AN ABSTRACT OF THE THESIS OF

Charlie Robson for the degree of Doctor of Philosophy in Mathematics presented on June 15, 2017.

Title: Computable Randomness, and Coding the Orbits of the Collatz Map

Abstract approved: _____

Ren Guo

In this thesis I will look at a definition of computable randomness from Algorithmic Information Theory as defined by Andre Nies through the lens of Computable Analaysis as defined by Klaus Weihrauch. I will show that despite the fact that these two paradigms generate distinct classes of computable supermartingales, the class of sets on which no computable supermartingale succeeds of either type is identical. Therefore, both theories generate the same collection of computably random sets. I will then consider how one might apply some of the techniques in Algorithmic Information Theory, including prefix free codes and the Kraft Inequality, to the study of the Collatz Conjecture.

Computable Randomness, and Coding the Orbits of the Collatz Map

by

Charlie Robson

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented June 15, 2017
Commencement June 2018

Doctor of Philosophy thesis of Charlie Robson presented on June 15, 2017

APPROVED:

Major Professor, representing Mathematics

Chair of the Department of Mathematics

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Charlie Robson, Author

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

COMPUTABLE RANDOMNESS, AND CODING THE ORBITS OF THE COLLATZ MAP

1 GENERAL INTRODUCTION

This thesis will consist of a general background introduction and two parts. The first part: "Computable Randomness in Two Notions of Computable Reals", which I will refer to as Part 1, concerns itself with the definition of computable randomness [1], a property of infinite sequences of 0's and 1's thought to capture our intuitive notion of randomness or unpredictability of the sequence in an effective way. This is achieved through what are known as computable supermartingales [1]. I will show that the natural definition for computable supermartingales from the perspective of Computable Analysis [3] gives rise to a larger class of supermartingales which may be considered computable. However, it will then be shown that the set of computably random sequences determined by this larger class remains unchanged. This implies that one may establish computable randomness using the definitions and theorems in [3] and be guaranteed computable randomness as specified in [1].

In the second part: "Coding the Orbits of the Collatz Map", which I will refer to as Part 2, I have attempted to apply some of the concepts from [1] and [10] to the Collatz conjecture. Take any positive integer, if it's even, divide by 2; if it's odd, multiply by 3 and add 1. The Collatz conjecture sates that iterating this procedure on any positive integer eventually leads you to 1. I first study the total stopping times (how many iterations it takes to reach 1) by coding the decisions at each stage into binary, converting them into what are called prefix-free codes [10], and then apply the Kraft inequality [1, 10]. The Kraft inequality then provides us with an indication of the possible behavior of the total stopping times. Next I will discuss how computable randomness might be applied to the codes developed, which requires a generalization of the original definition. Part 1 and 2 can be thought of thematically as theory and applications of references [1] and [10] respectively.

I continue this introduction with a summary of the concepts utilized in Part 1 and 2, including references where detailed treatments seem impractical.

1.1 Background for "Computable Randomness in Two Notions of Computable Reals"

1.1.1 Turing Machine Overview, computability in Σ^* and Σ^ω

We will follow the computability notions as prescribed in [2] and [3]. I have provided a precise description of Turing machines in the Appendix for reference by adapting the flowchart scheme from [2] to the requirements of [3], but only the informal definition that follows will be required to follow the arguments of this thesis. Let Σ be a finite set with at least two elements (usually assumed to be $\{0,1\}$, see section "A Note on Alphabets") and Σ^* be the set of all finite strings of symbols in Σ, including the empty string λ. Also let Σ^ω be the set of all one way infinite sequences of symbols in Σ.

We will consider Turing machines as in [3] of the form $f_M :\subseteq Y_1 \times ... \times Y_n \to Y_0$ where $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ and "$\subseteq Y_1 \times ... \times Y_n$" indicates that the Turing machine may only be defined on a subset of $Y_1 \times ... \times Y_n$. In particular, our Turing machines will have $n \geq 0$ read only input tapes, $m \geq 0$ two-way read-write work tapes, and a single write-only output tape on which the Turing machine may only move it's head to the right.

Let $B \notin \Sigma$ be the "blank symbol". When one of the arguments is of the form $w \in \Sigma^*$, then the Turing machine's corresponding input tape contains the sequence: $BB...BBw(0)w(1)...w(|w|-1)BB...$ with the head initially at the cell containing $w(0)$. Where $w(i)$, $i \geq 0$, indicates the i^{th} symbol in string w, and $|w|$ denotes the length of w. When the argument is of the form $p \in \Sigma^\omega$, then the Turing machine's corresponding input tape contains the sequence $BB...BBp(0)p(1)p(2)...$ with the head initially located on the cell containing $p(0)$. All other tapes are initialized to contain only the blank symbol $...BBB....$

When the range set is Σ^* the Turing machine halts according to it's standard internal halting condition and returns the left most maximal length word on the output tape not

containing the blank symbol. When the range set is Σ^ω, the Turing machine returns output $q \in \Sigma^\omega$ only if for all $n \in \mathbb{N}$, $BBq(0)q(1)...q(n)BB...$ eventually appears on the output tape. That is the Turing machine makes progress printing q on the output indefinitely. If the halting condition is not reached or if the calculation does not produce an infinite sequence then it is said to diverge, denoted $\uparrow$. I will also sometimes use $\uparrow$, more generally, to indicate that a partial function is undefined on some input.

A function $f :\subseteq Y_1 \times ... \times Y_n \to Y_0$ is said to be *computable* if there exists a Turing machine $f_M :\subseteq Y_1 \times ... \times Y_n \to Y_0$ such that $f = f_M$. The symbol M in f_M may be given a precise definition in terms of the flowchart paradigm as presented in the Appendix, but in practice I will use M to refer the intuitive algorithm that justifies the existence of f_M, which follows the style of justification in part 3 of [2], and [3].

Some examples of computable functions $f :\subseteq Y_1 \times ... \times Y_n \to Y_0$ such that $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ are the *wrapping function* and the *tupling functions* [3].

Definition 1.1. Define the *wrapping function* $\iota : \Sigma^* \to \Sigma^*$ by $\iota(a_1a_2...a_n) = 110a_10a_20...a_n011$ for all $n \in \mathbb{N}$, $a_i \in \Sigma$.

For $x, x_0, x_1, ... \in \Sigma^*$, $p, p_0, p_1, ... \in \Sigma^\omega$ define:

(1) $< x_1, x_2, ..., x_k >:= \iota(x_1)\iota(x_2)...\iota(x_k) \in \Sigma^*$

(2) $< x, p >:= \iota(x)p \in \Sigma^\omega$

(3) $< p, x >:= \iota(x)p \in \Sigma^\omega$

(4) $< p_1, p_2, ..., p_k >:= p_1(0)...p_k(0)p_1(1)...p_k(1)... \in \Sigma^\omega$.

The instructions for the machine which computes the wrapping function for $\iota(w)$ simply requires that we write 11 on the output, then alternate writing 0's and copying the bits of w to the output until the blank at the end of w is reached, then writes 011 and halts. Since this can be done for any w, ι is called total computable.

Informal descriptions of algorithms such as the one given above for ι are considered sufficient for demonstrating the existence of a Turing machine which calculates the function. The arguments can always be reduced to the formalism in part 1 of [2]; in particular, by utilizing the precise Turing machine model as provided in the Appendix. Readers seeking justification for of the validity of such informal arguments are usually referred to Church's Thesis [1, 2, 3, 4, 5, 7, 10]. For an in depth discussion on the history of the development of informal arguments in computability see Soare, "Formalism and Intuition in Computability" [6].

Convention 1.1. In the following I will box text for machine descriptions, which define the Turing machine, and use the symbol $\Diamond$ to indicate that the description has ended. The symbol $\square$ will indicate the end of a proof.

By a similar argument to that for the wrapping function we can also show that $<,>$: $\Sigma^* \times \Sigma^\omega \to \Sigma^\omega$ is computable. Using our new conventions:

Proof.

Let M be the machine that on input (x, p) computes $\iota(x)$ as a subroutine and copies it to the output, and then begins copying p from its input tape to the output. $\Diamond$

Thus, $<,>: \Sigma^* \times \Sigma^\omega \to \Sigma^\omega$ is computed by f_M.

$\square$

The key elements of this argument are that since we know $\iota : \Sigma^* \to \Sigma^*$ is computable on all of Σ^*, it can be called as a subroutine in the computation of $<,>$, and furthermore since its range is Σ^* we know when it converges (as it always does) it does so in finite time. Thus there's no danger of the infinite computation of $< x, p >$ getting "hung-up" in $\iota(x)$'s calculation. Thus, we're guaranteed that the computation will make progress outputting the bits of $\iota(x)p(0)p(1)...$ forever; which is the requirement of computable functions with range Σ^w.

1.1.2 Transferring Computability to $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, ...

In order to transfer the notion of computability to functions beyond those on Σ^* and Σ^ω in [2] and [3], naming systems are introduced. A *naming system* for a set X is simply an onto map $\gamma : \Sigma^a \to X$ where $a \in \{*, \omega\}$. If $\gamma(w) = x$, then w is called a name of x. When $a = *$, γ is sometimes reffed to as a *notation*. When $a = \omega$, γ is sometimes reffed to as a *representation*.

Definition 1.2. Let $\gamma_1 :\subseteq Y_1 \to X_1$ and $\gamma_0 :\subseteq Y_0 \to X_0$ where $Y_0, Y_1 \in \{\Sigma^*, \Sigma^\omega\}$ and X_0, X_1 are sets. Then $f :\subseteq X_1 \to X_0$ is said to be (γ_1, γ_0)-*computable* iff there is a computable function $g :\subseteq Y_1 \to Y_0$ such that $f(\gamma_1(p)) = \gamma_0(g(p))$ for all $p \in Y_1$ such that $p \in dom(f \circ \gamma_1)$.

This means intuitively that there is a Turing machine (in this case g) that can coordinate with the naming systems to emulate $f :\subseteq X_1 \to X_0$ in the spaces Σ^* and Σ^ω. I will also use the notation $f(\gamma_1(p)) \downarrow$ to emphasize that $p \in dom(f \circ \gamma_1)$ or that a calculation converges.

The following are some examples of naming systems which will be used frequently in the Part 1:

Example 1.1.

(1) *The identity*: $id_{\Sigma^*} : \Sigma^* \to \Sigma^*$ such that $id_{\Sigma^*}(w) := w$.

(2) *The binary notation*: $bin :\subseteq \Sigma^* \to \mathbb{N}$, such that $dom(bin) = \{0\} \cup 1\{0,1\}^*$ and $bin(a_k...a_0) := \sum_{i=0}^{k} a_i 2^i$, and $a_i \in \{0,1\}$, $1 \leq i \leq k$.

(3) *The standard notation of* $\mathbb{N}$: $num : \Sigma^* \to \mathbb{N}$ such that $num(w) = n$ iff $bin^{-1}(n+1) = 1w$. (See [2] for extending num to larger alphabets, in [2] num is referred to as ν_Σ).

(4) *The standard notation of* $\mathbb{Z}$: $\nu_\mathbb{Z} :\subseteq \Sigma^* \to \mathbb{Z}$ such that $dom(\nu_\mathbb{Z}) = \{0\} \cup 1\{0,1\}^* \cup -1\{0,1\}^*$ such that $\nu_\mathbb{Z}(w) = bin(w)$ and $\nu_\mathbb{Z}(-w) = -bin(w)$ for $w \in dom(bin)$. Note that Σ has been expanded to the alphabet $\{-1, 0, 1\}$ for this definition, and thus our

Turing machine definitions must be adapted to allow another non-blank symbol. See the section "A Note on Alphabets" and [2] for details on Turing machines with arbitrary finite alphabets.

(5) *The standard notation of* $\mathbb{Q}$: $\nu_{\mathbb{Q}} :\subseteq \Sigma^* \to \mathbb{Q}$ such that $dom(\nu_{\mathbb{Q}}) := \{u/v | u \in dom(\nu_{\mathbb{Z}}), v \in dom(bin), \text{and } bin(v) \neq 0\}$, and $\nu_{\mathbb{Q}}(u/v) := \frac{\nu_{\mathbb{Z}}(u)}{bin(v)}$.

(6) *The standard representation of* $\mathbb{R}$ or simply *The standard representation*: $\rho :\subseteq \Sigma^\omega \to \mathbb{R}$ such that $\rho(p) := x$ if and only if $x \in B_{\nu_{\mathbb{Q}}(b)}(\nu_{\mathbb{Q}}(a)) \iff \iota(\iota(a)\iota(b))\triangle p$ for all $a, b \in dom(\nu_{\mathbb{Q}})$. Where $B_r(y)$ is the interval in $\mathbb{R}$ centered at y with radius r, and $w\triangle p$ means w is a subword of p (i.e. $p = uwq$ for some $u \in \Sigma^*, q \in \Sigma^\omega$).

The standard notation, example 3, is a bijection often used implicitly in computability with the fact from [2] that $g :\subseteq \Sigma^* \to \Sigma^*$ is computable iff $f := num \circ g \circ num^{-1} :\subseteq \mathbb{N} \to \mathbb{N}$ is (num, num)-computable. This gives a direct correspondence between computable word and number functions. And since each can be emulated by the other, algorithms are ofter considered that work with a mixture of numbers and bit strings [2].

Convention 1.2. By the above correspondence a function $f :\subseteq \mathbb{N} \to \mathbb{N}$ will often simply be called computable if it is (num, num)-computable. The num correspondence can also be defined for larger alphabets than $\{0, 1\}$ as in [2] with identical computability results.

Example 1.2. $f(n) = 2n$ is computable.

Proof. We need to show there exists a computable $g : \Sigma^* \to \Sigma^*$ such that $f(num(w)) = num(g(w))$ for all $w \in \Sigma^*$.

Let M be the machine that on input w:

(i) appends a 1 on the left of w to get $1w$, then

(ii) subtracts 1 in binary from $1w$ to get aw_1 for some $a \in \Sigma$ and $w_1 \in \Sigma^*$, then

(iii) appends a 0 on the right to get $aw_1 0$, and

(iv) returns $w_1 0$. $\Diamond$

Then $f(num(w)) = num(f_M(w))$ for all $w \in \Sigma^*$.

□

An example of M's function is $00 \xrightarrow{\text{(i)}} 100 \xrightarrow{\text{(ii)}} 11 \xrightarrow{\text{(iii)}} 110 \xrightarrow{\text{(iv)}} 11$. Since $num(00) = 3$ and $num(11) = 6$, we have $f(num(00)) = 2num(00) = 2 \times 3 = 6 = num(11) = num(f_M(00))$.

The naming system (6) is the one used to define computable real functions in [3]. It says $p \in \Sigma^\omega$ is a name for $x \in \mathbb{R}$ if p contains the code for every rational interval (as a subword) that contains x. Extensive arguments are provided in [3] suggesting that this is the correct naming system for $\mathbb{R}$, especially for investigating computability in Analysis.

Definition 1.3. Two naming systems $\gamma_1 :\subseteq Y \to X$ and $\gamma_2 :\subseteq Y' \to X$ such that Y and Y' are some finite product of Σ^* or Σ^ω, are *equivalent*, denoted $\gamma_1 \equiv \gamma_2$, if there exist computable functions $f :\subseteq Y \to Y'$ and $g :\subseteq Y' \to Y$ such that $\gamma_1(p) = \gamma_2(f(p)), \forall p \in dom(\gamma_1)$ and $\gamma_2(p) = \gamma_1(g(p)), \forall p \in dom(\gamma_2)$.

Intuitively this means two naming systems are equivalent if there exists fixed algorithms that allows us to emulate each naming system with the other. Corollary 3.1.9 in [3] states that naming systems γ_1 and γ_2 above induce the same computability concepts on X iff they are equivalent.

Example 1.3. *bin* and *num* are equivalent.

Proof.

Let Q be the machine that on input $w \in dom(bin)$ adds 1 in binary to get aw_1 such that $a \in \Sigma$ and $w_1 \in \Sigma^*$, and then outputs w_1. $\Diamond$

Then $bin(w) = num(f_Q(w))$ for all $w \in dom(bin)$.

Let N be the machine that on input w subtracts 1 in binary from $1w$ and returns the result. $\Diamond$

Then $num(w) = bin(f_N(w))$ for all $w \in \Sigma^*$. Therefore, $num \equiv bin$.

□

This tells us that $f(n) = 2n$ is also (bin, bin)-computable since using f_M from the proof that f is (num, num)-computable we have

$$\begin{aligned} f(bin(w)) &= f(num(f_Q(w)) \\ &= num(f_M(f_Q(w)) \\ &= bin(f_N(f_M(f_Q(w)))). \end{aligned}$$

Since $(f_N \circ f_M \circ f_Q)(w) \downarrow$ for all $w \in dom(bin)$ is computable, f is (bin, bin)-computable (where recall $\downarrow$ means the function is defined on its input). This example indicates how equivalent naming systems induce the same computability results.

The following useful naming systems are equivalent to the representation ρ, naming system (6) in our example, and thus may be used interchangeably:

Example 1.4.

(8) *The Cauchy representation*: $\rho_C :\subseteq \Sigma^\omega \to \mathbb{R}$ such that $\rho_C(p) := x$ if and only if there are words $w_0, w_1, ... \in dom(\nu_\mathbb{Q})$ such that $p = \iota(w_0)\iota(w_1)...$ and $|\nu_\mathbb{Q}(w_i) - \nu_\mathbb{Q}(w_k)| \leq \frac{1}{2^i}$ for $i < k$ and $x = \lim_{i\to\infty} \nu_\mathbb{Q}(w_i)$.

(9) ρ'_C, ρ''_C, and ρ'''_C are obtained from ρ_C by substituting $|\nu_\mathbb{Q}(w_i) - \nu_\mathbb{Q}(w_k)| < \frac{1}{2^i}$, $|\nu_\mathbb{Q}(w_i) - x| < \frac{1}{2^i}$, and $|\nu_\mathbb{Q}(w_i) - x| \leq \frac{1}{2^i}$ for $|\nu_\mathbb{Q}(w_i) - \nu_\mathbb{Q}(w_k)| \leq \frac{1}{2^i}$ in the definition of ρ_C.

(10) $(\rho_< \wedge \rho_>)(p) = x$ iff $p =< p_1, p_2 >$ such that $\forall w \in \Sigma^*$, $\nu_\mathbb{Q}(w) < x \iff \iota(w) \triangle p_1$, and $\nu_\mathbb{Q}(w) > x \iff \iota(w) \triangle p_2$.

(11) $\rho_{(a,b)}(p) = x$ iff $\forall a, b \in \mathbb{Q},\ x \in (a,b) \iff \iota(\iota(a)\iota(b))\triangle p$.

The equivalence of (6)-(10) are demonstrated in [3]. The naming system in (11) is another useful representation which is equivalent to (6), this will be proved at the end of this section.

For functions of several variables $f :\subseteq X_1 \times ... \times X_k \to X_0$ with naming systems $\gamma_i :\subseteq Y_i \to X_i$, such that $Y_i \in \{\Sigma^*, \Sigma^\omega\}$, $0 \leq i \leq k$, we can define the computability of f induced by the naming systems $((\gamma_1, ..., \gamma_k), \gamma_0)$ by: f is $((\gamma_1, ..., \gamma_k), \gamma_0)$-*computable* if there exists a computable $g :\subseteq Y_1 \times ... \times Y_k \to Y_0$ such that $f(\gamma_1(y_1), ..., \gamma_k(y_k)) = \gamma_0(g(y_1, ..., y_k))$ for all $(y_1, ..., y_k) \in Y_1 \times ... \times Y_k$ such that $f(\gamma_1(y_1), ..., \gamma_k(y_k)) \downarrow$ [3].

This amounts to simply replacing the naming system γ_1 of Y_1 in the one-dimensional case with the naming system $\gamma = \gamma_1 \times ... \times \gamma_k$ of $Y = Y_1 \times ... \times Y_k$.

Example 1.5. The *Cantor pairing function*:

$$\begin{aligned} < x, y > :&= \sum_{k=0}^{x+y} k + y \\ &= \frac{(x+y)(x+y+1)}{2} + y \end{aligned}$$

is a commonly used bijection between $\mathbb{N}^2$ and $\mathbb{N}$. It is derived by counting the steps of Cantor's diagonal path through the ordered pairs of nonnegative integers. Cantor's pairing function is $((num, num), num)$-computable (or by our convention above, may be simply called computable).

Proof. Since 2 divides either $x + y$ or $x + y + 1$ we're guaranteed that we can perform the calculation on the right in binary. Thus, $< x, y >$ is $((bin, bin), bin)$-computable and thus $((num, num), num)$-computable as well.

□

The inverse functions $L_C(< x, y >) := x$ and $R_C(< x, y >) := y$ of the Cantor pairing function are also computable [2, 8]. With this fact it is easy to prove that naming system (11) is equivalent to (6)-(10) as mentioned above.

Theorem 1.1. $\rho \equiv \rho_{(a,b)}$.

Proof. We'll show $\rho_{(a,b)} \equiv (\rho_< \wedge \rho_>)$.

Suppose $(\rho_< \wedge \rho_>)(p) = x$. Then by definition $p =< p_1, p_2 >$ where $\forall w \in \Sigma^*$, $\nu_{\mathbb{Q}}(w) < x \iff \iota(w) \triangle p_1$, and $\nu_{\mathbb{Q}}(w) > x \iff \iota(w) \triangle p_2$. Recall that $p =< p_1, p_2 >= p_1(0)p_2(0)p_1(1)p_2(1)....$

> Thus we let M be the machine that on input p operates in stages n. At stage n it scans the even bits of p for the $L_C(n)^{th}$ ι-word say $\iota(a_{L_C(n)})$ and the odd bits of p for the $R_C(n)^{th}$ ι-word say $\iota(b_{R_C(n)})$ and prints $\iota(\iota(a_{L_C(n)})\iota(b_{R_C(n)}))$ to the output. $\Diamond$

Then $(\rho_< \wedge \rho_>)(p) = \rho_{(a,b)}(f_M(p)) \forall p \in dom(\rho_< \wedge \rho_>)$.

Conversely, suppose $\rho_{(a,b)}(p) = x$.

> Let N be the machine with two work tapes that on input p operates in stages. At stage n it scans the n^{th} ι-word of p say $\iota(\iota(a_n)\iota(b_n))$ and prints $\iota(a_n)$ to work tape 1 and $\iota(b_n)$ to work tape 2. N then alternates copying and deleting symbols from work tape 1 to the even indexes of the output and the symbols from work tape 2 to the odd indices of the output. When either work tape 1 or work tape 2 is exhausted N proceeds to stage $n + 1$. $\Diamond$

(Note that the work tapes are necessary since the output head "can't go back".) Thus, $\rho_{(a,b)}(p) = (\rho_< \wedge \rho_>)(f_N(p))$, $\forall p \in dom(\rho_{(a,b)})$.

So, $\rho_{(a,b)} \equiv (\rho_< \wedge \rho_>)$.

□

1.1.3 Transferring Computability of Sets

Definition 1.4. [3] Let $X \subseteq Z \subseteq Y = Y_1 \times ... \times Y_k$, $k \geq 1$, and $Y_i \in \{\Sigma^*, \Sigma^\omega\}$.

(1) X is called *recursively enumerable open* (*r.e. open* or just *r.e.*) in Z iff $X = dom(f) \cap Z$ for some computable $f :\subseteq Y \to \Sigma^*$.

(2) X is called *decidable* in Z iff X and $Z \setminus X$ are r.e. open in Z. If $Z = Y$ we omit "in Z".

Example 1.6.

(a) Since ι is computable, $dom(\iota) = \Sigma^*$ is r.e. open. Furthermore, since there is a Turing machine $f :\subseteq \Sigma^* \to \Sigma^*$ that diverges on every input, $\emptyset$ is r.e. open as well. So, Σ^* is decidable.

(b) Similarly there is a machine that copies $p \in \Sigma^\omega$ to its output forever. Thus, Σ^ω is r.e. open. And the machine that diverges for all $p \in \Sigma^\omega$ gives us $\emptyset$ is r.e. open in Σ^ω. Thus, Σ^ω is decidable too.

Example 1.7. A standard result in classical computability theory [4] is that every infinite r.e. set $A \subseteq \Sigma^*$ is the range of some total computable 1-1 function (this result is used in Part 1).

Proof. Suppose $A \subseteq \Sigma^*$ is r.e. and infinite. Then by definition there exists a $g :\subseteq \Sigma^* \to \Sigma^*$ computable such that $dom(g) = A$.

Let M be a machine with 2 worktapes 1 and 2 reserved for the following process: on input $w \in \Sigma^*$ M computes

(0) $g_{\boxed{0}}(\lambda)$

(1) $g_{\boxed{1}}(\lambda)$, $g_{\boxed{1}}(0)$

(2) $g_{\boxed{2}}(\lambda)$, $g_{\boxed{2}}(0)$, $g_{\boxed{2}}(1)$

$\vdots$

(n) $g_{\boxed{n}}(\lambda)$, $g_{\boxed{n}}(0)$, $g_{\boxed{n}}(1)$, ..., $g_{\boxed{n}}(string(n))$

$\vdots$

Where $g_{\boxed{i}}(u)$ is the total computable function that calculates g for i steps: returning 1 if g halts in this time, and 0 otherwise.

Whenever a calculation of $g_{\boxed{i}}(u)$ in the list above halts, M checks whether $\iota(u)$ is a substring of work tape 1. If it is, then it proceeds to the next partial calculation of g in the list.

Otherwise, if $\iota(u)$ is not a substring of worktape 1, then M appends $\iota(u)$ to worktape 2. And checks the following two cases:

If worktape 2 contains $num(w)$ 1's, then M halts and returns u.

If worktape 2 does not contain $num(w)$ 1's, then M proceeds to the next partial calculation of g. $\Diamond$

Since $dom(g)$ is infinite the $num(w)^{th}$ halting calculation for g can always be reached, so f_M is total computable. Furthermore, the list of unique words in worktape 1 ensures that f_M is 1-1. Since every possible input and time are considered for input w sufficiently large, f_M is onto A. Thus, f_M is 1-1, total computable, and onto A.

$\square$

A more direct but less descriptive way to implement M would be to utilize the Cantor Pairing function:

Let M be the machine that on input w reserves two worktapes 1 and 2 for the following process: at stage n, M computes:

$$g_{\boxed{L_C(n)}}(string(R_C(n))).$$

If $g_{\boxed{L_C(n)}}(string(R_C(n)))$ reaches g's halting condition in $\leq L_C(n)$ steps and $\iota(string(R_C(n)))$ is not listed on worktape 1, append it to worktape 1. Then on worktape 2 append a 1.

If worktape 2 contains num(w) 1's, halt and return $string(R_C(n))$.

Otherwise proceed to stage $n+1$. $\Diamond$

The process of computing by alternatively executing on only finitely many steps of several algorithms, as in $g_{\boxed{i}}(u)$ above is a standard technique in computability theory reffered to as *dove tailing* [4].

A useful equivalence to the definition of a decidable set is the following theorem also from [3]:

Theorem 1.2. X is decidable in Z iff there is a computable function $h :\subseteq Y \to \Sigma^*$ such that $h(y) = 1$ if $y \in X$ and $h(y) = 0$ if $y \in Z \setminus X$, for all $z \in Z$.

Proof. Suppose X is decidable in Z. Then X and $Z \setminus X$ are r.e. open in Z. I.e. there exists $f, g :\subseteq Y \to \Sigma^*$ such that $X = dom(f) \cap Z$ and $Z \setminus X = dom(g) \cap Z$.

Let M be the machine that on input $y \in Y$ computes:

$f_{\boxed{0}}(y)$,$g_{\boxed{0}}(y)$, $f_{\boxed{1}}(y)$, $g_{\boxed{1}}(y)$...

If f halts first return 1.

If g halts first return 0.

Thus, f_M is a computable function which can decide membership of X in Z.

Conversely, suppose there is a computable function $h :\subseteq Y \to \Sigma^*$ such that $h(y) = 1$ if $y \in X$ and $h(y) = 0$ if $y \in Z \setminus X$, for all $z \in Z$.

Let M be the machine that on input $y \in Y$ computes $h(y)$. If $h(y) = 1$ return 1, otherwise diverge the calculation. $\Diamond$

Thus, $dom(f_M) \cap Z = X$ so X is r.e. open. Finally:

Let N be the machine that on input $y \in Y$ computes $h(y)$. If $h(y) = 0$ return 1, otherwise diverge the calculation. $\Diamond$

Then $dom(f_N) \cap Z = Z \setminus X$. So, $Z \setminus X$ is r.e. open. Since both X and $Z \setminus X$ are r.e. open, X is decidable in Z.

□

We can transfer our computability notions of sets of strings to other sets using our naming systems:

Definition 1.5. [3] Let $\gamma :\subseteq Y \to X$, be a naming system of X where $Y = Y_1 \times ... \times Y_k$ such that $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ and $k \geq 1$. A set $Z \subseteq X$ is called γ-r.e. (γ-decidable) iff $\gamma^{-1}(Z)$ is r.e. (decidable) in $dom(\gamma)$.

Using this definition we define a subset A of $\mathbb{N}$ as being r.e. (decidable) iff A is num-r.e. (num-decidable).

Theorem 1.3. A set $A \subseteq \mathbb{N}$ is decidable iff there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $f = \chi_A$, the characteristic function of A.

Proof. Suppose $A \subseteq \mathbb{N}$ is decidable. Then by definition A is num-decidable. Thus there exists a computable function $g : \Sigma^* \to \Sigma^*$ such that $g(w) = 1$ if $w \in num^{-1}(A)$ and $g(w) = 0$ if $w \notin num^{-1}(A)$, for all $w \in \Sigma^*$.

Let M be the machine that on input w computes $g(w)$. If $g(w) = 1$, return 0; and if $g(w) = 0$, return λ. $\Diamond$

Then $f_M(w) = \begin{cases} 0, & w \in num^{-1}(A) \\ \lambda, & w \notin num^{-1}(A) \end{cases}$.

Hence, $num(f_M(w)) = \begin{cases} 1, & w \in num^{-1}(A) \\ 0, & w \notin num^{-1}(A) \end{cases}$, and so

$num(f_M(num^{-1}(n))) = \begin{cases} 1, & n \in A \\ 0, & n \notin A \end{cases} = \chi_A.$

Conversely, if $f = \chi_A$ for some computable $f : \mathbb{N} \to \mathbb{N}$ then there exists an $h :\subseteq \Sigma^* \to \Sigma^*$ computable such that $f = num \circ h \circ num^{-1}$, i.e. $h = num^{-1} \circ f \circ num$.

So, $h(w) = num^{-1} \circ f \circ num(w) = \begin{cases} 0, & w \in num^{-1}(A) \\ \lambda, & w \notin num^{-1}(A) \end{cases}$.

Let N be the machine that on input w computes $h(w)$. If $h(w) = 0$, return 1; and if $h(w) = \lambda$, return 0. $\Diamond$

Thus, $f_N(w) = \begin{cases} 1, & w \in num^{-1}(A) \\ 0, & w \notin num^{-1}(A) \end{cases}$. So, $num^{-1}(A)$ is decidable by f_N. Which means A is num-decidable or simply decidable.

□

An important standard example of a r.e. set in $\mathbb{N}$ that will be used in Part 1 is the halting set $K = \{i \in \mathbb{N}$ such that the i^{th} Turing machine (in an effective list of all Turing machines) halts on input i $\}$.

To discuss the halting set we need some standard facts from classical computability. See [2, 7, 8, 9] that:

Theorem 1.4.

(1) The Turing machines can be listed by virtue of their finite instruction sets: $\Phi_0, \Phi_1, ...$ (This is often achieved by effectively coding their instruction sets into the integers [2, 9]).

(2) There is a single Turing machine u called the *universal machine* which can compute all the other Turing machines by accepting these indexes as an extra input: $u(i, n) = \Phi_i(n)$. u is constructed in full rigor in [2, 8, 9] (this is achieved by a subroutine which decodes the program indices from (1) above).

(3) Turing machines can effectively pass arguments to their index: for all $\Phi_a \subseteq \mathbb{N}^k \to \mathbb{N}$ there is a total Turing machine $r : \mathbb{N}^{j+1} \to \mathbb{N}$ such that $1 \leq j \leq k$, $\Phi_a(x_1, ..., x_k) = \Phi_{r(a,x_1,...,x_j)}(x_{j+1}, ..., x_k)$ [9]. This is called the *parameter theorem.*

Where the Turing machine $\Phi_i :\subseteq \mathbb{N}^k \to \mathbb{N}$ is defined in terms of the corresponding Turing machine $\Phi'_i :\subseteq (\Sigma^*)^k \to \Sigma^*$ by $\Phi_i = num \circ \Phi'_i \circ (num^{-1}, ..., num^{-1})$ and $u :\subseteq \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined in terms if $u' :\subseteq \Sigma^* \times \Sigma^* \to \Sigma^*$ by $u(i, n) := num(u'(num^{-1}(i), num^{-1}(n))$.

Utilizing this notation the halting set can be written:

Definition 1.6. $K := \{x | \Phi_x(x) \downarrow\}$.

Theorem 1.5. K is r.e., but not decidable [2, 4, 5].

Proof. That K is r.e. is clear since it is the domain of the computable function $f(x) = u(x,x)$.

Suppose that K is also decidable for a contradiction.

Then there exists a Turing machine Φ_a such that $\Phi_a = \chi_K$.

> Let M be the machine that on input x computes $\Phi_a(x)$.
>
> If $\Phi_a(x) = 1$, return $\Phi_x(x) + 1$
>
> otherwise return 0. $\Diamond$

Then f_M is total computable. Since if $x \in K$, then $\Phi_x(x) \downarrow$ so $f_M(x) = \Phi_x(x) + 1 \downarrow$; and if $x \notin K$, then $f_M(x) = 0$ so f_M is total.

Since the Turing machines can be listed, there exists a $b \in \mathbb{N}$ such that $f_M = \Phi_b$. I.e.

$$\Phi_b(x) = \begin{cases} \Phi_x(x) + 1, & \Phi_x(x) \downarrow \\ 0, & \text{o.w.} \end{cases}.$$

Since Φ_b is total however, $\Phi_b(b) \downarrow = \Phi_b(b)+1$, a contradiction. Thus, K cannot be decidable.

□

1.1.4 A Note on Alphabets

Some times it is conceptually easier to use larger alphabets, for example when we coded the rationals with the notation $\nu_{\mathbb{Q}} \subseteq \{0,1,/,-1\}^* \to \mathbb{Q}$, but analytically it is easier if we assume that our alphabets are always $\{0,1\}$. [3] assures us that we may do this, as a result of the following:

Definition 1.7. For $\Sigma = \{a_1, ..., a_k\}, n \geq 2$ define $c : \Sigma \to \{0,1\}^*$ by

$$c(a_i) := \begin{cases} 1^{i-1}0, & i < n \\ 1^{n-1}, & i = n \end{cases}, \text{ and}$$

$c_* : \Sigma^* \to \{0,1\}^*$ by $c_*(b_1 b_2 ... b_k) = c_\omega(b_0)c_\omega(b_1)....$

Then:

Theorem 1.6. c_* is 1-1, and c_ω is bijective. Furthermore, $f :\subseteq \Sigma^{d_1} \times ... \times \Sigma^{d_k} \to \Sigma^{d_0}$, $d_i \in \{*, \omega\}$ is computable iff $c_{d_0} \circ f(c_{d_1}^{-1}, ..., c_{d_k}^{-1}) :\subseteq \{0,1\}^{d_1} \times ... \times \{0,1\}^{d_k} \to \{0,1\}^{d_0}$ is computable.

Thus there is a 1-1 correspondence between computable word functions on Σ^* and Σ^ω for arbitrary alphabets, and on $\{0,1\}^*$ and $\{0,1\}^\omega$.

The computability induced by naming systems is also uneffected by the choice of alphabet.

Theorem 1.7. Let $\gamma :\subseteq \Sigma^{d_1} \to X$ and $\gamma_0 :\subseteq \Sigma^{d_2} \to Y$ be naming systems. Then f is (γ, γ_0)-computable iff f is $(\gamma \circ c_{d_1}^{-1}, \gamma \circ c_{d_2}^{-1})$-computable.

Proof. First, since c_{d_1} and c_{d_2} are total, $\gamma \circ c_{d_1}^{-1} :\subseteq \{0,1\}^{d_1} \to X$ and $\gamma \circ c_{d_2}^{-1} :\subseteq \{0,1\}^{d_2} \to Y$ are onto, and thus naming systems.

Suppose f is (γ, γ_0)-computable. Then there is a $g :\subseteq \Sigma^{d_1} \to \Sigma^{d_2}$ computable such that $f(\gamma(p)) = \gamma_0(g(p))$ for all p such that $f(\gamma(p)) \downarrow$. By the above result from [3], $c_{d_2} \circ g \circ c_{d_1}^{-1} :\subseteq \{0,1\}^{d_1} \to \{0,1\}^{d_2}$ is computable.

Then $f(\gamma(c_{d_1}^{-1}(q))) = \gamma_0(c_{d_2}^{-1}(c_{d_2}(g(c_{d_1}^{-1}(q)))))$. And if $f((\gamma \circ c_{d_1}^{-1})(q)) \downarrow$, then $f(\gamma(c_{d_1}^{-1}(q))) \downarrow$, so, $(\gamma_0 \circ c_{d_2}^{-1})((c_{d_2} \circ g \circ c_{d_1}^{-1})(q)) = \gamma_0(g(c_{d_1}^{-1}(q)) \downarrow$. So, f is $(\gamma \circ c_{d_1}^{-1}, \gamma_0 \circ c_{d_2}^{-1})$-computable.

Conversely, suppose f is $(\gamma \circ c_{d_1}^{-1}, \gamma_0 \circ c_{d_2}^{-1})$-computable. Then there exists a computable $h :\subseteq \{0,1\}^{d_1} \to \{0,1\}^{d_2}$ such that $f((\gamma \circ c_{d_1}^{-1})(q)) = (\gamma_0 \circ c_{d_2}^{-1})(h(q))$. But $c_{d_2} \circ h \circ c_{d_1}^{-1} :\subseteq \Sigma^{d_1} \to \Sigma^{d_2}$ is computable. So, if $c_{d_1}(p) = q$ we have $f(\gamma(p)) = f((\gamma \circ c_{d_1}^{-1})(c_{d_1}(p)) = (\gamma_0 \circ c_{d_2}^{-1})(h(c_{d_1}(p)) = \gamma_0((c_{d_2}^{-1} \circ h \circ c_{d_1})(p))$. I.e. $f(\gamma(p)) = \gamma_0((c_{d_2}^{-1} \circ h \circ c_{d_1})(p))$. So, f is (γ, γ_0)-computable.

□

We also have for sets that:

Theorem 1.8. $A \subseteq Y_1 \times ... \times Y_k$ is r.e. open iff $\bar{c}(Y_1) \times ... \times \bar{c}(Y_k)$ is r.e. open. Where $\bar{c}$ is c_* or c_ω as required.

Proof. We consider the case when $A \subseteq \Sigma^d, d \in \{*, \omega\}$.

Suppose A is r.e. open. Then there exists a computable $g :\subseteq \Sigma^d \to \Sigma^*$ such that $A = dom(g)$. Then $c_* \circ g \circ c_d^{-1} : \{0,1\}^d \to \{0,1\}^d$ is computable and $dom(c_* \circ g \circ c_d^{-1}) = c(A)$ is r.e. open.

Conversely suppose $c(A)$ is r.e. open in $\{0,1\}^d$. Then there exists a computable $h :\subseteq \{0,1\}^d \to \{0,1\}^*$ such that $dom(h) = c(A)$. Since for every $y \in range(c)$ we have $h(y) = c_* \circ c_*^{-1} \circ h \circ c_d \circ c_d^{-1}(y) = (c_* \circ (c_*^{-1} \circ h \circ c_d) \circ c_d^{-1})(y)$. So, $c_*^{-1} \circ h \circ c_d :\subseteq \Sigma^d \to \Sigma^*$ is computable. So since $dom(c_*^{-1} \circ h \circ c_d) = A$, A is r.e. open.

Thus, A is r.e. open iff $c(A)$ is r.e. open. A similar argument holds for $A \subseteq \Sigma^{d_1} \times ... \times \Sigma^{d_k}$.

□

So for example we can encode the alphabet $\{0, 1, /, -1\}$ with

$$c(0) = 0$$

$$c(1) = 10$$

$$c(/) = 110$$

$$c(-1) = 111$$

After which we may assume our notation $\nu_{\mathbb{Q}}$ has a binary domain, without disturbing our computability results.

1.2 Background for "Coding the Orbits of the Collatz Map"

The only unmentioned ideas we will use in Part 2 is the prefix free codes and the Kraft inequality.

Definition 1.8. A set of strings $A \subseteq \Sigma^*$ is said to be *prefix free*, if no string in A is a prefix of any other. i.e. if $u, v \in A$ and $u \leq v$, then $u = v$.

So for example the set $\{00, 01, 100, 1111\}$ is prefix free, but the set $\{00, 01, 001, 1010\}$ is not since $00 \leq 001$.

Definition 1.9. A *prefix free code* will be a 1-1 map $\phi :\subseteq \mathbb{N} \to \Sigma^*$ or $\phi :\subseteq \Sigma^* \to \Sigma^*$ such that $range(\phi)$ is prefix free.

Theorem 1.9. *Kraft's Theorem* [1, 10]: For $N \in \mathbb{N} \cup \{\infty\}$ and positive integers n_i such that $1 \leq i \leq N$, these n_i are the lengths of a prefix free code $\phi :\subseteq \mathbb{N} \to \Sigma^*$ or $\phi :\subseteq \Sigma^* \to \Sigma^*$ iff $\sum_{i=1}^{N} \frac{1}{|\Sigma|^{n_i}} \leq 1$.

In Part 2 I will only utilize the forward direction of the theorem, primarily for $\Sigma = \{0, 1\}$.

This direction follows from the fact that $(\lambda, \beta, \Sigma^\omega)$ is a Borel measure space formed by the basis of open cylinders $\{u\Sigma^\omega\}$ such that $u \in \Sigma^*$, and for each open set $X \subseteq \Sigma^\omega$, $\lambda(X) := \Sigma \frac{1}{2^{|x_i|}}$ where x_i are prefix free and $X = \bigcup x_i \Sigma^\omega$. Since measures are countably additive, if $range(\phi)$ is prefix free, then

$$\begin{aligned} 1 &= \lambda(\Sigma^\omega) \\ &\geq \lambda(\bigcup_{n \in \mathbb{N}} \phi(n)\Sigma^\omega) \\ &= \sum_{n \in \mathbb{N}} \lambda(\phi(n)\Sigma^\omega) \\ &= \sum_{n \in \mathbb{N}} \frac{1}{2^{|\phi(n)|}}. \end{aligned}$$

2 COMPUTABLE RANDOMNESS IN TWO NOTIONS OF COMPUTABLE REALS

Abstract

In [1] an effective definition of random bit sequences is defined in terms of what are known as computable supermartingales. I will show that the natural definition for computable supermartingales from the perspective of Computable Analysis [3] gives rise to a larger class of supermartingales which may be considered computable. However it will then be shown that the set of computably random sequences determined by this larger class remains unchanged. This investigation is merited by the fact that other alterations in the requirements for an effective supermartingale do lead to different classes of random sets. For example: left c.e., computable, and partially computable supermartingals all give distinct classes of computably random sets [1]. So the fact that this alteration does not is significant.

Furthermore, since the sets of computably random sets determined by the two distinct definitions of computable supermartingales in this paper is identical, this implies that one may establish computable randomness using the definitions and theorems in [3] and still be guaranteed computable randomness as specified in [1]. It also suggests a certain robustness of the definition of computable randomness as provided in [1].

2.1 Introduction

Computable randomness in AIT is a definition targeted at capturing the randomness of infinite strings of 0's and 1's [1]. The original definition of randomness for infinite bit strings which marked the beginning of AIT was characterized in the 1960's independently by Gregory Chaitin, Andrei Kolmogorov, and Ray Solomonoff [12].

This was done by recognizing that a single Turing machine which attempts to output every possible finite bit string while simultaneously attempting to minimize the length of its finite input strings will eventually require longer inputs to compute the more "complicated" bit strings in the output [12, 10].

If this Turing machine cannot output the finite initial substrings of an infinite string Z with inputs that are up to a fixed constant shorter than these strings, then Z is considered random, or Kolmogorov random.

[11] C.P. Schnorr created a definition of randomness using martingales, functions that bet on the bits of an infinite sequence Z, such that the martingales are approximable from below by a Turing machine. If no martingale of this type can bet it's way through Z to unbounded capital, then Z is considered random. Then Schnorr showed that these random bit sequences Z are equivalent to the Kolmogorov random bit sequences. Schnorr then followed up these martingales with those that are approximable from both above and below. These are called computable martingales.

There are several distinct definitions for randomness that have been proposed over the years, and organizing these definitions according to their strengths is an important problem in AIT. For example Kolmogorov randomness $\Longrightarrow$ partially computable randomness $\Longrightarrow$ computable randomness, but the converses do not hold [1, 11, 13]. And, each of these is distinct because of very different computability requirements on the reals.

I ask the question using computable analysis [3] of whether a more subtle variation in the computability requirements of a real number can produce a distinct class of computably random sets in the heirarchy.

2.2 Nies-computable martingales and Weihrauch-computable martingales are distinct

Let Σ be a finite set, called an alphabet. Σ will by default be considered as the set $\{0,1\}$ but may be assumed to contain other symbols as required. Let $L_C, R_C : \mathbb{N} \to \mathbb{N}$ be the left and right inverses of the Cantor pairing function [8].

In [1] a real number r is called computable iff $\{q \in \mathbb{Q}_2 | q < r\}$ is computable, where $\mathbb{Q}_2$ is the set of dyadic rationals. This can be rigorously defined using the methods in [3]. By restricting the domain of the standard notation of the rationals $\nu_{\mathbb{Q}}$ to $\mathbb{Q}_2$, and defining r is a computable real iff $\{q \in \mathbb{Q}_2 | q < r\}$ is $\nu_{\mathbb{Q}}|^{\mathbb{Q}_2}$-decidable. However by the density of $\mathbb{Q}$ and $\mathbb{Q}_2$ in $\mathbb{R}$ this is equivalent to: r is computable iff $\{q \in \mathbb{Q} | q < r\}$ is $\nu_{\mathbb{Q}}$-decidable. By [3], $\{q \in \mathbb{Q} | q < r\}$ is $\nu_{\mathbb{Q}}$-decidable iff there exists a computable function $g :\subseteq \Sigma^* \to \Sigma^*$ such that $g(w) = 1$ if $w \in \nu_{\mathbb{Q}}^{-1}(\{q \in \mathbb{Q} | q < r\})$ and $g(w) = 0$ if $w \in dom(\nu_{\mathbb{Q}}) \setminus \nu_{\mathbb{Q}}^{-1}(\{q \in \mathbb{Q} | q < r\})$.

Since $dom(\nu_{\mathbb{Q}})$ is decidable in Σ^* is equivalent to there exists a total computable $h : \Sigma^* \to \Sigma^*$ such that

$$h(w) := \begin{cases} 1, & w \in \nu_{\mathbb{Q}}^{-1}(\{q \in \mathbb{Q} | q < r\}) \\ 0, & w \notin \nu_{\mathbb{Q}}^{-1}(\{q \in \mathbb{Q} | q < r\}) \end{cases}.$$

I.e. that $\{w \in \Sigma^* | \nu_{\mathbb{Q}}(w) < r\}$ is decidable in Σ^* in the usual sense on Σ^*.

This is the version of the definition from [1] that we will use:

Definition 2.1. r is computable iff $\{w \in \Sigma^* | \nu_{\mathbb{Q}}(w) < r\}$ is decidable.

[3] has an alternate definition for a real number.

Definition 2.2. [3] Let $\iota : \Sigma^* \to \Sigma^*$ be the *wrapping function* defined by $\iota(w) := 110w(0)0w(1)0...0w(|w|-1)011$ where $w(k)$ indicates the k^{th} bit of w and $|w|$ is the length of w.

In [3] the standard representation ρ of $\mathbb{R}$ is defined:

Definition 2.3. $\rho : \Sigma^\omega \to \mathbb{R}$ such that $\rho(p) = x$ iff $x \in B_{\nu_\mathbb{Q}(b)}(\nu_\mathbb{Q}(a)) \iff \iota(\iota(a)\iota(b))\triangle p$. Where $B_r(y)$ is the interval in $\mathbb{R}$ centered at y with radius r, and $w\triangle p$ indicates w is a subword of p (i.e. $p = uwq$ such that $u \in \Sigma^*, p \in \Sigma^\omega$).

And, r is said to be computable iff there exists a computable function $f :\subseteq \Sigma^* \to \Sigma^\omega$ such that $\rho(f(\lambda)) = r$. I.e. there is a Turing machine that on no input computes the codes for all upper and lower rational bounds of r

The representation ρ is shown to be equivalent to the following *Cauchy representations* in [3]:

Definition 2.4. (1) $\rho_C :\subseteq \Sigma^\omega \to \mathbb{R}$ such that $\rho_C(p) := x$ iff there are words $w_0, w_1, ... \in dom(\nu_\mathbb{Q})$ such that $p = \iota(w_0)\iota(w_1)...$ where $|\nu_\mathbb{Q}(w_i) - \nu_\mathbb{Q}(w_k)| \leq \frac{1}{2^i}$ for $i < k$ and $x = \lim_{i\to\infty} \nu_\mathbb{Q}(w_i)$.

(2) $\rho'_C, \rho''_C, \rho'''_C$ obtained by substituting $|\nu_\mathbb{Q}(w_i) - \nu_\mathbb{Q}(w_k)| < \frac{1}{2^i}, |\nu_\mathbb{Q}(w_i) - x| < \frac{1}{2^i}, |\nu_\mathbb{Q}(w_i) - x| \leq \frac{1}{2^i}$ with $|\nu_\mathbb{Q}(w_i) - \nu_\mathbb{Q}(w_k)| \leq \frac{1}{2^i}$ respectively in the definition for ρ_C.

It can also easily be shown that the following representation is also equivalent to ρ:

Definition 2.5. $\rho_{(a,b)}(p) := x$ iff $x \in (\nu_\mathbb{Q}(u), \nu_\mathbb{Q}(v)) \iff \iota(\iota(u)\iota(v))\triangle p$.

Since each of these representations are equivalent they induce the same computability concepts on $\mathbb{R}$ [3], and thus may be used interchangeably.

Thus, for example $r \in \mathbb{R}$ is computable in the sense of [3] iff there exists a computable $f :\subseteq \Sigma^* \to \Sigma^\omega$ such that $\rho'_C(f(\lambda)) = r$.

I call the first type of computability Nies-computable, and the second type Weihrach-computable.

Theorem 2.1. With respect to individual reals, Nies-computable is equivalent to Weihrauch-computable.

Proof. Let $A_r := \{w \in \Sigma^* | \nu_{\mathbb{Q}}(w) < r\}$.

Suppose r is Weihrauch-computable. Then there exists a computable $f :\subseteq \Sigma^* \to \Sigma^\omega$ such that $\rho(f(\lambda)) = r$.

Case1: $r \notin \mathbb{Q}$:

Let M be the machine that on input $w \in \Sigma^*$ first tests whether $w \in dom(\nu_{\mathbb{Q}})$ returning 0 if not. Then M proceeds through the codes in $f(\lambda) = \iota(\iota(a_0)\iota(b_0))\iota(\iota(a_1)\iota(b_1))...$ checking each $\iota(a_i)\iota(b_i)$ to see if:

(1) $\nu_{\mathbb{Q}}(w) > \nu_{\mathbb{Q}}(a_i) + \nu_{\mathbb{Q}}(b_i)$

(2) $\nu_{\mathbb{Q}}(w) < \nu_{\mathbb{Q}}(a_i) - \nu_{\mathbb{Q}}(b_i)$

If (1) occurs, M returns 0 (since this implies $\nu_{\mathbb{Q}}(w) > r$).

If (2) occurs, M returns 1. $\Diamond$

(See [2] for proofs that $+, -, *, /$ are each $\nu_{\mathbb{Q}}, \nu_{\mathbb{Q}}$)-computable and "<", ">", and "=" are $(\nu_{\mathbb{Q}}, \nu_{\mathbb{Q}})$-decidable.)

Since $r \notin \mathbb{Q}$ condition (1) or (2) must occur in M. Thus $f_M = \chi_{A_r}$.

Case2: $r \in \mathbb{Q}$: Then there exists an $w_r \in \Sigma^*$ such that $\nu_{\mathbb{Q}}(w_r) = r$. Since $<$ is $(\nu_{\mathbb{Q}}, \nu_{\mathbb{Q}})$-decidable, χ_{A_r} is computable independent of the hypothesis. (Note that this proof for case 2 is nonconstructive.)

Conversely, suppose A_r is decidable.

Case1: $r \notin \mathbb{Q}$: Since A_r is decidable there exists a computable $f : \Sigma^* \to \Sigma^*$ computable such that $f = \chi_{A_r}$.

Let M be the machine that on any input $w \in \Sigma^*$ at stage $n \geq 0$ performs the following:

case(1): If $f(string(L_C(n))) = 0$ and $f(string(R_C(n))) = 1$,

print $\iota(\iota(R_C(n))\iota(L_C(n)))$ to the output and proceed to stage $n+1$.

case(2): If $f(string(L_C(n)) = 1$ and $f(string(R_C(n)) = 0$, print $\iota(\iota(L_C(n)\iota(R_C(n)))$ to the output. And proceed to stage $n+1$.

case(3): Otherwise proceed to stage $n+1$.

(Note: input w is never utilized so the output is constant.) $\Diamond$

Then $\rho_{(a,b)}(f_M(\lambda)) = r$. Hence r is $\rho_{(a,b)}$-computable, and thus ρ-computable.

Case2: $r \in \mathbb{Q}$: Then there exists a $w_r \in \Sigma^*$ such that $\nu_{\mathbb{Q}}(w_r) = r$.

Let N be the machine that on input $w \in \Sigma^*$ ignores its input and prints $\iota(w_r)\iota(w_r)...$ to its output. $\Diamond$

Then $\rho_C(f_N(\lambda)) = r$ since every ι-word w_i, w_k of $f_N(\lambda)$ is just w_r, we have $|\nu_{\mathbb{Q}}(w_i) - \nu_{\mathbb{Q}}(w_k)| = 0 \leq \frac{1}{2^i}$ and $\lim_{i\to\infty} \nu_{\mathbb{Q}}(w_i) = \lim_{i\to\infty} \nu_{\mathbb{Q}}(w_r) = \lim_{i\to\infty} r = r$. Thus r is ρ_C-computable and thus ρ-computable. This shows that all rationals are ρ-computable [3]. (Again, Case2's proof is nonconstructive.)

□

Thus both schemes give the same set of computable real numbers. We'll see later that they begin to differ when sequences are considered.

Definition 2.6. In [1] a *supermartingale* is defined to be a map $L : \Sigma^* \to \mathbb{R}^{\geq 0}$ such that $L(w0) + L(w1) \leq 2L(w)$.

And a supermartingale L is computable if it's undergraph $\{(w, q) \in \Sigma^* \times \mathbb{Q} : q < L(w)\}$ is computable.

Which we interpret to mean in the language of [3]:

Definition 2.7. A supermartingale L is *Nies-computable* if $U_L := \{(w,u) \in \Sigma^* \times \Sigma^* : \nu_{\mathbb{Q}}(u) < L(w)\}$ is decidable in $\Sigma^* \times \Sigma^*$.

This will be compared to the natural definition using the philosophy of [3]:

Definition 2.8. A supermartingale L is *Weihrauch-computable* if it is $(id^*_\Sigma, \rho|^{\mathbb{R}^{\geq 0}})$-*computable*.

Theorem 2.2. If L is Nies-computable, then L is Weihrauch-computable.

Proof. Suppose L is Nies-computable. Then $U_L := \{(w,u) \in \Sigma^* \times \Sigma^* : \nu_{\mathbb{Q}}(u) < L(w)\}$ is decidable in $\Sigma^* \times \Sigma^*$.

Let $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ be the computable function such that $f = \chi_{U_L}$. To show that L is $(id^*_\Sigma, \rho|^{\mathbb{R}^{\geq 0}})$-computable we need to establish the existence of a computable function $g : \Sigma^* \to \Sigma^\omega$ such that $L(id^*_\Sigma(w)) = \rho|^{\mathbb{R}^{\geq 0}}(g(w))$ for all $w \in \Sigma^*$ such that $L(id^*_\Sigma(w)) \downarrow$. I.e. $L(w) = \rho|^{\mathbb{R}^{\geq 0}}(g(w))$ for all $w \in \Sigma^*$.

To this end:

Let M be the machine that on input w computes in stages.

At stage $n \geq 0$: M computes in substages $k \geq 0$:

(1) If $string(L_C(k)), string(R_C(k)) \in dom(\nu_{\mathbb{Q}})$, then proceed to (2). O.w. proceed to stage $k+1$.

(2) If $f(w, string(L_C(k))) = 1, f(w, string(R_C(k))) = 0$, and $\nu_{\mathbb{Q}}(string(R_C(k))) - \nu_{\mathbb{Q}}(string(L_C(k))) < \frac{1}{2^n}$, then print $\iota(L_C(k))$ on the output and proceed to stage $n+1$. Otherwise proceed to stage $k+1$.

$\Diamond$

Then $L(w) = \rho'''_C|^{\mathbb{R}^{\geq 0}}(f_M(w))$ for all $w \in \Sigma^*$.

Hence L is $(id^*_\Sigma, \rho|^{\mathbb{R}^{\geq 0}})$-computable. I.e. L is Weihrauch-computable.

□

The converse implication does not hold for supermartingales: Weihrauch computable $\not\Longrightarrow$ Nies-computable.

We'll look at the supermartingale:

$L^*(w) := \frac{1+x(num(w))}{2^{|w|}}$ such that

$$x(n) = \begin{cases} 0, & \text{if } n \notin range(a) \\ \frac{1}{2^k}, & \text{if } a(k) = n \end{cases}$$

where $a : \mathbb{N} \to \mathbb{N}$ is a 1-1 total computable function such that $range(a)$=K, the halting set (such a function a exists by [4]). And, we'll show L^* is Weihrauch-computable, but not Nies-computable. The sequence $x : \mathbb{N} \to \mathbb{Q}$ is mentioned in [3].

First we'll check:

Lemma 2.1. L^* is a supermartingale.

Proof. Since $|x(n)| \leq 1$ for all n,

$$\begin{aligned} L^*(w0) + L^*(w1) &= \frac{1 + x(num(w0))}{2^{|w0|}} + \frac{1 + x(num(w1))}{2^{|w1|}} \\ &= \frac{2 + x(num(w0)) + x(num(w1))}{2^{|w|+1}} \\ &\leq \frac{2+1+1}{2^{|w|+1}} \\ &= \frac{2}{2^{|w|}} \\ &\leq 2\frac{(1 + x(num(w)))}{2^{|w|}} \\ &= 2L^*(w). \end{aligned}$$

□

Lemma 2.2. L^* is not Nies-computable.

Proof. Suppose L^* is Nies-computable for a contradiction. Then $\chi_{U_{L^*}}$ is computable.

> Let N be the machine that on input $w_n = string(n)$ computes $\chi_{U_{L^*}}(w_n, \nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}}))$ and returns λ if $\chi_{U_{L^*}}(w_n, \nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}})) = 0$, and 0 if $\chi_{U_{L^*}}(w_n, \nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}})) = 1$. $\Diamond$

Where we use the convention that for $q \in \mathbb{Q}$, $\nu_{\mathbb{Q}}^{-1}(q)$ is some code in $dom(\nu_{\mathbb{Q}})$ for q that is effectively attainable.

Then χ_K is (num, num)-computable, since

$n \in K \implies x(n) = \frac{1}{2^k} > 0$ for some $k \geq 0$.

So

$$\begin{aligned} L^*(w_n) &= \frac{1 + x(n)}{2^{|w_n|}} \\ &> \frac{1}{2^{|w_n|}} \\ &= \nu_{\mathbb{Q}}(\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}})) \end{aligned}$$

Thus, $\chi_{U_{L^*}}(w_n, \nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}})) = 1$.

And, $n \notin K \implies$

$$\begin{aligned} L^*(w_n) &= \frac{1 + x(n)}{2^{|w_n|}} \\ &= \frac{1 + 0}{2^{|w_n|}} \\ &= \frac{1}{2^{|w_n|}} \\ &= \nu_{\mathbb{Q}}(\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}})) \end{aligned}$$

So, $\chi_{U_{L^*}}(w_n, \nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w_n|}})) = 0$.

Hence, $\chi_K(num(w_n))) = num(f_N(w_n))$. A contradiction since K is not (num, num)-decidable.

□

Lemma 2.3. L^* is $(id_{\Sigma^*}, \rho|^{\mathbb{R}^{\geq 0}})$-computable.

Proof. We need to show there exists a computable $h : \Sigma^* \to \Sigma^\omega$ such that $L^*(w) = \rho|^{\mathbb{R}^{\geq 0}}(h(w))$ for all $w \in \Sigma^*$. To this end we:

Let M be the machine that on input w computes in stages:

At stage $k \geq 0$, M computes $a(k)$.

(1) If $a(k) = num(w)$, then print $\iota(\nu_{\mathbb{Q}}^{-1}\left(\frac{1+\frac{1}{2^k}}{2^{|w|}}\right))\iota(\nu_{\mathbb{Q}}^{-1}\left(\frac{1+\frac{1}{2^k}}{2^{|w|}}\right))$... on the output.

(2) If $a(k) \neq num(w)$, print $\iota(\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{|w|}} + \frac{1}{2^k}))$ on the output and proceed to stage $k+1$. $\Diamond$

If case (1) occurs in M then we found $L^*(w)$ exactly so we just repeat its code indefinitely.

If case (2) occurs in M then either $L^*(w) = \frac{1+\frac{1}{2^m}}{2^{|w|}}$, for some $m > k$ or $L^*(w) = \frac{1+0}{2^{|w|}} = \frac{1}{2^{|w|}}$, i.e. $num(w) \notin K$.

In either case $|L^*(w) - (\frac{1}{2^{|w|}} + \frac{1}{2^k})| \leq \frac{1}{2^k}$. Thus, $L^*(w) = \rho_C'''|^{\mathbb{R}^{\geq 0}}(f_M(w))$ for $w \in \Sigma^*$. Which implies that L^* is $(id_{\Sigma^*}, \rho|^{\mathbb{R}^{\geq 0}})$-computable.

□

Thus,

Theorem 2.3. L^* is Weihrauch-computable but not Nies-computable.

We now proceed to the discussion of computable randomness.

2.3 Nies-computable randomness = Weihrauch-computable randomness

Definition 2.9. In [1], a supermartingale is said to *succeed* on a sequence $Z \in \{0,1\}^\omega$ iff $\sup_n L(Z|n) = \infty$ where $Z|n := Z(0)Z(1)...Z(n-1)$. And a set Z is called *computably*

random iff there does not exist a Nies-computable supermartingale which succeeds on Z.

By the proof of the theorem above, we showed that the Nies-computable supermartingales are a subset of the Weihrauch-computable supermartingales. We can easily adapt the defintions above in the obvious way to form Weihrauch-computably random sequences. An immediate question then becomes can we find a Nies-computably random sequence Z that is not Weihrauch-computably random? I.e. is there a supermartingale in the larger set of Weihrauch-computable supermartingales that succeeds on Z where the Nies-computable supermartingales fail?

The answer to this question is negative. The randomness notions coincide.

Theorem 2.4. The set of Nies-computably random sequences is identical to the set of Weirauch-computably random sequences.

Proof. Suppose L is $(id_{\Sigma^*}, \rho|^{\mathbb{R}^{\geq 0}})$-computable supermartingale (i.e. L is Weihrauch-computable) and L succeeds on $Z \in \Sigma^\omega$. We'll show that there is a Nies-computable supermartingale R that also succeeds on Z.

Since L is $(id_{\Sigma^*}, \rho|^{\mathbb{R}^{\geq 0}})$-computable, there exists a computable $h : \Sigma^* \to \Sigma^\omega$ such that $L(w) = \rho|^{\mathbb{R}^{\geq 0}}(h(w))$ for all $w \in \Sigma^*$.

> Let M be the machine that on input $(w, string(n))$ begins computing $h(w)$ until an ι-word of the form $\iota(\iota(a)\iota(\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{n+1}})))$ is found, for some $a \in dom(\nu_{\mathbb{Q}})$, and returns $\nu_{\mathbb{Q}}^{-1}(\nu_{\mathbb{Q}}(a) + \frac{1}{2^{n+1}})$. $\Diamond$

Then f_M is total computable from the definition of ρ and the total computability of L.

And we have $0 < \nu_{\mathbb{Q}}(f_M(w, string(n))) - L(w) < \frac{1}{2^n}$ for all w and n.

Now define $\hat{R}_n(w) := \nu_{\mathbb{Q}}(f_M(w, string(n)))$.

And define R inductively as follows:

(1) Define $R(\lambda) := \hat{R}_0(\lambda)$.

(2) Let $s(w) := \min_{t \geq 0} \frac{\hat{R}_t(w0)+\hat{R}_t(w1)}{2} \leq R(w)$.

(3) And define $R(w0) := \hat{R}_{s(w)}(w0)$ and $R(w1) := \hat{R}_{s(w)}(w1)$. Define all $R(w)$ for $|w| = n$ before defining $R(w)$ for $|w| = n + 1$.

Then such an R is a supermartingale since:

$\frac{R(w0)+R(w1)}{2} = \frac{\hat{R}_{s(w)}(w0)+\hat{R}_{s(w)}(w1)}{2} \leq R(w)$.

Now we check that R is defined by induction on w.

Suppose that $R(w)$ is defined for some w. Then there exists an n such that $R(w) = \hat{R}_n(w)$. Then by definition of $\hat{R}_n$, $0 < \hat{R}_n(w) - L(w) < \frac{1}{2^n}$. I.e. $0 < R_n(w) - L(w) < \frac{1}{2^n}$.

So there exists a $p \in \mathbb{N}$ such that $\frac{1}{2^p} < R(w) - L(w) < \frac{1}{2^n}$. Which implies $L(w) + \frac{1}{2^p} < R(w) < L(w) + \frac{1}{2^n}$. Then

$$\begin{aligned}
\frac{\hat{R}_p(w0) + \hat{R}_p(w1)}{2} &< \frac{(L(w0) + \frac{1}{2^p}) + (L(w1) + \frac{1}{2^p})}{2}, \text{ since } 0 < \hat{R}_p(wi) - L(wi) < \frac{1}{2^p} \text{ for } i = 0, 1, \\
&\leq \frac{2L(w) + \frac{2}{2^p}}{2} \\
&= L(w) + \frac{1}{2^p} \\
&< R(w).
\end{aligned}$$

This implies that $s(w)$ exists, and hence $R(w0) = \hat{R}_{s(w)}(w0)$ and $R(w1) = \hat{R}_{s(w)}(w1)$ are defined as well. So the supermartingale R is well defined.

Furthermore it is clear that $R(w) > L(w)$ for all w. Thus, R succeeds on Z.

Now we show that R is Nies-computable. I.e. we show $U_R := \{(w, u) \in \Sigma^* \times \Sigma^* | \nu_{\mathbb{Q}}(u) < R(w)\}$ is $\Sigma^* \times \Sigma^*$ decidable.

Let N be the machine that on input $(w, u) \in \Sigma^* \times \Sigma^*$ computes the rational $R(w)$ according to the inductive definition of R and,

(1) If $\nu_{\mathbb{Q}}(u) < R(w)$, returns 1.

(2) If $\nu_{\mathbb{Q}}(u) \geq R(w)$ or $u \notin dom(\nu_{\mathbb{Q}})$, returns 0. $\Diamond$

Since $f_M : \Sigma^* \to \Sigma^*$ is total computable and $<$ and $\geq$ are $(\nu_{\mathbb{Q}}, \nu_{\mathbb{Q}})$-decidable, f_N is a total computable function which calculates χ_{U_R}.

So, R is Nies-computable.

Thus, R is a Nies-computable supermartingale which succeeds on Z. So, if a Weihrauch-computable supermartingale succeeds on some set Z, then there is a Nies-computable supermartingale which also succeeds on Z. Therefore, the Weihrauch-computably random sets are equivalent to the Nies-computably random sets.

□

Despite the fact that Weihrauch-computability does not form a separate notion of computable randomness, the utility in this finding is that one may freely use the intuitive tools in [3] including the various naming systems $\rho, \rho_C, \rho'_C, ...$ when demonstrating Nies-computable randomness, by proving equivalently that a set Z is Weihrauch-computably random. It also suggests a certain robustness in the notion of computable randomness: that it is not disturbed by subtle variations in the definition of computability on the reals.

Several alternate definitions of computable reals are recorded in the appendix of [3]. An interesting follow up question is whether these alternate definitions also lead to the same computably random sets.

3 CODING THE ORBITS OF THE COLLATZ MAP

Abstract

The 3x+1 problem or Collatz conjecture asks whether repeated self composition of the Collatz map $C : \mathbb{N} \to \mathbb{N}$ such that $C(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases}$, returns all positive integers to 1. I.e whether for every $n \in \mathbb{N}$ there exists a $k \in \mathbb{N}$ such that $C^{(k)}(n) = 1$.

It is often stated that the question resists solution because it has inherent complexities. I.e. not that it simply appears complicated because of how we are approaching it, but that it contains some intrinsic property of randomness. Thus the problem is naturally an attractive question for complexity theorists [10].

In this paper I will show how one may create a prefix-free binary code for the orbits of each convergent integer and apply the Kraft inequality to analyze the total stopping times: the minimum k required for $C^{(k)}(n) = 1$. Finally, I will attempt to phrase the problem of measuring the complexity of the orbits of the collatz map in terms of computable randomness, using the codewords for the paths which will be shown to be a decidable set in $\{0, 1\}^*$.

3.1 Introduction

In [10] Calude asks in what way we can understand the complexity of several famously difficult problems including the Collatz conjecture. There has been a lot of work from the dynamical systems perspective [14] but not as much from algorithmic information theory side. This motivated the following investigation. In [12] there is a discussion of

how Gregory Chaitin had used the description lengths of a code for the integers to reprove some results about the distribution of the primes from the algorithmic information theory perspective. I realized that the binary path of an integer to 1 by the Collatz map provided a code for the integers as well. And, the Kraft inequality is the primary tool for investigating consequences of code lengths.

Kraft's inequality can be used to focus knowledge on the $3x + 1$ problem into a single statement of orbit lengths:

$a < \sum_{k=1}^{\infty} \frac{1}{2^{|\phi(k)|}} \leq b$. Where $a \geq 0$ and $b \leq 1$.

a represents your knowledge about which numbers satisfy the conjecture and b represents your knowledge about impossible orbits to 1. Each piece of knowledge concerning paths of the numbers which satisfy the conjecture increases the value of a, and each piece of knowledge concerning the impossibility of paths reduces the value of b. If b is ever brought below a then the Collatz conjecture is false.

Despite the fact that paths are in the Baire space $\mathbb{N}^{\mathbb{N}}$, each descision by the Collatz map

$$C(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases},$$

is binary. Since I am only interested in the information content of the names generated by the path of the Collatz map in the Baire space, binary representations are more to the point. I first begin with what I call the standard code. It is very simple, start with $n \in \mathbb{N}^{\geq 1}$, every time you divide by 2 write a 0, and every time you multiply by 3 and add 1 write a 1. Stop the process when $C^{(k)}(n) = 1$. Finally append 11 to the code w_n generated by this proceedure. The 11 suffix ensures that the code is prefix-free (i.e. no code is a substring of any other). The prefix-free property is what guarantees the Kraft inequality. (I'm using the convention of marking non-orbit bits of the code in red, these bits are only to ensure that the codes are prefix-free.)

Example 3.1. When we iterate C on 3 we get the integers: $10, 5, 16, 8, 4, 2, 1$. Thus our standard code for 3 is 10100001$\underline{1}$.

The standard code will allow us to add value to the a term discussed above.

Next I will discuss the reverse code. The reverse code is essentially the standard code written in reverse. A new coding scheme is required for those codes to ensure that they will remain prefix free without changing their lengths. Since the final reverse code words have the same length as the standard codes, the results on the orbit lengths for the reverse code may be easily related to the standard code. Reversing the standard codes makes it easier to rule out impossible codes. Each time we rule out impossible codes we can decrease the value of b mentioned above.

The utility of this idea is that it could have potential for disproving the conjecture if ever $b < a$. But, more practically it allows us to test our hypotheses about the behavior of the total stopping time function. Furthermore, the analysis of binary codes of the orbits of the Collatz map can allow one to investigate the complexity of the behavior of this map on the integers from the viewpoint of [1], which is investigated in the last section of this paper.

Other authors have considered some aspect of binary trees or binary encodings of the Collatz map under iterations. In [14], subtrees of the Baire space are recognized as binary trees. This is used as an aide for constructing algorithms which verify classes of numbers for which iterations return to 1. No binary encodings scheme is explicitly considered however. In [15, 16] binary trees are implemented in order to assist in establishing classes of numbers which converge to 1. In [17] an explicit binary encoding scheme is considered similar to the one in this paper. But, the encoding terminates as soon as $C^{(i)}(n) < n$ for some i, i.e. the i^{th} iteration of the Collatz map falls below the original value. The primary motivation for this encoding scheme is to establish a concise strategy for proving the conjecture by induction. This encoding is also prefix free, but fails to be 1-1 for each n and is thus

ineffective as code for $\mathbb{N}$, and thus the Kraft inequality cannot be directly applied. [17] also observes the relative ease with which patterns of certain allowable orbits to 1 may be discovered when viewed in a binary setting. In [18] a coding idea is implemented with the goal of extending the Collatz conjecture into dense sets.

3.2 The Standard Code

Definition 3.1. I denote the *standard code* whose algorithm was described in the introduction as $\phi :\subseteq \mathbb{N} \to \{0,1\}^*$ such that $\phi(n) = \begin{cases} w_n 11, & \text{if } n \text{ returns to 1} \\ \uparrow, & \text{o.w.} \end{cases}$

I call w_n the *binary path* of n.

Below are $\phi(n)$ for $1 \leq n \leq 16$:

$1 - \lambda 11$

$2 - 011$

$3 - 1010000 11$

$4 - 0011$

$5 - 1000011$

$6 - 01010000 11$

$7 - 101010010001000011$

$8 - 00011$

$9 - 1001010100100010000 11$

$10 - 01000011$

$11 - 10100100010000 11$

$12 - 001010000 11$

$13 - 10001000011$

$14 - 01010100100010000011$

$15 - 10101010000010000011$

$16 - 000011$

.

Theorem 3.1. The standard code is prefix free and 1-1.

Proof. That ϕ is prefix free follows from the fact that:

(1) No binary path can end with 1 ($3a + 1 = 1$ for no $a \geq 1$), and

(2) No binary path w_n contains the subword 11 ($3a + 1$ is only applied when a is odd, but then $3a + 1$ is itself even, so a 0 must follow).

Thus, if $\phi(n) \leq \phi(m)$, then $w_n 11 \leq w_m 11$. By (1) and (2) this implies $w_n 11 = w_m 11$, i.e. $\phi(n) = \phi(m)$. So, ϕ is prefix free.

That ϕ is 1-1 follows from the fact that given $w_n 11$ we can remove 11 and then uniquely reconstruct n by reading the binary code w_n in reverse: start at 1 and whenever a 0 is encountered in w_n multiply by 2, whenever a 1 in w_n is encountered subtract 1 and divide by 3. Thus, ϕ is invertible and hence 1-1. Thus, ϕ is 1-1.

□

Since the standard code is prefix free the Kraft inequality applies:

$0 < \sum_{n=1}^{\infty} \frac{1}{2^{|\phi(n)|}} \leq 1.$

Where we make the natural assignment $|\phi(n)| = \infty$ if $\phi(n) \uparrow$, and thus $\frac{1}{2^\infty} = 0$, which removes divergent paths from the sum.

By studying binary paths we can verify some well known results of convergent orbits in the context of the standard code and achieve our goal of illustrating how to improve our lower

bound on the sum. My process for verifying these facts on convergent orbits is simply to write down several examples of binary paths, form a conjecture about a pattern I'm noticing in the binary words, and then verify the conjecture with induction. This is similar to [17].

Example 3.2. (Known convergence results)

(1) For $n = 2^k, k \geq 0$, we have $\phi(2^k) = 0^k11$. So $|\phi(2^k)| = k + 2$. Thus, $\sum_{k=0}^{\infty} \frac{1}{2^{|\phi(2^k)|}} = \sum_{k=0}^{\infty} \frac{1}{2^{k+2}} = \frac{1}{2}$.

(2) For $n = \frac{2^{2m}-1}{3}, m \geq 2$, we have $\phi(\frac{2^{2m}-1}{3}) = 10^{2m}11$. This implies $\phi(2^k(\frac{2^{2m}-1}{3})) = 0^k10^{2m}11$ for $k \geq 0$ and $m \geq 2$. So $|\phi(2^k(\frac{2^{2m}-1}{3}))| = k + 2m + 3$. Thus, $\sum_{k=0}^{\infty}\sum_{m=2}^{\infty} \frac{1}{2^{|\phi(2^k(\frac{2^{2m}-1}{3}))|}} = \sum_{k=0}^{\infty}\sum_{m=2}^{\infty} \frac{1}{2^{k+2m+3}} = \frac{1}{48}$.

(3) For $n = 2^k * 3, k \geq 0$ we have $\phi(2^k * 3) = 0^k1010000011$. Thus, $\sum_{k=0}^{\infty} \frac{1}{2^{|\phi(2^k*3)|}} = \sum_{k=0}^{\infty} \frac{1}{2^{k+9}} = \frac{1}{2^8}$.

Since 0^k11, $0^k10^{2m}11$, and $0^k1010000011$ are distinct codes in the binary tree we can sum their contributions $\frac{1}{2} + \frac{1}{46} + \frac{1}{2^8} = \frac{3095}{5888}$ to form a lower bound on the sum of the weights: $\frac{3095}{5888} < \sum_{n=1}^{\infty} \frac{1}{2^{|\phi(n)|}} \leq 1$.

3.3 The Reverse Code

The next prefix free code is called the Parity Reverse Code since it writes the binary paths w_n in reverse w_n^R and alternates its prefix free suffixing, and now prefixing, strategy with

evens and odds.

Definition 3.2. I denote this code $\phi_{PR} :\subseteq \mathbb{N} \to \Sigma^*$ such that

$$\phi_{PR}(n) = \begin{cases} w_n^R 11, & \text{if } n \text{ is even or 1, and } w_n \text{ exists} \\ 1 w_n^R 1, & \text{if } n \text{ is odd but not 1, and } w_n \text{ exists} \\ \uparrow, & \text{o.w.} \end{cases}$$

Theorem 3.2. ϕ_{PR} is prefix free and 1-1.

Proof. That ϕ_{PR} is prefix free follows from the following facts:

(1) If n is even, $\phi_{PR}(n)$ starts with 0.

(2) If n is odd and not 1, then $\phi_{PR}(n)$ starts with 10.

(3) For n even, w_n begins with 0, so w_n^R ends with 0.

(4) For n odd and not 1, w_n begins with 10 and thus $\phi_{PR}(n)$ must end with 011.

(5) Since 11 is not a substring of w_n, 11 is not a substring of w_n^R.

By (1) and (2), no even code word can be a substring of an odd code word and and no odd code word can be a substring of an even code word.

Thus, if $\phi_{PR}(n) \leq \phi_{PR}(m)$, then there's three cases:

case1: If n and m are even, then $w_n^R 11 \leq w_n^R 11$. But, by (3) and (5) this implies $w_n^R 11 = w_n^R 11$, i.e. $\phi_{PR}(n) = \phi_{PR}(m)$.

case2: If n and m are odd and both not 1, then $1 w_n^R 1 \leq 1 w_m^R 1$. But, by (2), (4), and (5) this implies $1 w_n^R 1 = 1 w_m^R 1$, i.e. $\phi_{PR}(n) = \phi_{PR}(m)$.

case3: If $n = 1$ or $m = 1$, then by (1) and (2), 1 is the only integer whose code word begins with 11, so $\phi_{PR}(n) = \phi_{PR}(m)$.

Thus, ϕ_{PR} is prefix free. That ϕ_{PR} is 1-1 again follows from the fact that it is invertible.

□

$\phi_{PR}(n)$ for $1 \leq n \leq 16$ is given below:

$$1 - \lambda 11$$

$$2 - 011$$

$$3 - 100001011$$

$$4 - 0011$$

$$5 - 100001 1$$

$$6 - 0000101011$$

$$7 - 100001000100101011$$

$$8 - 00011$$

$$9 - 1000010001001010011$$

$$10 - 0000101 1$$

$$11 - 1000010001001011$$

$$12 - 0000101001 1$$

$$13 - 100001000 11$$

$$14 - 00001000100101010 11$$

$$15 - 10000100000101010101$$

$$16 - 00001 1$$

.

Since the Parity Reverse code is again prefix free, the Kraft inequality applies:

$0 < \sum_{n=1}^{\infty} \frac{1}{2^{|\phi_{PR}(n)|}} \leq 1.$

Using the measure interpretation of the Kraft Inequality allows us to reduce the upper bound by ruling out impossible branches in the binary tree.

One can show by induction that binary paths may not end in the following suffixes:

- 100
- $10^{2k+1}, k \geq 0$
- $10^{2l}10^{6p+4}, p \geq 0, l \geq 1$

The proof strategy using the framework for the binary paths for $10^{2l}10^{6p+4}, p \geq 0$, and $l \geq 1$ is as follows:

Proof. If $10^{2l}10^{6p+4}, p \geq 0, l \geq 1$ provides a binary path of some n to 1, then taking n through the iterations of the map prescribed by the path we see that n must satisfy the equation $\frac{3^2n+3+2^{2l}}{2^{2l+6p+4}} = 1$. This implies that $3n + 1 = \frac{2^{2l}(2^{6p+4}-1)}{3}$. But as shown below $\frac{2^{2l}(2^{6p+4}-1)}{3} = 3m + 2$ for some m, a contradiction.

One starts with $\frac{2^{6p+4}-1}{3} = 3m + 2$ for $p \geq 0$.

Subproof.

Base case: For $p = 0$, $\frac{2^{6p+4}-1}{3} = \frac{2^{6(0)+4}-1}{3} = 5 = 3(1) + 2$.

Induction Hypothesis: Suppose $\frac{2^{6p+4}-1}{3} = 3m + 2$ for some $m \in \mathbb{Z}$. Then $\frac{2^{6(p+1)+4}-1}{3} = \frac{2^6 2^{6p+4}-1}{3} = \frac{2^6(2^{6p+4}-1+1)-1}{3} = \frac{2^6(3\frac{2^{6p+4}-1}{3}+1)-1}{3} = \frac{2^6(3(3m+2)+1)-1}{3} = \frac{2^6(3(3m+2)+1)-1}{3} = \frac{2^6(9m+7)-1}{3} = \frac{576m-447}{3} = 192m + 149 = (192m + 147) + 2 = 3(64m + 49) + 2$. Thus, for every $p \geq 0$ there's some m such that $\frac{2^{6p+4}-1}{3} = 3m + 2$.

Now $\frac{2^{2l}(2^{6p+4}-1)}{3} = 3m + 2$ for all $p \geq 0, l \geq 1$ follows by induction on l:

Base case: For $l = 1$ we have $\frac{2^{2l}(2^{6p+4}-1)}{3} = \frac{2^2(2^{6p+4}-1)}{3} = 4(3m + 2) = 12m + 8 = (12m + 6) + 2 = 3(4m + 2) + 2$.

Induction Hypothesis: Suppose $\frac{2^{2l}(2^{6p+4}-1)}{3} = 3m + 2$. Then $\frac{2^{2(l+1)}(2^{6p+4}-1)}{3} = \frac{2^{2(l+1)}(2^{6p+4}-1)}{3} = 4\frac{2^{2l}(2^{6p+4}-1)}{3} = 4(3m+2) = 12m+8 = (12m+6)+2 = 3(4m+2)+2$.

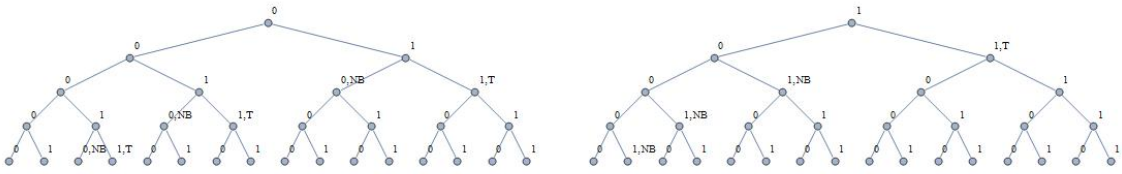
□

Since the binary paths are forbidden from ending with the above codes, it follows that the reverse codes may not begin with:

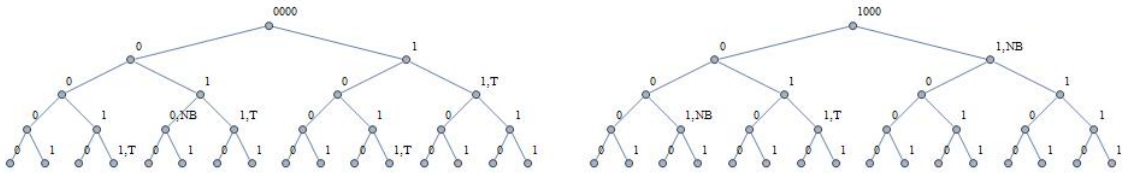
- 001 or 1001 except for $\phi_{PR}(4) = 0011$
- $0^{2k+1}1$ or $10^{2k+1}1$, excpet for $\phi_{PR}(2^k) = 0^{2k+1}11$ for $k \geq 0$
- $0^{6p+4}10^{2l}1$ or $10^{6p+4}10^{2l}1$, except for $\phi_{PR}((\frac{2^{6p+4}-1}{3})2^{2l}) = 0^{6p+4}10^{2l}11$, for $l \geq 1$ and $p \geq 0$.

I refer to these impossible prefixes in the binary tree as Nonbranching nodes (NB) as in [10].

Below is the binary tree to level 5. T indicates a terminal node: the location of a parity reverse code; and NB indicates a nonbranching node: a node in which an impossible branch first emerges.



The following graph continues with the only undetermined branches at level 4: 0000 and 1000.



Deductions from nonbranching nodes after root 0 are:

(1) For 0010, we deduct $\frac{1}{16}$.

(2) For $0^{2k+1}10$, we deduct $\frac{1}{2^{2k+3}}$ for each $k \geq 0$. Deducting totally: $\sum_{k=0}^{\infty} \frac{1}{2^{2k+3}} = \frac{1}{6}$.

(3) For $0^{6p+4}10^{2l}1$, we deduct $\frac{1}{2^{6p+2l+7}}$ for each $l \geq 1$ and $p \geq 0$. Deducting totally: $\sum_{p=0}^{\infty} \sum_{l=1}^{\infty} \frac{1}{2^{6p+2l+7}} = \frac{1}{378}$.

(4) For 1001, we deduct $\frac{1}{16}$.

(5) For $10^{2k+1}1$, we deduct $\frac{1}{2^{2k+3}}$ for each $k \geq 0$. Deducting totally: $\sum_{k=0}^{\infty} \frac{1}{2^{2k+3}} = \frac{1}{6}$.

(6) For $10^{6p+4}10^{2l}$, we deduct $\frac{1}{2^{6p+2l+7}}$ for each $l \geq 1$ and $p \geq 0$. Deducting totally: $\sum_{p=0}^{\infty} \sum_{l=1}^{\infty} \frac{1}{2^{6p+2l+7}} = \frac{1}{378}$.

Net deductions are thus: $\frac{2}{16} + \frac{2}{6} + \frac{2}{378} = \frac{701}{1512}$.

And thus, $0 < \sum_{n=1}^{\infty} \frac{1}{2^{\phi_{PR}(n)}} \leq 1 - \frac{701}{1512} = \frac{811}{1512}$.

Now since it is immediately clear that $|\phi_{PR}(n)| = |\phi(n)|$, we have:

$\frac{3095}{5888} < \sum_{n=1}^{\infty} \frac{1}{2^{|\phi(n)|}} \leq \frac{811}{1512}$. Giving us our sum to approximately 0.011.

Definition 3.3. Let $\sigma_\infty(n)$ denote the *total stopping time function*, the minimum number of iterations of the Collatz map required to reach 1 starting at n.

Since $\sigma_\infty(n) = |w_n|$, $|\phi(n)| = |w_n| + 2 = \sigma_\infty(n) + 2$. We have:

$\frac{3095}{1472} < \sum_{n=1}^{\infty} \frac{1}{2^{\sigma_\infty(n)}} \leq \frac{811}{378}$.

Or approximately to 5 decimals: $2.10258 < \sum_{n=1}^{\infty} \frac{1}{2^{\sigma_\infty(n)}} \leq 2.14551$. An error of about 0.042.

These bounds tell us three things about the total stopping times:

(1) They can be used immediately to reject estimates of the total stopping times obtained by hueristic probability arguments or other nondeductive means (see [14] for examples), which force the sum out of bounds.

(2) Conversely they could be called upon as further empirical evidence that a conjecture about the behavior of the total stopping time is correct when they do satisfy the bounds.

(3) One can form an upper bound of the number of possible total stopping times of each length remaining. For example: let $A_m := \{n||\phi(n)| = m\}$. Then the weight contributed to the graph of these $n \in A_m$ is : $\frac{|A_m|}{2^m}$. Since the total weight is bounded above by $\frac{811}{1512} < 0.53638$, this tells us that $|A_m| < 0.53638 * 2^m$, i.e. only about half the nodes at level m are utilizable as terminal, or substrings of binary paths.

3.4 Symmetries in Graph Weight

In the previous calculations there appeared to be similar results when calculating the weights of the branches of the graph corresponding to the binary paths of even or odd

numbers. This is supported by the following calculations:

$$\begin{aligned}\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(n)|}} &= \sum_{n=0}^{\infty}\frac{1}{2^{|\phi(2n+1)|}} + \sum_{n=1}^{\infty}\frac{1}{2^{|\phi(2n)|}} \\ &= \sum_{n=0}^{\infty}\frac{1}{2^{|\phi(2n+1)|}} + \sum_{n=1}^{\infty}\frac{1}{2^{|\phi(n)|+1}}\end{aligned}$$

Which implies: $\frac{1}{2}\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(n)|}} = \sum_{n=0}^{\infty}\frac{1}{2^{|\phi(2n+1)|}}$ or $\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(n)|}} = 2\sum_{n=0}^{\infty}\frac{1}{2^{|\phi(2n+1)|}}$. Thus the total weight of the graph is twice the weight as the contributions of the odds. Which implies the weight contribution of the odds is equal to the weight contributions of the evens.

We can continue this procedure:

$$\begin{aligned}\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(n)|}} &= 2\sum_{n=0}^{\infty}\frac{1}{2^{|\phi(2n+1)|}} \\ &= 2(\frac{1}{2^{|\phi(1)|}} + \sum_{n=1}^{\infty}\frac{1}{2^{|\phi(2n+1)|}}) \\ &= 2(\frac{1}{2^2} + \sum_{n=1}^{\infty}\frac{1}{2^{|\phi(3(2n+1)+1)|-1}}) \\ &= \frac{1}{2} + 4\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(6n+4)|}} \\ &= \frac{1}{2} + 4\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(3n+2)|+1}} \\ &= \frac{1}{2} + 2\sum_{n=1}^{\infty}\frac{1}{2^{|\phi(3n+2)|}}.\end{aligned}$$

Which tells us the total graph weight is determined by the numbers $3n+2, n \geq 1$.

This also gives us: $\sum_{n=0}^{\infty} \frac{1}{2^{|\phi(2n+1)|}} = \frac{1}{4} + \sum_{n=1}^{\infty} \frac{1}{2^{|\phi(3n+2)|}}$, which implies: $\sum_{n=1}^{\infty} \frac{1}{2^{|\phi(2n+1)|}} = \sum_{n=1}^{\infty} \frac{1}{2^{|\phi(3n+2)|}}$.

These observations and $|\phi(n)| = \sigma_\infty(n) + 2$ provide us with the following facts about the total stopping times:

Theorem 3.3. The following equations hold:

(1) $\sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(n)|}} = 2 \sum_{n=0}^{\infty} \frac{1}{2^{|\sigma_\infty(2n+1)|}}$,

(2) $\sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(n)|}} = 2 + 2 \sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(3n+2)|}}$,

(3) $\sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(2n+1)|}} = \sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(3n+2)|}}$.

3.5 The Ternary Code

One can also pursue this investigation with an alternate version of the Collatz map:

$$T(n) = \begin{cases} \frac{n}{2}, & n \text{ is even} \\ \frac{3n-1}{2}, & n \text{ is odd} \end{cases}$$

by assinging a 0 whenever n is even and 1 when n is odd. This replaces the "01" subwords from the prvious codes with just "1". In doing this however, we can no longer ensure the prefix-free property by appending 11, since 11 may now be a subword of a binary path. To ensure that the paths are prefix free I resorted to the simple ternary code: $\phi^T(n) := w_n *$, where w_n is the binary path given now by iterations of T, and $*$ is a new special symbol. An example of the first 16 Ternary codes:

$2 - 0*$

$3 - 11000*$

$4 - 00*$

$5 - 1000*$

$6 - 011000*$

$7 - 11101001000*$

$8 - 000*$

$9 - 1011101001000*$

$10 - 01000*$

$11 - 1101001000*$

$12 - 0011000*$

$13 - 1001000*$

$14 - 011101001000*$

$15 - 111100001000*$

$16 - 0000*$

.

From the ternary code one can then simply define the ternary reverse code simply as $\phi_R^T(n) := w_n^R*$. Because the binary paths are unique it is trivial to show that these codes are prefix free.

Since our alphabet now consists of 3 symbols: $\{0, 1, *\}$, the Kraft inequality becomes:

$0 < \sum_{n=1}^{\infty} \frac{1}{3^{|\phi^T(n)|}} \leq 1.$

By going through similar arguments as for the binary tree: building up the weight from below with ϕ^T and trimming the weight from above with ϕ_R^T from the ternary tree we can

arrive at the ineqalities:

$0.5086 < \sum_{n=1}^{\infty} \frac{1}{3^{|\phi^T(n)|}} < 0.5134.$

and

$1.5258 < \sum_{n=1}^{\infty} \frac{1}{3^{\sigma_\infty^T(n)}} < 1.5402.$

Since $\phi^T(n) = \sigma_\infty^T(n) + 1$ where $\sigma_\infty^T(n)$ is the total stopping time using map T.

The ternary code may be of greater interest as a means of studying weights of binary paths since T is more commonly used than C when studying the conjecture [14]. I prefer the binary codes however since they are more readily applicable to the concepts in [1, 10].

[14] refers to results which estimate $\sigma_\infty^T(n)$ is on average approximately bounded below by (1)$6.95212log(n)$, (always) above by (3)$41.677647log(n)$, and is approximated on average by: (2)$\frac{1}{x}\sum_{n=1}^{x} \sigma_\infty^T(n) \approx \frac{2}{ln(\frac{4}{3})} ln(x)$.

In *log*-base 3 these are:

(1) $\approx 3.31700log_3(n)$

(2) $\approx 7.63768341log_3(n)$

(3) $\approx 19.885291log_3(n)$.

These results should imply from (1) and (3) that:

$\sum_{n=1}^{\infty} \frac{1}{n^{19.885291}} < \sum_{n=1}^{\infty} \frac{1}{3^{\sigma_\infty^T(n)}} < \sum_{n=1}^{\infty} \frac{1}{n^{3.31700}}.$

But, $\sum_{n=1}^{\infty} \frac{1}{n^{3.31700}}$ is much less than the lower bound found by studying the weights in the ternary tree.

One explaination could be that there is a bias in the weight contribution of smaller numbers. One can attempt to minimize this by removing the weights of the first k numbers from the inequality.

For $k = 16$ our ternary inequality becomes: $0.009098667 < \sum_{n=17}^{\infty} \frac{1}{3^{\sigma_\infty^T(n)}} < 0.02340484$.

But, $\sum_{n=17}^{\infty} \frac{1}{n^{3.31700}}$ still appears to be much less than the lower bound. Thus, these averages for $\sigma_\infty^T(n)$, if true, apply only for much larger numbers.

3.6 Decidability of range(ϕ)

First we verify that $range(\sigma_\infty)$ is decidable since the arguments are similar. Clearly $\sigma_\infty :\subseteq \mathbb{N} \to \mathbb{N}$ is partial computable: we just run the Collatz map keeping track of the number of iterations until 1 is reached, or we compute forver.

We let $\sigma_{\infty,j}(i)$ be the machine that returns $\sigma_\infty(i)$ if 1 is reached in j or less iterations of the Collatz map and 0 o.w. Then $\sigma_{\infty,j}(i) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is clearly total computable.

To decide whether $m \in range(\sigma_\infty)$,

> Let M be the machine that on input m computes:
>
> $\sigma_{\infty,m}(1), \sigma_{\infty,m}(2), ..., \sigma_{\infty,m}(2^m)$.
>
> If $\sigma_{\infty,m}(i) = m$ for some i, $1 \leq i \leq 2^m$, then return 1. Otherwise return 0. $\Diamond$

Then $f_M(m) = \chi_{range(\sigma_\infty)}(m)$, viz. $range(\sigma_\infty)$ is decidable.

The heart of the argument is that $\sigma_\infty(n) \geq log_2(n)$, thus the largest possible number that can have total stopping time m is 2^m. So, it suffices to check only 2^m cases up to m iterations of the Collatz map.

Theorem 3.4. $range(\phi)$ is decidable.

Proof. Let $\phi_m(n)$ be the machine that on input n computes the first m bits of $\phi(n)$'s binary output by appropriately applying the Collatz map to n. Then $\phi_m(n) : \mathbb{N} \times \mathbb{N} \to \Sigma^*$ is total computable.

Let M be the machine that on input $w \in \Sigma^*$ computes:

$\phi_{|w|}(1), \phi_{|w|}(2), ..., \phi_{|w|}(2^{|w|-2})$.

And checks whether $\phi_{|w|}(i) = w$ for some i, $1 \leq i \leq 2^{|w|-2}$.

If this is true return 1, otherwise return 0. $\Diamond$

(We only compute to $2^{|w|-2}$ since $|\phi(n)| = \sigma_\infty(n) + 2$.)

Then $f_M(w) = \chi_{range(\phi)}(w)$. I.e. $range(\phi)$ is decidable.

□

$range(\phi_{PR})$ is also decidable: use the same proof with ϕ_{PR} written in place of ϕ.

Thus, we can determine in finite time whether a binary word w is the code word provided by ϕ (or ϕ_{PR}) of some n under iterations of the Collatz map.

Before concluding this section I also include, for curiosities sake, a novel proof that $range(\sigma_\infty)$ is decidable using techniques from [3].

Proof. Let $a(n) := \begin{cases} \frac{1}{2^{\sigma_\infty(n)}}, & \sigma_\infty(n) \downarrow \\ 0, & o.w. \end{cases}$.

Then $a(n)$ is a computable sequence of real numbers. Since

Let N be the machine that on input n prints:

$\iota(\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{\sigma_{\infty,1}(n)}}))\iota(\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{\sigma_{\infty,2}(n)}}))...$

to the output. $\Diamond$

Where $\nu_{\mathbb{Q}}^{-1}(\frac{1}{2^{\sigma_{\infty,i}(n)}})$ is some effectively attainable code in $dom(\nu_{\mathbb{Q}})$ for $\frac{1}{2^{\sigma,i(n)}}$, $i \geq 1$.

Then since $|\frac{1}{2^{\sigma_{\infty,k}(n)}} - a(n)| \leq \frac{1}{2^k}$, $a(n) = \rho_C'''(f_N(n))$ for all $n \in \mathbb{N}$.

So since $a : \mathbb{N} \to \mathbb{R}$ is computable, by [3] (page 106), there exists a $b : \mathbb{N} \to \mathbb{R}$ computable and 1-1 such that $range(b) = range(a)$.

Since b is a computable sequence one can show that $d(N) = \sum_{n=1}^{N} \frac{1}{2^{b(n)}}$ is also a computable sequence. Thus by [3] theorem 4.2.3 (p102) $\lim_{N\to\infty} \sum_{n=1}^{N} \frac{1}{2^{b(n)}} = \sum_{n=1}^{\infty} \frac{1}{2^{b(n)}} = \sum_{m\in range(\sigma_\infty)} \frac{1}{2^m}$ is a computable real number.

And thus by [3] (p5, example 1.3.2) $range(\sigma_\infty)$ is decidable.

□

Both of these argument forms for demonstrating $range(\sigma_\infty)$ is decidable will work for proving more generally that the range of a computable function into $\mathbb{N}$ bounded below by some order function is decidable.

3.7 Applying Computable Martingales

In [1] a supermartingale $L : \Sigma^* \to \mathbb{R}^{\geq 0}$ is defined by the property $L(w0) + L(w1) \leq 2L(w)$.

In Part 1 of the thesis we adapted the definitions of [1] to the paradigms of [3] such that L is called *Nies-computable* if $U_L := \{(w, u) \in \Sigma^* \times \Sigma^* | \nu_{\mathbb{Q}}(u) < L(w)\}$ is decidable in $\Sigma^* \times \Sigma^*$.

Then a sequence $Z \in \Sigma^\omega$ is called computably random if no Nies-computable martingale L succeeds on Z. Where L is said to succeed on Z iff $\sup_n L(Z|n) = \infty$. Where $Z|n$ is the restriction of Z to its first n bits.

In an attempt to apply computable randomness to the $3x+1$ problem I extend the definition to:

Definition 3.4. A collection $\{a_k\}_{k\in\mathbb{N}}$ such that $a_k \in \Sigma^*$ is a *computably random collection* iff $\sup_k L(a_k) < \infty$ for each Nies-computable supermartingale L.

Then we can ask are the coded orbits a computably random collection? I.e. is $\sup_n L(\phi(n)) < \infty$ for every supermatingale L?

The answer is no. We simply let L be the martingale that bets half its capital on 0. I.e.

$L(\lambda) := 1$

$L(w0) := \frac{3}{2}L(w)$

$L(w1) := \frac{1}{2}L(w)$.

Then L is clearly computable. And

$$\begin{aligned}\sup_n L(\phi(n)) &\geq \sup_n L(\phi(2^n)) \\ &= \sup_n L(0^n 11) \\ &= \sup_n \left(\frac{3}{2}\right)^n \left(\frac{1}{2}\right)^2 \\ &= \infty.\end{aligned}$$

But of course there's nothing random about the orbits of 2^n. However, one could follow up this question by restricting to subsets A with more conspicuous orbits by asking: for $A \subseteq \mathbb{N}$ is $\sup_{n\in A} L(\phi(n)) < \infty$ for all Nies-computable supermartingales L? If A is decidable, but $\phi(A)$ is a computably random collection, this could be a compelling argument for the apparent intrinsic randomness of the behavior of the Collatz map on $\mathbb{N}$.

3.8 Questions

When the Collatz map is generalized such that the prime which divides n in the division step is > 2, the Kraft Inequality becomes more interesting because orbits can converge to 1 much faster than $log_2(n)$. Thus, there is a greater threat of divergence in the sum

of the graph weights which allows for an increased potential for negative results in the convergence of orbits to 1. So the first question is: what can these techniques teach us about the stopping times of the Collatz maps defined with different primes?

My next question is: is it possible to effectively approximate the value of $\sum_{n=1}^{\infty} \frac{1}{2^{|\phi(n)|}}$ from above to arbitrary precision by making unending progress in the removal of nonbranching nodes in the binary tree? This would imply that $\sum_{n=1}^{\infty} \frac{1}{2^{|\phi(n)|}}$ and $\sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(n)|}}$ are computable real numbers, and that ϕ^{-1} is a computable measure machine [1]. (Note that knowledge of the computability of $\sum_{n=1}^{\infty} \frac{1}{2^{|\sigma_\infty(n)|}}$ does not imply that the Collatz conjecture is true, since σ_∞ is not 1-1 see [3].)

Another possibility for analyzing the computable complexity of the orbits was simply to concatenate them into an infinite sequence. But then one can ask in what order shall they be listed without an arbitrary bias? This motivates the following question: Let $A = \{u_i | u_i \in \Sigma^*\}$, and let $A_\tau = u_{\tau(1)}u_{\tau(2)}u_{\tau(3)}... \in \Sigma^\omega$, such that τ is a computable bijection on $\mathbb{N}$. Is A a computably random collection iff A_τ is computably random?

4 APPENDIX

In [3] it is mentioned that the paradigms set forth in [2] (in particular see the chapter titled "Tape Machines") can be used to construct a machine which accomplishes the computability tasks in [3]. This appendix provides the details of such a machine. It is maintained that all informal machine descriptions in this thesis may be reduced to precise computations with the Turing machine as defined in this Appendix.

In [2], a machine is defined in terms of a flowchart which is a general 6-tuple that describes: a data set (which provides a snapshot of the current progress of a computation), a set of states, and a transition function that indicates how to change states and update the data set.

Definition 4.1. [2] A *flowchart* is a 6-tuple $F = (Q, D, \sigma, q_0, s, (q_1, ..., q_s))$ such that

(1) Q is a finite set (of states)

(2) D is a set (the data set)

(3) $q_0 \in Q$ (the initial state)

(4) $s \geq 1$ (the number of final states in F)

(5) $q_1, ...q_s \in Q$ such that $Q_e := \{q_1, ...q_s\}$, the set of final states, consists of distinct elements $q_1, ...q_s$.

(6) σ is a function (the transition function) which assigns to any $q \in Q \backslash \{q_1, ...q_s\}$ a tuple $\sigma(q) = (f, t, (p_1, ..., p_k)))$ for some $k \geq 1$ such that $f \subseteq D \to D, t :\subseteq D \to \{1, ..., k\}$, and $p_1, ..., p_k \in Q$ and $dom(f) = dom(t)$.

The function t is called a *test*, and indicates how to change state based on the current

configuration of the data set which is an element of D. The function f is used to alter the data set. This is described precisely through the semantics of the flowchart:

Definition 4.2. [2] Let $F = (Q, D, \sigma, q_0, s, (q_1, ..., q_s))$ be a flowchart.

(1) $CON := Q \times D$ is called the *set of configurations.* $\{q_1, ..., q_s\} \times D$ is called the *set of final configurations.*

(2) The *single step function* $SF :\subseteq CON \to CON$ is defined by the following cases:

(i) Case 1: $q \in Q_e$.

Then $SF(q, d) :=\uparrow$.

(ii) Case 2: $q \in Q \setminus Q_e$.

Suppose $\sigma(q) = (f, t, (p_1, ..., p_k))$. Then

Case 2.1: $d \notin dom(f)$.

Then $SF(q, d) :=\uparrow$.

Case 2.2: $d \in dom(f)$.

Then $SF(q, d) := (p_{t(d)}, f(d))$.

(3) The *computation time function* $CT :\subseteq CON \to \mathbb{N}$ is defined by

$$CT(q,d) := \begin{cases} \uparrow \text{ if for no } n \in \mathbb{N},\ SF^n(q,d) \text{ is a final configuration,} \\ \text{The unique } n \text{ such that } SF^{(n)}(q,d) \text{ is a final configuration o.w.} \end{cases}$$

(4) The *total step function* $TF :\subseteq CON \to CON$ is defined by

$$TF(q,d) := \begin{cases} SF^{(CT(q,d))}(q,d), \text{ if } (q,d) \in dom(CT) \\ \uparrow \text{ o.w.} \end{cases}$$

(5) The functions $f_F :\subseteq D \to D$ and $t_F :\subseteq D \to \{1, ..., s\}$, which provide the final configuration of the data set and the final state, are defined as: $f_F(d)$ exists $\iff$

$t_F(d)$ exists $\iff$ $TF(q_0, d)$ exists . And, $f_F(d) = d'$ and $t_F(d) = k$, if $TF(q_0, d) = (q_k, d')$.

The machine definition allows the flowchart to interact with an input and output set. It achieves this through two maps: the input encoding and output encoding. The input encoding encodes the input into the data set, the flowchart then acts on this data set, and then the output encoding encodes the output from the data set to the format of the output set. The machine is precisely defined as follows:

Definition 4.3. [2] A *machine* is a quintuple $M = (F, X, Y, IC, OC)$ such that for some data set D, (1)-(4) hold:

(1) F is a flowchart with data set D.

(2) X (the input set) and Y (the output set) are sets.

(3) $IC : X \to D$ is total (the input encoding).

(4) $OC : D \to Y$ is total (the output encoding).

Suppose F has s halting states, then to each machine M there is *the function computed by* M denoted $f_M :\subseteq X \to Y$, and the function $t_M :\subseteq X \to \{1, ..., s\}$ which determines the final halting state. These are defined by $f_M := OC \circ f_F \circ IC$ and $t_M := t_f \circ IC$.

Let $\Sigma = \{0, 1\}$ and $\Gamma = \{B\} \cup \Sigma$. In the following machine we will use $d : \mathbb{Z} \to \Gamma$ to represent our notion of a Turing tape. It stores the current configuration information of the tape and is updated by SF's calls to the transition function σ in the flowchart. The following notation is to reinforce this representation:

Definition 4.4. [2] $[d(-i)d(-i+1)...d(-1), d(0), d(1), ..., d(j)] := d$, if $d(k) = B$ whenever $k < -i$ or $k > j$.

$[d(-i)d(-i+1)...d(-1), d(0), d(1), d(2)...) := d$, if $d(k) = B$ whenever $k < -i$.

The comma indicates the location of the Turing tape head which is always at index 0 of the biinfinite sequence d.

I will now define the Turing machines that are the focus of this Appendix.

Definition 4.5. Let Let $\Sigma = \{0, 1\}$ and $\Gamma = \{B\} \cup \Sigma$. A *Turing machine* is a quintuple $M := (F, X, Y, IC, OC)$ such that:

(1) $X := (Y_1' \times ... \times Y_k')$ such that $Y_i' = \Sigma^*$ or $Y_i' = \Sigma^\omega$.

(2) $Y := Y_0 := \Sigma^*$ or Σ^ω.

(3) Let $\Gamma^{\mathbb{Z}} := \{d : \mathbb{Z} \to \Gamma\}$. Then $D := Y_0 \times Y_1 \times ... \times Y_k \times H_1 \times ... \times H_n$, where $Y_0, Y_1, ..., Y_k, H_1, ..., H_n = \Gamma^{\mathbb{Z}}$. Y_0 serves as the domain of the output tape configurations, $Y_1, ..., Y_k$ the domain of the k input tape configurations, and $H_1, ..., H_n$ the domain of the n work tape configurations.

(4) Let $IC : Y_1' \times ... \times Y_k' \to D$ be the input encoding such that $IC(y_1, ..., y_k) := ([\varnothing, B, \varnothing], [\varnothing, B, y_1), ..., [\varnothing, B, y_k), [\varnothing, B, \varnothing], ..., [\varnothing, B, \varnothing])$.

(5) $OC : D \to Y_0'$ such that $OC(d_0, d_1, ..., d_k, d_{k+1}, ..., d_{k+n}) = y_0$ where $d_0 = [v, a, \varnothing]$ and y_0 is the longest suffix of va such that such that $y_0 \in \Sigma^*$ (i.e. y_0 contains no B's).

(6) We restrict our flowchart transition function to either functions or tests of the following kinds: if $\sigma(q) = (f, t, p_1, ..., p_j)$ we allow two cases:

Case 1: $j = 1$, then $t(d) = 1$ and $f \in \{L_i, R_i, W_i\}$.

Case 2: $j = 2$, then $f(d) := d$ (i.e. f is the identity) and $t := \text{if}_i(a)$ for some $a \in \Gamma$ and $1 \leq i \leq k + n$.

Where

(6.1) $L_i(d_0, ..., d_{i-1}, d_i, d_{i+1}, ..., d_{k+n}) := (d_0, ..., d_{i-1}, d'_i, d_{i+1}, ..., d_{k+n})$ such that $d'_i(m) := d_i(m-1)$ (shift the tape left), if $i \neq 0$ (i.e. d_i is not the output tape which is not permitted to move left).

(6.2) $R_i(d_0, ..., d_{i-1}, d_i, d_{i+1}, ..., d_{k+n}) := (d_0, ..., d_{i-1}, d'_i, d_{i+1}, ..., d_{k+n})$ such that $d'_i(m) := d_i(m+1)$ (shift the tape right) where $0 \leq i \leq k+n$.

(6.3) $W_i(a)(d_0, ..., d_{i-1}, d_i, d_{i+1}, ..., d_{k+n}) := (d_0, ..., d_{i-1}, d'_i, d_{i+1}, ..., d_{k+n})$ such that

$$d'_i(m) := \begin{cases} d_i(m), \text{ if } m \neq 0 \\ a, \ m = 0 \end{cases}$$

(which writes a at the location of the tape head) where for $k < i \leq k+n$, $a \in \Gamma$; and for $i = 0$, $a \in \Sigma$. W_i is not defined for $1 \leq i \leq k$ since the input tapes may not be written to.

(6.4) $$\text{if}_i(a)(d_0, ..., d_i, ..., d_{k+n}) := \begin{cases} +, \ d_i(0) = a \\ -, \ d_i(0) \neq a \end{cases}$$

for all i, $0 \leq i \leq k+n$. And $+ := 1$ and $- := 2$, in this context, for convenience of later diagramming of the flowcharts.

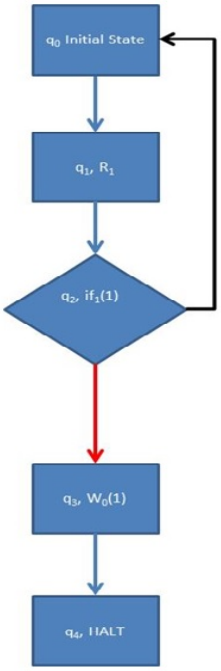
(7) We diverge now from the usual specification of a machine in the evaluation of the function computed by the Turing machine to allow for machines whose output is in Σ^ω. We say that for the function computed by the Turing machine $f_M :\subseteq X \to Y$, that $f_M(x) = y$ if

Case 1: $y \in \Sigma^*$. And $y = (OC \circ f_F \circ IC)(x)$ (the usual function computed by the machine works in this case).

Case 2: $y \in \Sigma^\omega$. For every $n \in \mathbb{N}$ there exists a $k \in \mathbb{N}$ such that $y|n \leq OC(\pi_2(SF^{(k)}(q_0, [\varnothing, B, \varnothing], IC(x), [\varnothing, B, \varnothing], ..., [\varnothing, B, \varnothing])))$ where q_0 is the initial state of F, $y|n \in \Sigma^*$ is the first n bits of y, and π_2 projects the second component of $SF^{(k)}(q_0, [\varnothing, B, \varnothing], IC(x), [\varnothing, B, \varnothing], ..., [\varnothing, B, \varnothing])$

(the output tape after k computation steps) into OC. I.e. $y = \lim_{k\to\infty} OC(\pi_2(SF^{(k)}(q_0, [\varnothing, B, \varnothing], IC(x), [\varnothing, B, \varnothing], ..., [\varnothing, B, \varnothing])))$. That is the computation proceeds forever but we can make arbitrary progress printing y to the output. Note that when $f_M(x) = y$ is defined, in this case, we must have $OC(\pi_2((SF^{(k)}(q_0, [\varnothing, B, \varnothing], IC(x), [\varnothing, B, \varnothing], ..., [\varnothing, B, \varnothing]))) \downarrow$ for all $k \geq 0$.

Example 4.1. The function $f :\subseteq \Sigma^\omega \to \Sigma^*$ such that $f(p) := \begin{cases} 1, \, p \neq 0^\omega \\ \uparrow, \, o.w. \end{cases}$ is computed by a Turing machine as defined above. We use zero work tapes, one input tape, and one output tape. The data set is $D = \Gamma^{\mathbb{Z}} \times \Gamma^{\mathbb{Z}}$. We can express the flowchart with a diagram:



Where the square represents a transition that is a function which alters the data set and a lozenge represents a transition that is a test. Arrows indicate the change in state as prescribed by the transition function of the flowchart where a red arrow corresponds to a test choosing $+$ and a black arrow corresponds to a test choosing $-$.

Specifically, following the definition of a Turing machine above we let:

$X = \Sigma^\omega$, $Y = \Sigma^*$, $D = \Gamma^{\mathbb{Z}} \times \Gamma^{\mathbb{Z}}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $Q_e = \{q_4\}$, and let f be the identity and $t(d) = 1$ for all $d \in D$. Our transition function is given by:

$\sigma(q_0) = (f, t, q_1)$,

$\sigma(q_1) = (R_1, t, q_2)$,

$\sigma(q_2) = (f, \text{if}_1(1), q_3)$,

$\sigma(q_3) = (W_0(1), t, q_4)$.

So, for example

$$\begin{aligned} f_M(0^5 10^\omega) :&= (OC \circ f_F \circ IC)(0^5 10^\omega) \\ &= OC \circ f_F([\varnothing, 0, 0^4 10^\omega)) \\ &= OC(\pi_2(TF(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega)))). \end{aligned}$$

And since $SF(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega)) = (q_1, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega)$,

we have

$$\begin{aligned} SF^{(2)}(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega)) &= SF(q_1, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega)) \\ &= (q_2, [\varnothing, B, \varnothing], [0, 0, 0^3 10^\omega)). \end{aligned}$$

Thus, $SF^{(10)}(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega) = (q_2, [\varnothing, B, \varnothing], [0^5, 1, 0^\omega))$.

And, $SF^{(11)}(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega) = (q_3, [\varnothing, B, \varnothing], [0^5, 1, 0^\omega))$,

so $SF^{(12)}(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega) = (q_4, [\varnothing, 1, \varnothing], [0^5, 1, 0^\omega))$.

Thus, since q_4 is a final state, $TF(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega) = (q_4, [\varnothing, 1, \varnothing], [0^5, 1, 0^\omega))$.

So,

$$f_M(0^5 10^\omega) = OC(\pi_2(TF(q_0, [\varnothing, B, \varnothing], [\varnothing, 0, 0^4 10^\omega))))$$

$= OC([\varnothing, 1, \varnothing])$

$= 1$, as required.

BIBLIOGRAPHY

1. Nies, Andre. Computability and Randomness. Oxford ; New York: Oxford UP, 2009. Print. Oxford Logic Guides ; 51.

2. Weihrauch, Klaus. Computability, volume 9 of EATCS Monographs on Theoretical Computer Science, 1987.

3. Weihrauch, Klaus. Computable Analysis: an introduction. Springer Science and Business Media, 2012.

4. Rogers, Hartley. "Theory of recursive functions and effective computability." (1987).

5. Soare, R. I. Recursively Enumerable Sets and Degrees : A Study of Computable Functions and Computably Generated Sets. Berlin ; New York: Springer-Verlag, 1987. Print. Perspectives in Mathematical Logic.

6. Soare, Robert Irving. "Formalism and intuition in computability." Phil. Trans. R. Soc. A 370.1971 (2012): 3277-3304.

7. Odifreddi, Piergiorgio. Classical recursion theory: The theory of functions and sets of natural numbers. Vol. 125. Elsevier, 1992.

8. Davis, Martin. Computability and Unsolvability. New York: McGraw-Hill, 1958. Print. McGraw-Hill Ser. in Information Processing and Computers.

9. Davis, Martin, Ron Sigal, and Elaine J. Weyuker. Computability, complexity, and languages: fundamentals of theoretical computer science. Newnes, 1994.

10. Calude, Cristian S. Information and randomness: an algorithmic perspective. Springer Science and Business Media, 2013.

11. Schnorr, Claus-Peter. "A unified approach to the definition of random sequences." Theory of Computing Systems 5.3 (1971): 246-258.

12. Li, Ming., and VitaÌĄnyi, P. M. B. An Introduction to Kolmogorov Complexity and Its Applications. 3rd ed. New York: Springer, 2008. Print. Texts in Computer Science.

13. Downey, Rodney G., and Denis R. Hirschfeldt. Algorithmic randomness and complexity. Springer Science and Business Media, 2010.

14. Lagarias, Jeffrey C., ed. The ultimate challenge: The 3x+ 1 problem. American Mathematical Soc., 2010.

15. Andrei, Stefan, and Cristian Masalagiu. "About the Collatz conjecture." Acta Informatica 35.2 (1998): 167-179.

16. Colussi, Livio. "The convergence classes of Collatz function." Theoretical Computer Science 412.39 (2011): 5409-5419.

17. Ren, Wei. Induction and Code for Collatz Conjecture or 3x+1 Problem. Preprint viXra: vixra.org/pdf/1609.0373v1.pdf

18. Scollo, Giuseppe. "ω-rewriting the Collatz problem." Fundamenta Informaticae 64.1-4 (2005): 405-416.