Some Results on Kolmogorov-Chaitin Complexity

David Lawrence Schweizer

Computer Science Department
California Institute of Technology

5233:TR:86

Some Results on Kolmogorov–Chaitin Complexity

by

David Lawrence Schweizer

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

Department of Computer Science
California Institute of Technology
Pasadena, California

1986

5233:TR:86

This work was supported by Caltech's Program in Advanced Technologies, sponsored by Aerojet General, General Motors, GTE, and TRW.

Acknowledgements

My advisor, Yaser Abu-Mostafa, guided me toward and through the work which makes up this thesis. Young-il Choo listened patiently to early versions of proofs, and provided invaluable assistance in convincing the TEX program to cooperate. Al Barr helped me over various obstacles in the earlier parts of my Caltech career.

My parents have always encouraged and supported my academic endeavors, sometimes with greater faith in my eventual success than I had myself.

To these, and to many others who have helped me realize just how much there is to learn, I am deeply grateful.

Contents

Acknowledgements ii

Introduction 1

Chapter 1 On Time Bounded Kolmogorov–Chaitin Complexity 2

1. Notation and Definitions 2
2. On the Existence of Apparently Random Strings 5
3. On the Significance of Apparently Random Strings 8

Chapter 2 On the Algorithmic Information of Halting Oracles 10

1. Notation and Definitions 11
2. On Discussing the Complexity of Infinite Binary Strings 12
3. Remarks on Relativized Kolmogorov–Chaitin Complexity 13
4. On the Algorithmic Information of Halting Oracles, Small Jumps 13
5. On the Algorithmic Information of Halting Oracles, Big Jumps 16

Chapter 3 On the Time Complexity of Extracting Bits
from a Compressed Halting Oracle 17

Conclusions and Directions for Future Research 20

Appendix 21

References 22

Introduction

This thesis describes results in algorithmic information theory, a field of research also known as Kolmogorov–Chaitin complexity. The thesis is divided into three chapters.

In the first chapter, the Kolmogorov–Chaitin definition of complexity is extended to include time complexity. This allows the absolute randomness of a binary string, as defined by its Kolmogorov–Chaitin complexity, to be distinguished from its apparent randomness, as measured by some bounded amount of computation. We show that under any computable bound on the amount of calculation allowed, there are binary strings of arbitrarily low complexity that appear totally chaotic. Further, we show that apparently random binary strings are a non-vanishing fraction of all binary strings of sufficient length.

In the second chapter, we study the Kolmogorov–Chaitin complexity of a class of infinite binary strings that arise naturally in computability theory. We begin with the halting oracle for Turing machines, and define a hierarchy of halting oracles for relativized computations. We show that the higher order oracles are of much greater Kolmogorov–Chaitin complexity than the first order oracle.

In the third chapter, we combine the results and techniques of the first two chapters to prove a strong result about the time complexity of extracting information from a compressed version of the halting oracle. We show that although the halting oracle is of low randomness, all significantly compressed versions are of high apparent randomness under all computable time bounds.

Chapter 1

On Time Bounded Kolmogorov–Chaitin Complexity

The average computer user is unwilling to wait 10^{10} years for a computation to yield a result. This impatience puts limitations on the algorithms such a person will use. In this chapter, we investigate the price the user must pay (in extra program length) to get results quickly. We show that the price can be very steep, and that it must be paid quite often.

1. Notation and Definitions

We will be working over the input alphabet $\Sigma = \{0,1\}$, and we write # for the blank symbol. We write s to denote a string in Σ^* (the set of all finite binary strings), and λ to denote the empty string. Binary strings can be put in *lexicographic order* thus: $\lambda, 0, 1, 00, 01, 10, 11, 000, \ldots$. If $\mathbf{s} \in \Sigma^n$ (that is, s is a binary string of length n) we write $|\mathbf{s}| = n$. Note that the binary strings of length k can be interpreted as the binary representations of the integers from 0 to $2^k - 1$ if we simply ignore leading zeros.

Formally, a *Turing Machine* is a triple (Q, Σ, δ), where:

(1) Q is a finite set of states, including two distinguished states q_0 (the start state) and q_1 (the halt state).

(2) Σ is a finite alphabet, including the blank symbol #.

(3) δ is a function from $Q \times \Sigma$ to $Q \times (\Sigma \cup \{L, R\})$, where L and R denote *move left* and *move right*, respectively. Further, once the machine enters the halt state, it never leaves it. That is, for any $\sigma \in \Sigma$, $\delta(q_1, \sigma) = (q_1, \sigma)$.

The machine is equipped with a single two-way infinite tape. Throughout this thesis, the tape alphabet is taken to be $\{\#, 0, 1\}$.

Informally, a Turing machine is a finite control that reads and writes symbols of a finite alphabet to and from a finite collection of infinite tapes. Changing the number of symbols or tapes allowed does nothing to enhance the computational power of the Turing machine, except to make it run faster or slower. It will be convenient to describe Turing machines in an informal manner throughout this thesis; everything presented can be made excruciatingly formal. Excellent expositions of the formal mechanics of Turing machines may be found in Turing's original paper [16], or in introductory texts such as Davis and Weyuker [5], Lewis and Papadimitriou [13], or Hopcroft and Ullman [9].

All functions are taken to have domain and range in the natural numbers. A function f is *computable* if there is a fixed Turing machine which will print the binary representation of $f(n)$ when started on the binary representation of n, for any value of n. Since there are only countably many Turing machines, but uncountably many functions from the natural numbers to the natural numbers, many functions are *uncomputable*.

There exists a class of Turing machines of paramount importance: the *universal Turing machines*. These machines are able to compute any function that can be computed by any Turing machine $\mathbf{M}$ by simulating the actions of $\mathbf{M}$ step by step.

We take $\mathbf{U}$ to be a particular implementation of a universal Turing machine with input alphabet Σ and states $\{q_0, q_1, \ldots, q_K\}$. We write $\rho(\mathbf{M})$ to denote the *encoding* of the Turing machine $\mathbf{M}$ and $\rho(\mathbf{M})\mathbf{w}$ to denote the binary string $\mathbf{w}$ concatenated onto the encoding of $\mathbf{M}$. (The encoding $\rho(\mathbf{M})$ is defined to be self-delimiting.) We call a string of the form $\rho(\mathbf{M})\mathbf{w}$ a *program*. The string $\mathbf{w}$ is called the *data*. For fixed $\rho(\mathbf{M})$, we refer to the various programs obtained by adding different data as *instances* of the encoding. (Note that $\rho(\mathbf{M}) = \rho(\mathbf{M})\lambda =$ the instance of $\rho(\mathbf{M})$ with the empty string as input.) If it is necessary to supply multiple strings as data, we can perform a syntactic transformation that makes the component parts of the data self-delimiting. We denote such a transformation by $[\mathbf{w}, \mathbf{v}]$. A very simple transformation that allows $|[\mathbf{w}, \mathbf{v}]| = 2(|\mathbf{w}| + |\mathbf{v}|) + 2$ is simply to write each bit of each string twice, and separate the strings with "01." This method generalizes to three or more strings.

Given $\rho(\mathbf{M})\mathbf{w}$ as input, $\mathbf{U}$ halts if and only if $\mathbf{M}$ would halt on the input $\mathbf{w}$, and leaves as output precisely what $\mathbf{M}$ would leave. $\mathbf{U}$ is further defined not to halt if its input is not a syntactically correct encoding of a Turing machine with input. It is convenient to think of

U as having an input tape, several work tapes, and an output tape. This convention allows a Turing machine **M** to have easy access to its own encoding, the binary string $\rho(\mathbf{M})$.

We use the term *simulation* to describe the situation of one Turing machine calculating the action of another Turing machine on a particular string. Note that since **U** itself is a Turing machine, it has an encoding which can be incorporated into other machines. Simulations of the action of **U** are of great importance.

We denote by **E** (for Everhalting) the machine which copies the contents of its input tape to its output tape and then halts. (Or, under the convention of Turing machines having only a single tape, justs halts.) The existence of this machine guarantees that every finite binary string is the output of some program. (The string **v** is generated by **U** when presented with $\rho(\mathbf{E})\mathbf{v}$.)

The status of a Turing machine computation at any moment can be represented by a quadruple called a *snapshot*. The quadruple $(q, \mathbf{v}, \sigma, \mathbf{w})$ indicates that the machine is in state q with the string **v** to the left of its head, the string **w** to the right of its head, the symbol σ currently being scanned, and the rest of the tape blank. We define the relation $\longrightarrow_{\mathbf{M}}$ on snapshots thus: if the machine **M** goes from the snapshot $(q, \mathbf{v}, \sigma, \mathbf{w})$ to the snapshot $(q', \mathbf{v}', \sigma', \mathbf{w}')$ in exactly one transition (i.e. by exactly one application of the transition function δ) then

$$(q, \mathbf{v}, \sigma, \mathbf{w}) \longrightarrow_{\mathbf{M}} (q', \mathbf{v}', \sigma', \mathbf{w}'),$$

and we define $\longrightarrow^*_{\mathbf{M}}$ to be the reflexive, transitive closure of $\longrightarrow_{\mathbf{M}}$. Note that

$$(q_1, \mathbf{v}, \sigma, \mathbf{w}) \longrightarrow_{\mathbf{M}} (q_1, \mathbf{v}, \sigma, \mathbf{w}).$$

The *Kolmogorov–Chaitin complexity* of a string **s** is defined as:

$$K(\mathbf{s}) = \min\{\, |\mathbf{p}| : (q_0, \lambda, \#, \mathbf{p}) \longrightarrow^*_{\mathbf{U}} (q_1, \lambda, \#, \mathbf{s}) \,\}.$$

That is,

$$K(\mathbf{s}) = \min\{\, |\mathbf{p}| : \mathbf{U} \text{ given } \mathbf{p} \text{ halts leaving } \mathbf{s} \text{ as output } \}.$$

Note that **p** is a binary string; we have not specified how it is divided into encoding and data. A string **s** is *random in the sense of Kolmogorov* if $K(\mathbf{s}) \approx |\mathbf{s}|$. These definitions were first presented by Kolmogorov [11], and were extended by Chaitin [4,3].

We now define the relation $\longrightarrow_{\mathbf{M}}^{t}$ recursively as follows:

(*i*) $$(q_i, \mathbf{u}, \sigma_j, \mathbf{v}) \longrightarrow_{\mathbf{M}}^{0} (q_i, \mathbf{u}, \sigma_j, \mathbf{v}),$$

(*ii*) $$(q_{i_1}, \mathbf{u_1}, \sigma_{j_1}, \mathbf{v_1}) \longrightarrow_{\mathbf{M}}^{t} (q_{i_2}, \mathbf{u_2}, \sigma_{j_2}, \mathbf{v_2})$$

if and only if there exists a snapshot $(q_i, \mathbf{u}, \sigma_j, \mathbf{v})$ such that

$$(q_{i_1}, \mathbf{u_1}, \sigma_{j_1}, \mathbf{v_1}) \longrightarrow_{\mathbf{M}}^{t-1} (q_i, \mathbf{u}, \sigma_j, \mathbf{v}) \quad \text{and} \quad (q_i, \mathbf{u}, \sigma_j, \mathbf{v}) \longrightarrow_{\mathbf{M}} (q_{i_2}, \mathbf{u_2}, \sigma_{j_2}, \mathbf{v_2}).$$

Intuitively, two snapshots satisfy this relation if the machine **M** transforms one snapshot into the other in t steps of computation. When we simulate, the machine doing the simulation can count the number of state transitions the simulated machine goes through during the course of the simulation. This allows a Turing machine to decide whether or not some other Turing machine produces a specified output within some time bound.

Finally, given a function τ, we define the *time bounded Kolmogorov–Chaitin complexity* by:

$$K_\tau(\mathbf{s}) = \min\{\, |\mathbf{p}| : (q_0, \lambda, \#, \mathbf{p}) \longrightarrow_{\mathbf{U}}^{\tau(|\mathbf{s}|)} (q_1, \lambda, \#, \mathbf{s}) \,\}.$$

Or,

$$K_\tau(\mathbf{s}) = \min\{\, |\mathbf{p}| : \mathbf{U} \text{ given } \mathbf{p} \text{ halts within } \tau(|\mathbf{s}|) \text{ steps leaving } \mathbf{s} \text{ as output } \}.$$

We assume that τ is always chosen large enough for **U** to reduce $\rho(\mathbf{E})\mathbf{s}$ to **s** within $\tau(|\mathbf{s}|)$ steps. We could eliminate this assumption by adding a clause defining $K_\tau(\mathbf{s})$ to be infinite if there is no program satisfying the requirement, but this makes the definition unnecessarily cumbersome given the weakness of the assumption above.

These definitions and conventions will be used throughout the thesis. Further, more specialized, definitions will be introduced at the beginning of chapter II.

2. On the Existence of Apparently Random Strings

We show that given a computable bound on the amount of time allowed for the production of a string from the program that generates it, there exist strings of arbitrarily low Kolmogorov–Chaitin complexity that appear maximally random. That is, given a notion

of quick, we show that there are strings that can be compressed to extremely short representations, but that cannot be recovered quickly from *any specification shorter than the original string.*

Reasons to be interested in the amount of time required to generate a binary string are most intuitively to be found in the realm of data compaction. Computer users often want to store large amounts of data, but are not concerned with the details of how the information is stored. It is important, however, that the stored data be readily accessible. The problem of data compression should really be called the problem of data re-expansion. Typically, we are concerned with the size of the compressed version and the difficulty of retrieval of the original information, not the difficulty of the compression process. Other areas of Computer Science are also well characterized as the study of compact representations. Computer Graphics seeks to model the real world in ways that permit generation of realistic images without having previously stored digitized versions of those images.

Suppose we limit the amount of computation that may be done to retrieve a binary string from its compressed version. Clearly, under some bounds some strings will require longer programs than they do under larger bounds. One might hope that a trade-off of space against time exists; that is, that restricting the amount of time allowed would cost some (but not all) of the space saved. Unfortunately, under any computable time bound at all there exist binary strings that appear incompressible (i.e. that are not generated by any short program within that time bound, but that have extremely short descriptions if we do not limit in advance the amount of computation allowed.)

More precisely, we have:

Theorem 1.1 Let τ and σ be any computable functions, with $\sigma(n) > n$. Then there exists a t_0 such that for all $t \geq t_0$ there exist binary strings $\mathbf{s}$ with $|\mathbf{s}| = \sigma(t)$ such that $K(\mathbf{s}) \leq t$ but $K_\tau(\mathbf{s}) \geq \sigma(t)$.

We call such a string *τ-incompressible.*

To emphasize the generality of this result, consider letting $\sigma(n) = A(n,n)$ and $\tau(n) = A(\sigma(n),\sigma(n))$, where A is Ackermann's generalized exponential. The theorem says that there is a program of length n that generates an enormous string, but takes a very long time to do it. But further, if we require a program to generate that string "fast" (and we should emphasize that $A(\sigma(n),\sigma(n))$ is huge) then we are completely stuck: we cannot get by with a program even one bit shorter than the string itself.

Proof The proof proceeds by constructing a Turing machine which attacks the problem by brute force: we examine the output of every short program that halts quickly, and then select a string not in the list of produced outputs. The proof is constructive, that is, it gives a procedure for exhibiting a τ-incompressible string.

We construct a Turing Machine M. Let $t_0 = |\rho(\mathbf{M})|$, and consider any instance $\rho(\mathbf{M})\mathbf{w}$ of $\rho(\mathbf{M})$ of length t. This proof relies heavily on the fact that Turing machines may be composed. Each step described below is complex, but clearly computable.

The Turing Machine M started on the input string w (or, equivalently, U started on the input string $\rho(\mathbf{M})\mathbf{w}$:

(1) Computes $\sigma(t)$ and $\tau(|\mathbf{s}|) = \tau(\sigma(t))$. We note that t does not need to be specified as a parameter to the computation: the program remains available on the input tape of U;

(2) Generates a list $\mathcal{L}$ of all binary strings of length $\sigma(t)$ in lexicographic order;

(3) Simulates the operation of U on each binary string of length $\leq \sigma(t) - 1$ until it (i.e. the simulated U) either halts or has computed for $\tau(\sigma(t))$ steps without halting;

(4) Removes from the list $\mathcal{L}$ every string of length exactly $\sigma(t)$ produced by a simulation; (A simulation must halt to be considered to have produced an output.)

(5) Prints the $\mathbf{w}^{\text{th}}$ remaining string; (We show below that there are at least $2^{|\mathbf{w}|}+1$ strings in $\mathcal{L}$ not produced by any simulation; hence the $\mathbf{w}^{\text{th}}$ remaining string exists.)

(6) Halts.

We note that M simulates U on $2^0 + 2^1 + \cdots + 2^{\sigma(t)-1} = 2^{(\sigma(t)-1)+1} - 1 = 2^{\sigma(t)} - 1$ potential programs. Since $\sigma(t) > t$, the $2^{|\mathbf{w}|}$ instances of $\rho(\mathbf{M})$ are in this collection, and each of them clearly computes for more than $\tau(\sigma(t))$ steps before it halts. Thus no more than $2^{\sigma(t)} - 1 - 2^{|\mathbf{w}|}$ strings are eliminated from $\mathcal{L}$, and each instance of $\rho(M)$ of length t will produce a distinct string of length $\sigma(t)$.

The construction above gives a program of length t that generates the string s. Thus $K(\mathbf{s}) \leq t$. But by the operation of M, s is not produced within $\tau(\sigma(t))$ steps by any program of length $\leq \sigma(t) - 1$. Thus $K_\tau(\mathbf{s}) \geq \sigma(t)$. ∎

Suppose we are given a binary string and wish to know whether or not it is τ-incompressible. We can simply enumerate all τ-incompressible strings of the same length as the given string, and then check to see if the given string is in the list. The time complexity of doing this is clearly a multiple of the time complexity of the operation of M, with the multiplier exponential in the length of the given string. The point is that the problem of generating

τ-incompressible strings and the problem of deciding if a given string is τ-incompressible are essentially the same. This has interesting ramifications for physics.

Erber and Putterman [7] have recently suggested that the validity of Quantum Mechanics be tested empirically by observing spectral emissions of single atoms stimulated by a laser. They hope that "...a search for unexpected patterns of order by cryptanalysis of the telegraph signal generated by the on/off time of the atom's fluorescence will provide new experimental tests of the fundamental principles of the quantum theory." The above theorem says that we can never increase our confidence in the validity of Quantum Mechanics on the basis of such an experiment: there could be an extremely short, but computationally complex, explanation for the string generated by the experiment.

There are some assumptions about the fundamental nature of computation and physical systems implicit in the discussion above. If the string generated truly requires enormous amounts of time to be produced, then how does the electron "decide" what to do quickly? The fundamental relation between randomness as studied in Quantum Mechanics and randomness as studied in computability theory is not at all well understood. Deutsch [6] has addressed the question, but it is far from entirely settled.

3. On the Significance of Apparently Random Strings

We now investigate how significant these apparently random strings are. That is, given τ and σ, how many strings of length $\sigma(n)$ are τ–incompressible? Leonid Levin [10] formulated the problem thus:

Let

$$S_{\tau,N} = \{\mathbf{s} \in \{0,1\}^N | K(\mathbf{s}) < K_\tau(\mathbf{s})\}.$$

Conjecture:

$$\lim_{N\to\infty} \frac{|S_{\tau,N}|}{2^N} > 0 \text{ for any computable } \tau.$$

The following theorem settles this conjecture affirmatively.

Theorem 1.2 For any fixed k and any computable function τ, there is an n_0 such that a constant fraction of all strings of length $n > n_0$ have $K(\mathbf{s}) \leq n - k$ but $K_\tau(\mathbf{s}) \geq n$.

Proof We use the definitions of theorem I.1, and base the proof on the machine $\mathbf{M}$ of that theorem. Let $\sigma(m) = m + k$. We again call $|\rho(\mathbf{M})| = t_0$. Since $|\mathbf{w}| = t - t_0$, we have 2^{t-t_0} instances of the encoding $\rho(\mathbf{M})$, each of which produces a distinct string of length $t + k$. Thus $2^{-(t_0+k)}$ — a fraction independent of n — of all strings of sufficient length can be compressed by k bits (i.e. are produced by programs k bits shorter than the strings themselves) if we are patient, but appear maximally random if we require that generating programs run under the time constraint τ. (We require that $n_0 = t_0$. To see that this suffices, take $\sigma(m) = m + 1$, and consider the string produced by $\rho(\mathbf{M})\lambda$.) ∎

Chapter 2

On the Algorithmic Information of Halting Oracles

We have already noted that there are functions not computable by Turing machines. Computing any function from the natural numbers to the natural numbers can be reduced to the problem of deciding some language, that is, determining whether or not certain binary strings are in a particular subset of Σ^*. This provides a convenient framework in which to study questions of *relativized computability.*

Suppose some agency provides us with a magical source of answers to a specified set of questions which we might be otherwise unable to answer. In this chapter we investigate some aspects of the additional computing power we would gain from such an oracle.

The Kolmogorov–Chaitin complexity can be thought of as a measure of the degree of uncomputability of infinite strings. If $\mathbf{X}_m$ denotes the first m bits of an uncomputable infinite binary string $\mathbf{X}$, the algorithmic information $K(\mathbf{X}_m)$ is the minimum number of bits needed to compute $\mathbf{X}_m$ on a universal Turing machine. How $K(\mathbf{X}_m)$ varies with m, for example, as $\log m$, $\sqrt{m}$, $\frac{1}{2}m$, etc., is a measure of how uncomputable $\mathbf{X}$ is.

On the other hand, the degree of uncomputability can be measured in terms of a hierarchy of halting oracles. If any desired number of bits of $\mathbf{X}$ can be computed by a fixed Turing machine which consults the n^{th} order halting oracle, but not by any machine that consults the $(n-1)^{\text{st}}$, then its degree of uncomputability is n.

The purpose of this section is to relate these two concepts of "degree of uncomputability," by giving exact estimates (identical lower and upper bounds) for the algorithmic information of halting oracles. Specifically, the algorithmic information of the i^{th} order halting oracle relative to a universal Turing machine which consults the j^{th} order halting oracle is

estimated for all i, j. The result shows a sharp contrast between the algorithmic information of the first order halting oracle (previously investigated by Chaitin [4]) and the second order halting oracle. Also, the essential changes in the Kolmogorov–Chaitin complexity of certain strings for different consulted oracles are discussed.

1. Notation and Definitions

An *oracle* for a language $L \subseteq \Sigma^*$ is an infinite binary string, where the n^{th} bit of the oracle is a 1 if and only if the n^{th} string of Σ^* (in lexicographic order) is in L.

An *oracle consulting Turing machine* ("OTM") is a Turing machine with an additional read-only tape, infinite in one direction (the *"oracle tape"*). At any stage of the computation the action and next state of the machine are determined by the current state and the symbols scanned on the work tape and the oracle tape. To the usual actions are adjoined actions for moving the oracle tape scanning head left or right. Intuitively, the oracle will be inscribed (by some means other than a Turing machine) on the oracle tape, and queries to the oracle will be accomplished by examining a particular bit of the oracle.

A *universal oracle consulting Turing machine* takes strings as input and halts with a specific output if and only if the input string is a syntactically correct encoding of an OTM (possibly including an input string for that machine) that halts with that output. (This is analogous to the definition of universal Turing machines given earlier.) We denote the universal OTM with some particular oracle $\mathbf{O}$ on its oracle tape by $\mathbf{U}(\mathbf{O})$. Note that the halting behavior of the universal OTM for some fixed string may be different for different oracles. (That is, a computation which depends on the contents of the oracle tape will proceed differently if the oracle tape is changed.)

We say that a function is *computable relative to the oracle* $\mathbf{O}$ (or *"$\mathbf{O}$-computable"*) if the function can be computed by an OTM with $\mathbf{O}$ on its oracle tape.

We define the *halting oracle for OTM's relative to the oracle* $\mathbf{O}$ to be the oracle for the language

$$L(\mathbf{O}) = \{\, \mathbf{s} : \mathbf{U}(\mathbf{O}) \text{ halts on } \mathbf{s} \,\}.$$

Finally, we define the *complexity of a string* $\mathbf{s}$ *relative to the oracle* O thus:

$$K^{\mathbf{O}}(\mathbf{s}) = \min\{\, |\mathbf{p}| : \mathbf{U}(\mathbf{O}) \text{ given } \mathbf{p} \text{ halts leaving } \mathbf{s} \text{ as output} \,\}.$$

That is, the complexity of a string is the length of the shortest program which produces the string. If the oracle tape is blank, then this is the usual Kolmogorov–Chaitin complexity. (In fact, if the oracle tape contains any computable string, then the relativized Kolmogorov–Chaitin complexity differs from the usual Kolmogorov–Chaitin complexity only by an additive constant.)

If $\mathbf{X}$ is some infinite binary string, $\mathbf{X}_m$ denotes the first m bits of $\mathbf{X}$.

For the remainder of this thesis, we will use the following hierarchy of oracle strings:

$\mathbf{H}^0$ is the empty string.

$\mathbf{H}^n$ is the halting oracle for OTMs relative to H^{n-1}.

Thus $\mathbf{H}^1$ is the halting oracle for normal Turing machines. This hierarchy corresponds to the sequence $\mathbf{0}$, $\mathbf{0}'$, $\mathbf{0}''$, ... of Turing degrees (see, e.g., Rogers [14]).

2. On Discussing the Complexity of Infinite Binary Strings

The complexity of an infinite binary string is a little harder to define than the complexity of a finite string. Clearly, we cannot expect a Turing machine to print an infinite string and then halt. The solution is to look at the complexity of all initial substrings.

Intuitively, an infinite string $\mathbf{X}$ is computable if there exists a fixed Turing machine which can compute $\mathbf{X}_m$ for any specified m. A string is uncomputable if we must specify not only how much we want, but also some information about the string.

It might seem that the complexity of (initial substrings of) computable strings does not grow with the length of the substring, but the complexity of (initial substrings of) uncomputable strings does. The problem is that it is possible for the *numeral* m to have high complexity.

It is therefore impossible to describe the behavior of the complexity of infinite strings by means of limits. We can, however, make the following statements about the Kolmogorov–Chaitin complexity of a computable infinite string $\mathbf{X}$:

$$\limsup_{m\to\infty} \frac{K(\mathbf{X}_m)}{\log m} = 1$$

and

$$\liminf_{m\to\infty} \frac{K(\mathbf{X}_m)}{\log\log\cdots\log m} = 0.$$

The first case is justified by noting that the numeral which represents the number m never requires more than $\lceil \log m \rceil$ digits. In the second case, we take the machine which computes $\mathbf{X}_m$ from m, and compose it with a machine that computes a function that grows extremely fast. We can always find a function that grows faster than the inverse of any fixed number of iterations of log, and thus generate enormously long initial substrings of $\mathbf{X}$ from very short inputs.

3. Remarks on Relativized Kolmogorov–Chaitin Complexity

1. $\mathbf{H}^{i+1}$ is not $\mathbf{H}^i$-computable. The case $i = 0$ is a statement of the usual halting problem. Other cases can be proved with exactly the same techniques.

2. $K^{\mathbf{H}^i}(s)$ is not $\mathbf{H}^i$-computable. (Assume that it is. Take the program that computes it and modify it to look for strings of very high complexity and print one. Since the program is of constant length, and we can specify how high we want the complexity to be with a logarithmic number of bits (the length of a numeral is log of the number it represents), we get a contradiction.) But $K^{\mathbf{H}^i}(s)$ is $\mathbf{H}^{i+1}$-computable. (Simply simulate all machines of appropriate length which the oracle says will halt and look for the shortest one generating s.)

3. A given string can have different complexities relative to different oracles. As an example, consider the worst string of length n relative to $\mathbf{H}^i$, that is, the string with the highest complexity (breaking any tie by lexicographic ordering). Relative to $\mathbf{H}^{i+1}$, we need only a constant program and $\log n$ bits of specification.

4. On the Algorithmic Information of Halting Oracles, Small Jumps

In this section and the next, we completely analyze the relative complexity of pairs of halting oracles. We find the somewhat surprising result that specifying an oracle one step above one we have requires much less information than specifying one two or more steps up. To be precise, we show:

$$K^{\mathbf{H}^i}(\mathbf{H}^j_m) = \begin{cases} \Theta(\log m), & i = j-1; \\ \Theta(m), & i < j-1. \end{cases}$$

(Θ is Knuth's [10] "Big Theta"; $f(n) = \Theta(g(n))$ means that $f = O(g)$ and $g = O(f)$, or, there exist c_1 and c_2 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$, for all n sufficiently large.)

The first case says that given some halting oracle $\mathbf{H}^i$, m bits of $\mathbf{H}^{i+1}$ can be compressed into a description of length essentially $\log m$, but no further. The second case says that all higher order halting oracles may be compressed only to some fraction of their length, but no further. We note that the omitted case, $i > j - 1$, is not interesting: given any halting oracle, we can compute lower order halting oracles at will.

Lemma about Lower Order Halting Oracles

Having said that the $i > j - 1$ case is uninteresting, we hasten to add that it is needed for later results. Therefore let us assume that we have $\mathbf{H}^i$ on the oracle tape and want bits of $\mathbf{H}^{i-1}$. We define a special OTM called the "checker," denoted **C**. Given the binary number w as input on the work tape, **C** reads the w^{th} bit of whatever is on its oracle tape (here assumed to be $\mathbf{H}^{i-1}$) and halts if that bit is a 1. If the bit is a 0, C enters an infinite loop. Since we can use $\mathbf{H}^i$ to decide the fate of C on any input w, we can determine the w^{th} bit of $\mathbf{H}^{i-1}$. By building checkers of checkers of checkers (etc.), we can work our way down to any halting oracle of degree lower than the one we have. For completeness, we note that we can always copy bits from the oracle tape to the work tape: this covers $i = j$. ∎

Theorem 2.1 We now wish to show that

$$K^{\mathbf{H}^i}(\mathbf{H}_m^{i+1}) = \Theta(\log m),$$

or, equivalently, that

$$K^{\mathbf{H}^i}(\mathbf{H}_{2^m}^{i+1}) = \Theta(m).$$

Proof This proof is in two pieces: first, we demonstrate by construction that the oracle may be compressed logarithmically; and second, we prove that it cannot be compressed further.

We define Ω_m^{i+1} to be the number of 1's in $\mathbf{H}_{2^m-1}^{i+1}$. (This is by no means the only way to compress a halting oracle. Chaitin's number Ω is equally dense; the proof using m bits of that string is essentially the same as what follows. See Gardner [8] for an excellent discussion of this material.)

We must first verify that if we are given Ω_m^{i+1} we have enough information to construct approximately 2^m bits of $\mathbf{H}^{i+1}$ using $\mathbf{U}(\mathbf{H}^i)$. We note that the number of 1's in some section of a halting oracle is the count of the number of programs which halt. In this case, we know how many programs of length less than m will halt. So we simulate the operation of all such programs until the correct number have halted. We then know which bits of $\mathbf{H}_{2^m-1}^{i+1}$ are 1's, and can write 0's in the other places, and thus print the entire $2^m - 1$ bits. (In the course of these simulations, we will need to refer to a great many bits of $\mathbf{H}^i$. In fact, it can be proved by a diagonalization argument that the number of bits we will need grows faster than any $\mathbf{H}^i$-computable function of m.)

Now assume that the halting oracle can be compressed further, i.e., that for some $\epsilon > 0$,

$$K^{\mathbf{H}^i}(\mathbf{H}_{2^m}^{i+1}) \leq m \cdot (1 - \epsilon).$$

This says that it is possible to use $m - \epsilon m$ bits to determine the fates of all programs of length less than m. Call such a string $\hat{\Omega}$, and consider the following Turing machine (called $\mathbf{T}$):

(1) Use $\hat{\Omega}$ to decide which programs of length less than m will halt;
(2) simulate all those programs;
(3) for every string that has been generated as the final output of one or more programs, find the shortest program that generates that string;
(4) find the longest of these programs;
(5) erase everything except the output of that program;
(6) halt.

We observe that there is a string of length m of complexity at least m. This follows from the fact that there are 2^m strings of length m, but only $2^0 + 2^1 + \cdots + 2^{m-1} = 2^m - 1$ programs of length less than m.

Now note that the encoding $\rho(\mathbf{T})$ of T has some fixed length; call it t. We recall that no string of length m has complexity greater than $m + |\rho(\mathbf{E})|$. For large m,

$$m - \epsilon m + t < m - \tfrac{\epsilon}{2} m.$$

But now we have found a contradiction. For, by the observation above, the output of $\mathbf{T}$ will be a string of complexity near m, but $\mathbf{T}$ and $\hat{\Omega}$ describe it with significantly fewer bits. Hence $\hat{\Omega}$ does not exist, and

$$K^{\mathbf{H}^i}(\mathbf{H}_m^{i+1}) = \Theta(\log m). \qquad \blacksquare$$

5. On the Algorithmic Information of Halting Oracles, Big Jumps

Now we wish to show that

Theorem 2.2

$$i < j - 1 \Rightarrow K^{\mathbf{H}^i}(\mathbf{H}_m^j) = \Theta(m).$$

Proof Our approach will be to use $2^r \cdot m$ bits of $\mathbf{H}^j$ to construct m bits of Ω^{i+1}. The exponent r depends only on $j - i$ and a constant.

Without loss of generality, let $i = j - 2$. (We have seen that we can compute lower order oracles from higher order ones.) Consider the following OTM $\overline{\mathbf{M}}$ with $|\rho(\overline{\mathbf{M}})| = k$ when started on a binary string $\mathbf{w}$ of length m:

(1) calculate the $(\mathbf{w}+1)^{\text{st}}$ bit of the binary numeral which represents the number of 1's in the first $2^{2^m} - 1$ bits of whatever is on its oracle tape;

(2) If that bit is a 1, halt;

(3) If it is a 0, enter an infinite loop.

If the oracle tape read by $\overline{\mathbf{M}}$ contains $\mathbf{H}^{i+1}$, then $\overline{\mathbf{M}}$ halts if and only if the $(\mathbf{w}+1)^{\text{st}}$ digit of $\Omega_{2^m}^{i+1}$ is a 1.

Assume we have $\mathbf{H}_{2^{m+k+1}}^{i+2}$. By definition, this contains halting information for all OTM programs of length less than or equal to $m + k$ with $\mathbf{H}^{i+1}$ on their oracle tapes. In particular, it contains halting information for $\overline{\mathbf{M}}$ with input strings of length m. There are 2^m of these programs, and each one yields exactly one bit of $\Omega_{2^m}^{i+1}$. We are done, since we have already shown that Ω^{i+1} is not compressible. ∎

Chapter 3

On the Time Complexity of Extracting Bits from a Compressed Halting Oracle

It is well known that the Kolmogorov–Chaitin complexity of initial substrings of the halting oracle is logarithmic in the length of the oracle which they specify. All the usual examples of corresponding compressions (e.g. Chaitin's number Ω) require unbounded amounts of time for expansion. Any method which requires simulating a group of Turing machines until all which will ever halt have halted must require an amount of time which grows faster than any computable function; if the time could be bounded above, we would be able to solve the halting problem. We now show that this is true of all such drastic compressions.

Theorem 3.1 Let $\hat{\mathbf{H}}^1$ be a compressed version of the halting oracle $\mathbf{H}^1$. Suppose that $f(n)$ bits of $\hat{\mathbf{H}}^1$ suffice to specify the first n bits of $\mathbf{H}^1$, that is, that there exists a fixed Turing machine which can compute $\mathbf{H}^1_n$ from $\hat{\mathbf{H}}^1_{f(n)}$ for any n. Assume that

$$\lim_{n\to\infty} \frac{f(n)}{n} = 0.$$

Then there is no computable bound on the amount of time required to extract any single bit of $\mathbf{H}^1$ from $\hat{\mathbf{H}}^1$.

Proof Assume that a block $\mathbf{u}$ of r bits of $\mathbf{H}^1$ ending on the s^{th} bit, i.e. the bits with indices from $s-r+1$ to s, can be extracted from a sufficiently long block $\hat{\mathbf{H}}^1_{f(n)}$ of $\hat{\mathbf{H}}^1$ in time $T(s,r)$, where T is a computable function, i.e., that there exist $\mathbf{M}''$ and T such that

$$(q_0, \lambda, \#, \rho(\mathbf{M}'')[s, r, \hat{\mathbf{H}}^1_{f(n)}]) \longrightarrow_{\mathbf{U}}^{T(s,r)} (q_1, \lambda, \#, \mathbf{u}\}.$$

We observe that this is equivalent to assuming that there is a computable bound on the time required to extract a single bit of $\mathbf{H}^1$ from $\widehat{\mathbf{H}}^1$. The assumption above clearly implies a computable bound on the time required to get one bit: set $r = 1$. If we have a bound on the time required to extract one bit, we can simply compose the Turing machine which meets that bound with itself r times. The resultant machine will run in time boundable by a computable function. (See, e.g. Lewis and Papadimitriou [13] for more details on composition of Turing machines.)

Let

$$\widehat{T}(s,r) = \sum_{1 \le j \le i \le s} T(i,j).$$

This expression has no explicit dependence on r, but $T(s,r)$ is obviously not defined for $r > s$. We define $\widehat{T}(s,r)$ in order to avoid putting requirements on $T(s,r)$. Note that $\widehat{T}$ is monotonic and bounds T above.

Recall the Turing machine $\mathbf{M}$ of theorem I.1. Let $\sigma(n) = n + 1$, and let

$$r(n) = \widehat{T}(2^n, n+1).$$

We will be considering the instances of $\mathbf{M}$ of length $2^k - 1$ of the form $\rho(\mathbf{M})00\cdots0$.

Let $\mathbf{M}'$ be a Turing machine which takes a k-bit string $\mathbf{n}$ as input and

(1) computes the $(\mathbf{n}+1)^{\text{st}}$ bit of the output of the appropriate instance of $\mathbf{M}$;

(2) halts if that bit is a 1;

(3) enters an infinite loop if that bit is a 0.

Note that for all sufficiently large k,

$$|\rho(\mathbf{M}')\mathbf{n}| = |\rho(\mathbf{M}')| + k < 2^k - k - 1,$$

and also

$$|\rho(\mathbf{M}')| + |\rho(\mathbf{M}'')| + 2k + 1 < 2^k.$$

Choose k large enough to satisfy the second inequality above. By the construction of $\mathbf{M}'$, the 2^k bits of $\mathbf{H}^1$ describing the halting behavior of the programs from $\rho(\mathbf{M}')00\cdots0$ to $\rho(\mathbf{M}')11\cdots1$ (the instances of $\mathbf{M}'$ with data of length k) are exactly identical to the string generated by $\mathbf{M}$.

Since $\widehat{T}$ bounds T above, we can use $\mathbf{M}''$ to print these bits in time at most

$$\widehat{T}(2^{|\rho(\mathbf{M}')|+k} + 1, 2^k) < r(2^k).$$

Further, we can use fewer than 2^k bits of input to $\mathbf{U}$ to do this:

- $|\rho(\mathbf{M}'')|$ is constant,
- specifying which bit of $\mathbf{H}^1$ gives the halting information for $\rho(\mathbf{M}')11\cdots1$ requires no more than $|\rho(\mathbf{M}')| + k + 1$ bits (this is the parameter s),
- specifying r requires k bits,
- by assumption, specifying

$$\mathbf{H}^1_{2|\rho(\mathbf{M}')|+k+1-1}$$

  requires fewer than 2^{k-b} bits, for all b,
- encoding the three data parameters requires at most $2(|\rho(\mathbf{M}')| + k + 1 + k + |\hat{\mathbf{H}}^1|) + 4$ bits, and we are done, for the sum of all these lengths

$$|\rho(\mathbf{M}'')| + 2|\rho(\mathbf{M}')| + 4k + 6 + 2^{k-b}$$

  can be made less than 2^k for some b.

It might appear that we have proved a stronger result than originally stated, viz., that there is some maximal fraction 2^{-b} limiting the degree of compression. The problem is that as T gets larger, τ gets larger, and as τ gets larger, $|\rho(\mathbf{M})|$ grows, and thus we cannot bound $|\rho(\mathbf{M}')|$ absolutely. ∎

Conclusions and Directions for Future Research

Randomness is a subtle and slippery idea. This thesis has tried to enhance our understanding of this important concept by exploring algorithmic notions of complexity.

The results on time bounded Kolmogorov–Chaitin complexity touch on the difficulty of discovering patterns. It is not always possible to find structure with a reasonable amount of effort.

In the last two chapters, we have illustrated the enormous power of halting oracles. We have made repeated use of the idea of reducing a computation to a yes or no answer, having a Turing machine halt or not halt depending on that answer, and then extracting the answer—without having to go through potentially prodigious amounts of computation—from a halting oracle. It is this powerful property of halting oracles which makes them so hard to compress—either in space, as with the hierarchy of halting oracles (Chapter 2), or in time, as with the first order halting oracle (Chapter 3).

There is a large literature on relativized computability, and there is a well understood hierarchy of recursive degrees. It would be interesting to characterize the Kolmogorov–Chaitin complexity of arbitrary oracles in the lattice of Turing degrees in a general manner. The degrees which are weaker than the halting oracle but not recursive are especially interesting.

Appendix

Some of the results presented in this thesis are not original. The final theorem was originally called a result on *general recursive majorants of complexity*. It was first proved by Barzdin' [2], and was presented again, with a different proof, by Zvonkin and Levin [18]. These articles were unknown to the author until L. Levin [12] mentioned the references.

Theorem 1.1 is, according to Levin, not at all deep. But there is some value in the proof given, in that it provides an easy way to prove theorem 1.2.

The results on algorithmic information of halting oracles are, so far as the author knows, original.

References

[1] Abu-Mostafa, Y. S., *Information and Complexity,* in preparation.

[2] Barzdin', Ja. M., "Complexity of Programs to Determine Whether Natural Numbers Not Greater Than n Belong to a Recursively Enumerable Set," *Soviet Math Doklady* **9:5** (1968), pp. 1251–1254.

[3] Chaitin, G. J., "Information-Theoretic Limitations of Formal Systems," *Journal of the ACM* **21** (1974), pp. 403–424.

[4] Chaitin, G. J., "A Theory of Program Size Formally Identical to Information Theory," *Journal of the ACM* **22** (1975), pp. 329–340.

[5] Davis, M. D. and E. J. Weyuker, *Computability, Complexity, and Languages,* New York: Academic Press, 1983.

[6] Deutsch, D. "Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer," *Procedings of the Royal Society of London A* **400** (1985), pp. 97–117.

[7] Erber, T. and S. Putterman, "Randomness in quantum mechanics—nature's ultimate cryptogram?," *Nature* **318** (7 November 1985), pp. 41–43.

[8] Gardner, M., "Mathematical Games," *Scientific American* **241** (November 1979), pp. 20–34.

[9] Hopcroft, J. E. and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Reading, Massachusetts: Addison-Wesley, 1979.

[10] Knuth, D. E., "Big Omicron and Big Omega and Big Theta," *ACM SIGACT News* (April–June 1976), pp. 18–24.

[11] Kolmogorov, A. N., "Three Approaches to the Quantitative Definition of Information," *Problemy Peredachi Informatsii* **1** (1965), pp. 3–11.

[12] Levin, L., private communication.

[13] Lewis, H. R. and C. H. Papadimitriou, *Elements of the Theory of Computation*, Englewood Cliffs, New Jersey: Prentice–Hall, 1981.

[14] Rogers, H., *Theory of Recursive Functions and Effective Computability*, New York: McGraw-Hill, 1967.

[15] Shannon, C. E. and W. Weaver, *The Mathematical Theory of Communication*, Urbana: University of Illinois Press, 1963.

[16] Turing, A. M., "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, ser. 2 **42** (1936), pp. 230–265, and ser. 2 **43** (1936), pp. 544-546.

[17] Turing, A. M., "Systems of Logic Based on Ordinals," *Proceedings of the London Mathematical Society*, ser. 2, **45** (1939), pp. 161–228.

[18] Zvonkin, A. K. and L. A. Levin, "The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms," *Russian Mathematical Surveys* **25:6** (1970), pp. unknown