.$$

It is clearly computably enumerable; we show that this real is not equal to Chaitin's Ω_U, but is still random.

Theorem 5. $\Omega_U > \zeta_U > \Omega_U/2$.

Proof. $2^{-\lfloor \log_2(p) \rfloor} \geq 2^{-\log_2(p)} > 2^{-\lfloor \log_2(p) \rfloor - 1}$, where equality holds only when p is a power of two, so

$$\Omega_U = \sum 2^{-|b(p)|} = \sum 2^{-\lfloor \log_2(p) \rfloor} > \sum 2^{-\log_2(p)} = \zeta_U > \sum 2^{-\lfloor \log_2(p) \rfloor - 1} = \sum 2^{-|b(p)|-1} = \Omega_U/2,$$

where all sums are over $p \in \mathbb{N} \mid b(p) \in \mathrm{dom}(U)$ and we have used the fact that the set of strings in the domain of U cannot all have lengths that are integer powers of two: due to the universality of U, there must exist some integer n such that for all $p \in \mathrm{dom}(Iota)$, there exists a string q_p such that $U(q_p) = Iota(p)$ and $|q_p| \leq |p| + n$. Also, all strings of the form $11010100x$ halt on Iota, where x is a preorder traversal of any full binary tree; this is equivalent to the lambda term $K(x)$. If we assume that the domain of U consists solely of strings whose lengths are powers of two, this implies that, even ignoring the restriction that halting programs cannot be prefices or extensions of each other, there are only

$$\sum_{i=0}^{\lfloor \log_2(n+8+|x|) \rfloor} 2^i = 2^{\lfloor \log_2(n+8+|x|) \rfloor + 1} = \text{the smallest power of two greater than } n + 8 + |x|$$

strings of the appropriate form. This grows linearly (if sporadically) with $|x|$, while the number of halting programs of that form, the $|x|$th Catalan number, grows exponentially with $|x|$, and we reach a contradiction. $\square$

Theorem 6. *The bits of ζ_U form a random sequence.*

Proof. Given $m + 1$ bits of ζ_U, we run programs in parallel and sum the weights of the halting programs until we can account for the bits. Then we know no more programs $b(j)$ such that $|b(j)| \leq m$ can halt: since

$$1/j = 2^{-\log_2(j)} > 2^{-\lfloor \log_2(j) \rfloor - 1},$$

the longest of such programs would affect at least the last bit. We can take the outputs of all the programs that have halted and output a string not in that set. The complexity of that string is greater than m; thus, there exists a computable function $\Psi : X \to X$ with the property that

$$H(\Psi(\zeta_U[m+1])) > m.$$

By the universality of U,

$$H(\zeta_U[m+1]) + c_\Psi \geq H(\Psi(\zeta_U[m+1])) > m$$

and

$$H(\zeta_U[m]) > m - (c_\Psi + 1),$$

which completes the proof. $\square$

3.4   Partial K-randomness

Tadaki [65] introduced the concept of partial randomness. Several of his open questions were answered by Calude *et al.* [16]. The equation describing the complexity of successive prefices of a random sequence describes a line with slope 1. The equation describing the prefices of a partially-random sequence describes a line with slope $0 < s < 1$.

Tadaki defines the c.e. partially-random real

$$\Omega^s = \sum_{p \,\in\, \mathrm{dom}(U)} 2^{-s|p|},$$

though we have altered his notation slightly. This number is $1/s$-random—that is, it satisfies

$$\exists c \ \forall m > 1 \ H(\Omega^s[m]) \geq m/s - c.$$

One may consider the (partial-)randomness of a sequence to be its complexity density. Martin-Löf and Chaitin independently showed [19, 50, 12] that there are no sequences x such that the *plain* complexity satisfies

$$\exists c \ \forall m > 1 \ K(x[m]) \geq m - c,$$

which has given rise to alternate definitions of random sequences with respect to the plain complexity [49]. We do not consider these in this thesis. However, we show that for all real, computable $s > 1$ there are sequences x^s such that

$$\exists c \ \forall m > 1 \ K(x^s[m]) \geq m/s - c.$$

We will call these sequences $1/s - K$-random.

Given a universal Turing machine U' (which may or may not be prefix-free), we define "the halting probability of U' at inverse temperature s" to be

$$\kappa^s = \left(\sum_{p \,\in\, \mathrm{dom}(U)} 2^{-s|p|} \right) \Bigg/ \left(\sum_{q \,\in\, \mathbb{N}} 2^{-s|b(q)|} \right) = (1 - 2^{-s+1}) \sum_{p \,\in\, \mathrm{dom}(U)} 2^{-s|p|}.$$

This converges for all real $s > 1$. We will explain the reference to temperature in section 3.6.

Theorem 7. *For real $s > 1$, $0 < \kappa^s < 1$.*

Proof. If U' is universal, then there must be some integer q such that $b(q) \notin \mathrm{dom}(U')$. Therefore the numerator, which sums only over those q such that $b(q) \in \mathrm{dom}(U)$, is smaller than the denominator, which sums over all positive natural q. Since there must be at least one program that halts, the numerator is positive. $\square$

Since κ^s is in the interval $(0,1)$, we can interpret it as a halting probability.

Theorem 8. *For real, computable $s > 1$, the bits of κ^s form a $1/s - K-$random sequence.*

Proof. Given the first $m + \log_2(1 - 2^{-s+1})$ bits of κ^s, we can compute the halting status of all programs $p \in \mathrm{dom}(U')$ such that $|p| < m/s$. Then there is a computable function Ψ that, given $\kappa^s[m + \log_2(1 - 2^{-s+1})]$, produces a string not in the output of those programs, and

$$K(\kappa^s[m]) \geq m/s - (c_\Psi + \log_2(1 - 2^{-s+1})).$$

$\square$

Theorem 9. *If U' is a universal prefix-free machine, then for real, computable $s > 1$, the bits of κ^s form a $1/s-$random sequence.*

Proof. The proof proceeds exactly as in the previous theorem, replacing the plain complexity K with the prefix-free complexity H. Note that in this case, κ^s is just a computable factor times Ω^s. $\qquad\square$

3.5 Riemann

Given a universal Turing machine U', we define the natural halting probability at inverse temperature s to be

$$
\kappa_n^s = \left(\sum_{q \,\in\, \mathbb{N} \,\mid\, b(q) \,\in\, \mathrm{dom}(U')} q^{-s} \right) \Big/ \left(\sum_{q \,\in\, \mathbb{N}} q^{-s} \right)
$$
$$
= \zeta_{U'}(s)/\zeta(s),
$$

where $\zeta(s)$ is Riemann's zeta function and

$$
\zeta_{U'}(s) = \sum_{q \,\in\, \mathbb{N} \,\mid\, b(q) \,\in\, \mathrm{dom}(U')} q^{-s}.
$$

Given that both $\zeta(s)$ and Ω_M can be written as infinite products, it begs the question of whether $\zeta_{U'}(s)$, the *natural* halting probability at inverse temperature s, can be written as one.

Given a UTM U', we can define the set

$$
V = \{p_i | b(i) \in dom(U')\},
$$

where p_i is the ith prime. Consider the set of natural numbers

$$
S' = \{n | (\text{prime factors of } n) \in V\}
$$

The set S' is the domain of a machine M' that performs the following steps on an input x:

1. Compute $n = b^{-1}(x)$.

2. Compute the prime factors p_i of n.

3. For each p_i, simulate $U'(b(i))$.

4. Output the empty string.

Then

$$
\zeta_{M'}(s) = \sum_{n \in S'} n^{-s} = \prod_{p \in V} 1/(1 - p^{-s}),
$$

and

$$
\kappa_n^s = \zeta_{M'}(s)/\zeta(s) = \zeta(s) \prod_{p \notin V} (1 - p^{-s})/\zeta(s).
$$

Theorem 10. *The program $b(j)$ halts if and only if*

$$
\kappa_n^{\frac{2\pi i}{\log(p_j)}} \neq 0.
$$

Proof. The function κ_n^s is zero when

$$s = \frac{m2\pi i}{\log(p)}$$

for any $p \notin V$ and any $m \in \mathbb{N}$. These values are purely imaginary, while the nontrivial zeros s of Riemann's zeta function all have $\text{Re}(s) > 0$. Therefore κ_n^s is well defined at that point and is equal to zero only if $p_j \notin V$. $\qquad\square$

The zeros of κ_n^s therefore encode halting information.

Theorem 11. *For all real, computable $1 < s < s_0$ there exist c, t such that for all $m > t$,*

$$K(\kappa_n^s[m]) \geq m/s_0 - c.$$

Proof. The prime number theorem implies that for $i > 5$, $i\log(i) < p_i$. Given ms bits of κ_n^s, we can compute the halting status of all programs $b(i)$ such that $p_i < 2^m$.

$$i\log(i) < 2^m$$
$$i < 2^m/W(2^m)$$
$$|b(i)| = \lfloor \log_2(i) \rfloor < m - \log_2(W(2^m)) = W(2^m)/\log(2)$$

Here, W is Lambert's W-function. Therefore, given ms bits, we can compute the halting status of all programs whose length is less than $W(2^m)/\log(2)$. Since

$$\lim_{m \to \infty} \frac{W(2^m)}{m\log(2)} = 1,$$

the result follows. We call such sequences asymptotically $1/s - K-$random. $\qquad\square$

Theorem 12. *If U' is a universal prefix-free machine, then for all real, computable $1 < s < s_0$ there exist c, n such that for all $m > n$,*

$$H(\kappa_n^s[m]) \geq m/s_0 - c.$$

Proof. The proof is exactly the same as the previous theorem, replacing K with H. We call such sequences asymptotically $1/s$-random. $\qquad\square$

Tadaki's generalized halting probability is well-defined even for arbitrary univeral Turing machines, but has a pole at $s = 1$. Since the natural generalized halting probability can be written as a zeta function with the Riemann zeta function as a sharp bound, one can view the fact that there are no random sequences with respect to plain complexity as a proof of the infinitude of primes!

3.6   Physical systems and open questions

We report in this chapter on some work in progress related to the physical nature of the halting probabilities and some open problems.

3.6.1   The quantum Riemann system

It is well known [46, 57, 7] that Riemann's zeta function can be interpreted as the partition function of a quantum system in equilibrium at inverse temperature s—that is, the system is in a classical mixture

of states, each with probability proportional to $\exp(-Hs)$, where $s = 1/kT$, k is Boltzmann's constant, and T is the temperature in Kelvins.

The Hamiltonian for the Riemann system is the sum of the Hamiltonians for the QHOs.

$$H = \sum_n \frac{\omega_n^2}{2} x^2 - \frac{1}{2} \frac{\partial^2}{\partial x^2} - \frac{\omega_n}{2},$$

where $\omega_n = \log(p_n)$ and we've chosen units such that $m = \hbar = 1$. The energy basis of the Riemann system is spanned by the set of vectors $|n>$ such that

$$H|n> = \log(n)|n> \ \text{ and } \ \exp(H)|n> = n|n>,$$

In analogy to the absorbtion and emission of photons by a QHO, the union over n of $\log(p_n)$ is often called the set of "primons."

The function κ_n^s has the form of an expectation value on a Riemann system at inverse temperature s. We can define a quantum computer that measures $\exp(H)$ on the current state, converts that number into a program for M' via the bijection b, executes it for at most T steps, and sets a qubit to $|1>$ if the program halted within T steps or $|0>$ if it did not. Then in the limit as T approaches infinity, the expectation value of the qubit will be κ_n^s.

3.6.2 Tadaki's universal semi-POVM

Tadaki [64, 66] has introduced a quantum analog of Ω for finite- and infinite-dimensional systems. We would like to establish what connection exists between the expectation value above and this computably enumerable sum of elements of the semi-POVM.

3.6.3 Variable base computers

The "natural" halting probability ζ_U depends indirectly on base 2 through the alphabet on which U acts. Calude *et al.* have considered [15] the randomness of strings over a set of finite alphabets X_i such that the ith symbol of the string is an element of X_i. If there is a finite bound b on the magnitude of X_i, then one can define a halting probability using b as a base; this is less than satisfactory, however, since it essentially maps the string into $\{0, \ldots, b-1\}^*$. If X_i has no finite bound, then it's unclear what base to use for the standard halting probability of such a machine.

The natural halting probability does not have either of these problems. In fact, one can even define the natural halting probability for computers such that the alphabet for the next symbol depends on the state of the machine. It's likely that this halting probability will be useful for producing a simple proof that random numbers are random in any base.

3.6.4 Partial 2-randomness

Since there are no sequences x such that

$$\exists c \ \forall m > 1 \ K(x[m]) \geq m - c,$$

the definition

$$\exists c \ K(x[m]) \geq m - c \text{ infinitely often}$$

has been proposed [49]. Sequences satisfying this definition have been shown to be 2-random [51, 52]; the canonical example of a 2-random sequence is the bits of the halting probablility of a universal prefix-free machine $U^{0'}$, i.e. one with oracle access to the halting probability of another universal prefix-free machine U.

What can be said about the randomness of partially 2-random strings? Inserting a zero between each bit of a 2-random string gives a sequence that is 1/2-random and 1/2-2-random. On the other hand, if we define a machine C such that $C(xx) = U^{0'}(x)$, the halting probability of C is 1/2-2-random, but it's much less clear what the randomness with respect to U is.

3.6.5 Halting primes in number theory and geometry

There are deep relationships between number theory and geometry. Borrowing a table from [2],

NUMBER THEORY	COMPLEX GEOMETRY
Integers	Polynomial functions on the complex plane
Rational numbers	Rational functions on the complex plane
Prime numbers	Points in the complex plane
Integers mod p^n	$(n-1)$st-order Taylor series
p-adic integers	Taylor series
p-adic numbers	Laurent series
Adeles for the rationals	Adeles for the rational functions
Fields	One-point spaces
Homomorphisms to fields	Maps from one-point spaces
Algebraic number fields	Branched covering spaces of the complex plane

If we restrict ourselves to "halting primes"—those primes p_i such that $b(i)$ halts—what happens?

3.6.6 Riemann's hypothesis

The definition of the function $\zeta_{U'}(s)$ is the same as the Riemann zeta function, but with the primes restricted to those primes p_i such that $b(i)$ halts. We can choose U' such that it halts on all programs less than a given length n; as $n \to \infty$, $\zeta_{U'}(s) \to \zeta(s)$. Can we say anything about the zeros of Riemann's zeta function by complexity-theoretic means?

Riemann's zeta function also satisfies the functional equation

$$F(s) = \pi^{-s/2}\Gamma(s/2)\zeta(s) = F(1-s).$$

Does $\zeta_U(s)$ satisfy a functional equation?

3.7 Conclusion

Chaitin's Ω can be written not only as an infinite sum, but also as an infinite product over halting programs. While Chaitin's definition explicitly assumes a binary alphabet (though the generalization to arbitrary bases is straightforward), there is a natural formulation such that the definition is independent of the underlying machine, and even accounts for machines defined over mixed-base strings.

Tadaki proposed to parameterize Ω, effectively turning it into a zeta function. The fact that Chaitin had to resort to prefix-free machines to have Ω converge is just an illustration of the pole of the zeta function at $s = 1$ for many universal Turing machines.

The natural generalized halting probability can be written as a product over primes, just like in the Riemann zeta function. In the limit as the universal machine is defined to halt on more an more programs, the zeta function approached Riemann's as a sharp bound. It would be interesting to know if AIT has anything to say about solving the Riemann hypothesis, or can shed light on any number of related questions.

Conclusion

We've presented new prefix-free machines whose designs are very simple; we hope they will be used in the study of concrete AIT. The halting probability of such machines is a random sequence; when such a sequence is the result given by a measurement of a system, the system itself can be shown to satisfy an uncertainty principle equivalent to Heisenberg's uncertainty principle. This uncertainty principle also implies Chaitin's strongest form of incompleteness.

We've shown that Tadaki's generalized halting probability can apply to all Turing machines, prefix-free or not, and that it has the form of a zeta function. Zeta functions are often partition functions of a dynamic system composed of loops; we've shown that the loops in this case correspond to halting programs. We've introduced the concept of partially K-random sequences, and shown that while there are no random sequences, in a certain sense we can define sequences that are "as close as we want" to random.

Finally, we noted a relationship with Riemann's zeta function, and showed that some of the zeros of our zeta function encode halting information. There are many open problems to consider, and we hope our work will encourage others to join the effort in answering them.

Bibliography

[1] Baez, J. "This Week's Finds in Mathematical Physics (Week 199)." December 8, 2003. `http://math.ucr.edu/home/baez/week199.html`

[2] Baez, J. "This Week's Finds in Mathematical Physics (Week 218)." June 5, 2005. `http://math.ucr.edu/home/baez/week218.html`

[3] Barker, C. "Iota and Jot: the simplest languages?" 2001. `http://ling.ucsd.edu/~barker/Iota/`

[4] Barker, C. "Zot." 2002. `http://ling.ucsd.edu/~barker/Iota/zot.html`

[5] Barrow, J. D. *Impossibility: The Limits of Science and the Science of Limits*, Oxford University Press, Oxford, 1998.

[6] Barrow, J. D. "Mathematical jujitsu: Some informal thoughts about Gödel and physics", *Complexity* 5, 5 (2000), 28–34.

[7] Bost, J.-B. and Connes, A. "Hecke Algebras, Type III factors and phase transitions with spontaneous symmetry breaking in number theory," *Selecta Math.* (New Series), 1 (1995) 411–457.

[8] Calude, C. S. *Information and Randomness - An Algorithmic Perspective.* 2nd Edition, Revised and Extended, Springer-Verlag, Berlin, 2002. p. 24, 104

[9] Calude, C. S. "Incompleteness, complexity, randomness and beyond," *Minds and Machines: Journal for Artificial Intelligence, Philosophy and Cognitive Science*, 12, 4 (2002), 503–517.

[10] Calude, C. S. "Chaitin Ω numbers, Solovay machines and incompleteness," *Theoret. Comput. Sci.*, 284 (2002), 269–277.

[11] Calude, C. S. and Chaitin, G. J. "Randomness everywhere," *Nature*, 400, 22 July (1999), 319–320.

[12] Calude, C. S., and Chiţescu, I. "Upper limitation of Kolmogorov complexity and universal P. Martin-Löf tests," *J. Computational Math.* 7(1989), 61–70.

[13] Calude, C., Dinneen, M. J., and Shu, C.-K. "Computing a glimpse of randomness," *Experimental Mathematics*, 2 (2002), 369–378

[14] Calude, C. S. and Pavlov, B. "The Poincaré–Hardy inequality on the complement of a Cantor set," in D. lpay, I. Gohberg, V. Vinnikov (eds.). *Interpolation Theory, Systems Theory and Related Topics*, Operator Theory: Advances and Applications, Vol. 134, Birkhäuser Verlag, Basel, 2002, 187–208.

[15] Calude, C. S., Staiger, L., and Svozil, K. "Randomness Relative to Cantor Expansions," *Comm. Nonlinear Science Numerical Simulation*, 10/8 (2005), 921–930.

[16] Calude, C. S., Staiger, L., and Terwijn, S. A. "On partial randomness," *Annals of Pure and Applied Logic,* accepted.

[17] Casti, J. L. and Traub, J. F. (eds.). *On Limits*, Santa Fe Institute Report 94-10-056, Santa Fe, NM, 1994.

[18] Casti, J. L. and Karlquist, A. (eds.). *Boundaries and Barriers. On the Limits to Scientific Knowledge*, Addison-Wesley, Reading, MA, 1996.

[19] Chaitin, G. J. "On the lengths of programs for computing binary sequences." *J. Assoc. Comput. Mach.* 13, 549–569. 1966.

[20] Chaitin, G. J. "A theory of program size formally identical to information theory," *Journal of the ACM* 22 (1975), 329–340. `http://www.cs.auckland.ac.nz/CDMTCS/chaitin/acm75.pdf`

[21] Chaitin, G. J. "Randomness and mathematical proof," *Scientific American*, 232 (5) (1975), 47–52.

[22] Chaitin, G. J. "Algorithmic Information Theory." *IBM Journal of Research and Development* 21 (1977), 350–359, 496.

[23] Chaitin, G. J. "Gödel's theorem & information," *International Journal of Theoretical Physics* 22 (1982), 941–954.

[24] Chaitin, G. J. *algorithmic Information Theory.* Cambridge University Press, Cambridge, 1987. `http://www.cs.auckland.ac.nz/CDMTCS/chaitin/cup.html`

[25] Chaitin, G. J. *Information, Randomness and Incompleteness, Papers on Algorithmic Information Theory*, World Scientific, Singapore, 1990. (Second edition)

[26] Chaitin, G. J. *Information-Theoretic Incompleteness*, Singapore: World Scientific, Singapore, 1992.

[27] Chaitin, G. J. "An Invitation to algorithmic Information Theory," *DMTCS'96 Proceedings*, Springer Verlag, Singapore, 1997, 1–23. `http://www.cs.auckland.ac.nz/CDMTCS/chaitin/inv.html`

[28] Chaitin, G. J. "Elegant Lisp Programs," *People and Ideas in Theoretical Computer Science,* C. Calude, ed. Springer-Verlag Singapore, 1999, 32–52. `http://www.cs.auckland.ac.nz/CDMTCS/chaitin/lisp.html`

[29] Chaitin, G. J. *The Unknowable*, Springer Verlag, Singapore, 1999.

[30] Chaitin, G. J. "Computers, paradoxes and the foundations of mathematics," *American Scientist* 90 March–April (2002), 164–171.

[31] Church, A. "An unsolvable problem of elementary number theory," *American Journal of Mathematics*, 58 (1936), 345–363.

[32] Church, A. "A note on the *Entscheidungsproblem*," *Journal of Symbolic Logic*, 1 (1936), 40–41.

[33] Clay Mathematics Institute. *Millenium Problems.* `http://www.claymath.org/millennium/`

[34] de Bruijn, N. G. "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation," *Indagationes Mathematicae* 34, 381–392, 1972.

[35] Delahaye, J.-P. *Information, Complexité et Hasard*, Hermes, Paris, 1994.

[36] Deutsch, D. *The Fabric of Reality*, Allen Lane, Penguin Press, New York, 1997.

[37] Downey, R., Hirschfeldt, D. *Algorithmic Randomness and Complexity*, Springer, Heidelberg, (to appear).

[38] Feferman, S., Dawson, J., Jr., Kleene, S. C., Moore, G. H., Solovay, R. M., van Heijenoort, J. (eds.). *Kurt Gödel Collected Works*, Vol. I, Oxford University Press, New York, 1986.

[39] Feferman, S., Dawson, J., Jr., Kleene, S. C., Moore, G. H., Solovay, R. M., van Heijenoort, J. (eds.). *Kurt Gödel Collected Works*, Vol. II, Oxford University Press, New York, 1990.

[40] Fokker, J. "The systematic construction of a one-combinator basis for Lambda-terms," *Formal Aspects of Computing* 4:776–780. 1992. `http://www.cs.uu.nl/people/jeroen/article/combinat/combinat.ps`

[41] Geroch, R., Hartle, J. B. "Computability and physical theories," *Foundations of Physics* 16, 6 (1986), 533–550.

[42] Gödel, K. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter, *Systeme Monatshefte für Mathematik und Physik*, 38 (1931), 173–198. (Received 17 November 1930)

[43] Hawking, S. W. "Gödel and the End of Physics," *Dirac Centennial Celebration*, Cambridge, UK, July 2002, `http://www.damtp.cam.ac.uk/strtst/dirac/hawking/`.

[44] Heisenberg, W. "Quantenmechanik," *Die Naturwissenschaften* 14 (1926), 899–894.

[45] Heisenberg, W. "Über den Anschaulichen Inhalt der Quantentheoretischen Kinematik und Mechanik," *Zeitschrift für Physik* 43 (1927), 172–198. (Received 23 March 1927) English translation in J.A. Wheeler, H. Zurek (eds.). *Quantum Theory and Measurement*, Princeton Univ. Press, Princeton, 1983, 62–84.

[46] Julia, B. L. "Statistical theory of numbers," in Luck, J. M., Moussa, P. and Waldschmidt, M. (eds.) *Number Theory and Physics*, Springer Proceedings in Physics, Vol. 47, Springer-Verlag, Berlin, 1990, 276–293. Summarized by Matthew Watkins in `http://www.maths.ex.ac.uk/~mwatkins/zeta/Julia.htm`

[47] Kennard, E. H. "Zur Quantenmechanik einfacher Bewegungstypen," *Zeitschrift für Physik* 44 (1927), 326–352.

[48] Kolmogorov, A. N. "Three approaches to the quantitative definition of information." *Prob. Inform. Transmission.* 1, 4–7. 1965.

[49] Loveland, D. W. "On minimal-program complexity measures," *ACM Symposium on the Theory of Computing (STOC)*, ACM Press, New York, May 1969, 61–78.

[50] Martin-Löf, P. "Complexity oscillations in infinite binary sequences," *Zeitschrift fr Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 19 (1971), 225–230.

[51] Miller, J. S. "Every 2-random real is Kolmogorov random," *J. Symbolic Logic* 69, 3 (2004), 907–913

[52] Nies, A., Stephan, F., and Terwijn, S. A. "Randomness, relativization and Turing degrees," *Journal of Symbolic Logic*, to appear.

[53] Pauli, W. In Hermann, A., von Meyenn, K. and Weiskopf, V. F. (eds.) *Wissentschaftlicher Briefwechsel mit Bohr, Einstein, Heisenberg u.a.* Volume 1 (1919–1929), Springer-Verlag, Berlin, 1979.

[54] Peres, A. "Einstein, Gödel, Bohr," *Foundations of Physics* 15, 2 (1985), 201–205.

[55] Peres, A. and Zurek, W. H. "Is quantum theory universally valid?" *Am. J. Phys.* 50(9) (1982), 807–810.

[56] Rudiak-Gould, B. "Lazy-K." `http://homepages.cwi.nl/~tromp/cl/lazy-k.html`

[57] Sabbagh, K. *Dr. Riemann's Zeros*, Atlantic Books, 2002, 134–136.

[58] Shannon, C. E. "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, 379–423 and 623–656, July and October, 1948. `http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html`

[59] Solomonoff, R. J. "A formal theory of inductive inferrence I, II." *Inform. Control* 7, 1–22, 224–254. 1964.

[60] R.M. Solovay. "A version of Ω for which ZFC can not predict a single bit," in C.S. Calude, G. Păun (eds.). *Finite Versus Infinite. Contributions to an Eternal Dilemma*, Springer-Verlag, London, 2000, 323-334.

[61] Svozil, K. "Computational universes," *CDMTCS Research Report* 216, May 2003; *Chaos, Solitons & Fractals*, to appear.

[62] Svozil, K. Private communication to Calude, 8 February 2004.

[63] Tromp, J. "Binary Lambda Calculus and Combinatory Logic." Sep 14, 2004. `http://homepages.cwi.nl/~tromp/cl/LC.pdf`

[64] Tadaki, K. "Upper bound by Kolmogorov complexity for the probability in computable POVM measurement." Proceedings of the 5th Conference on Real Numbers and Computers, RNC5, 193–214, 2003.

[65] Tadaki, K. "A generalization of Chaitin's halting probability and halting self-similar sets." Hokkaido Math. J., **31** (2002), pp.219–253.

[66] Tadaki, K. "An extension of Chaitin's halting probability Ω to measurement operator in infinite dimensional quantum system." Presented at Real Numbers and Computers 2004 (RNC6), to appear in a special issue of *Theoretical Informatics and Applications*, EDP Sciences.

[67] Tarski, A. *Introduction to Logic and to the Methodology of Deductive Sciences*, Oxford University Press, New York, 1994. (4th edition)

[68] Turing, A. "On computable numbers, with an application to the *Entscheidungsproblem*," *Proceedings of the London Mathematical Society*, Series 2, 42 (1936), pp 230–265. Errata appeared in Series 2, 43 (1937), pp 544–546.

A

A lambda-calculus-to-Javascript source-to-source filter

```
// The "Be Lazy" wrapper
function _(x){return function(){return x}}

// Evaluate the parsed source
function __(x){
    return eval("(function(){var x="+parse(x)+"; return x})()")
}

// The guts
function parse(x) {
    if (x.indexOf('‘')==-1) return x;
    var pos=1, count, start, c;

    // find left tree
    for (count=0, start=pos;
         count >= 0 && pos < x.length;
         pos++)
    {
        c=x.charAt(pos);
        count += (c=='‘')?1:((c==' ' || c=='^')?-1:0);
    }
    var left = x.substring(1,pos);

    // find right tree
    for (count=0, start=pos;
         count >= 0 && pos < x.length;
```

```
        pos++)
    {
        c=x.charAt(pos);
        count += (c=='`')?1:((c==' ' || c=='^')?-1:0);
    }
    var right = x.substring(start,pos);

    if (left.substring(0,2)=='`^')
        return "function(){return function("+
                left.substring(2)+"){return "+parse(right)+"()}}";
    return "function(){return "+parse(left)+"()("+parse(right)+")}";
}


S=__("```^x ``^y ``^z ``x z `y z");
K=__("``^x ``^y x");
I=__("``^x x");
omega=__("``^x `x x");
Omega=__("`omega omega");


// Calling syntax:
// K()(I)(Omega)(S) = S()
// or
// __("```K I Omega S")() = S()
```

B
An implementation of Keraia

```
// Keraia's alphabet
_0=__("'''^c 'c Keraia");


// This version of _1 uses Javascript strings
// instead of the Pair combinator
_1=__("'''^c ''^L ''''Bit L ''^c 'c _(0) L "+
      "'''^l ''^R ''''Bit R ''^c 'c _(0) R ''^r 'c ''Cat l r");


// Assembles the string to interpret
Cat=function(){return
        function(x){return
            function(y){return
                "1"+x()+y()
    }}};


// 'Bit _0 = K, 'Bit _1 = 'K I
Bit=__("'''^b ''b 'K 'K K 'K 'K 'K 'K 'K 'K I");


Keraia=function(){
    return function(x){
        function parse(x) {
            if (x.indexOf('1')==-1) return "_"+x+" ";


            var pos=1, count, start, c;


            // find left tree
            for (count=0;count >= 0 && pos < x.length; pos++)
            { c=x.charAt(pos); count += (c=='1'||c=='3')?1:-1; }
            var left = x.substring(1,pos);
```

```
            // find right tree
            for (count=0, start=pos;
                 count >= 0 && pos < x.length;
                 pos++)
            { c=x.charAt(pos); count += (c=='1'||c=='3')?1:-1; }
            var right = x.substring(start,pos);

            if (left.substring(0,2)=='10')
            {
                var arg=left.substring(2).
                        replace(/0/g,'2').
                        replace(/1/g,'3');
                return "''^_"+arg+" "+
                        parse(right.replace(
                        new RegExp(left.substring(2),'g'),arg));
            }
            return "'"+parse(left)+parse(right);
        }
        return __(parse(x())
// uncomment this line for the prefix-free version
//          .replace(/_0/g,'R')
        )();
    }
}
epsilon=_0;
I_k = __("'''''epsilon _1 _1 _0 _0 _0");            // ''^x x

K_k = _( epsilon()(_1)(_1)(_0) (_1)(_0)(_1)(_0)(_0)  // ''^x
               (_1)(_1)(_0) (_0)                     // ''^y
               (_1)(_0)(_1)(_0)(_0) );               // x
// These functions are only used in the prefix-free version
InputBits='';

// Input monad
R=function(){
    return function(x){
        // loop on underflow
        if (InputBits=='') Omega();

        var c=InputBits.charAt(0);
        InputBits=InputBits.substring(1);
        return c=='1'?K()(I)(x):K()(x);
    }
}

pf-Keraia=function(x){
    var pos=0, count, c;

    // find a full tree
```

```
    for (count=0; count >= 0 && pos < x.length; pos++)
    { c=x.charAt(pos); count += (c=='1'||c=='3')?1:-1; }

    // loop on syntax error
    if (count>-1) Omega();

    // Take the remainder of the bits as input
    InputBits = x.substring(pos);

    var combo = _(Keraia()(_(x.substring(0,pos))));

    // loop on overflow
    if (InputBits != '') Omega();

    return combo;
}

// pf-Keraia("1001")() = I()
```