

Quantum Random Self-Modifiable Computation

Michael Stephen Fiske

December 31, 2018

Abstract

Among the fundamental questions in computer science, at least two have a deep impact on mathematics. What can computation compute? How many steps does a computation require to solve an instance of the 3-SAT problem? Our work addresses the first question, by introducing a new model called the *ex-machine*. The ex-machine executes Turing machine instructions and two special types of instructions. *Quantum random instructions* are physically realizable with a quantum random number generator. *Meta instructions* can add new states and add new instructions to the ex-machine. A countable set of ex-machines is constructed, each with a finite number of states and instructions; each ex-machine can compute a Turing incomputable language, whenever the quantum randomness measurements behave like unbiased Bernoulli trials. In 1936, Alan Turing posed the halting problem for Turing machines and proved that this problem is unsolvable for Turing machines. Consider an enumeration $\mathcal{E}_a(i) = (\mathfrak{M}_i, T_i)$ of all Turing machines \mathfrak{M}_i and initial tapes T_i . Does there exist an ex-machine \mathfrak{X} that has at least one evolutionary path $\mathfrak{X} \rightarrow \mathfrak{X}_1 \rightarrow \mathfrak{X}_2 \rightarrow \dots \rightarrow \mathfrak{X}_m$, so at the m th stage ex-machine \mathfrak{X}_m can correctly determine for $0 \leq i \leq m$ whether \mathfrak{M}_i 's execution on tape T_i eventually halts? We demonstrate an ex-machine $\mathfrak{Q}(x)$ that has one such evolutionary path. The existence of this evolutionary path suggests that David Hilbert was not misguided to propose in 1900 that mathematicians search for finite processes to help construct mathematical proofs. Our refinement is that we cannot use a fixed computer program that behaves according to a fixed set of mechanical rules. We must pursue methods that exploit randomness and self-modification so that the complexity of the program can increase as it computes.

Contents

1	Introduction	2
1.1	Related Work – Computation	3
1.2	Related Work – Quantum Randomness	4
2	The Ex-Machine	5
2.1	Standard Instructions	5
2.2	Quantum Random Instructions	7
2.3	Meta Instructions	10
3	Quantum Randomness	14
3.1	Mathematical Determinism and Unpredictability	15
3.2	Quantum Random Theory	16
4	Computing Ex-Machine Languages	20

5	Some $\Omega(x)$ Observations Based on Cantor and Gödel	31
6	An Ex-Machine Halting Problem	32
6.1	Some Observations Based on Theorem 6.1	34
7	Two Research Problems	36
8	Appendix – Execution of the Collatz Machine on $n = 5$	44

1 Introduction

Consider two fundamental questions in computer science:

1. What can a computing machine compute?
2. How many computational steps does a computational machine require to solve an instance of the 3-SAT problem? The 3-SAT problem [27] is the basis for the famous $P \stackrel{?}{=} NP$ problem [26].

These two questions are usually studied with the assumption that the Turing machine (TM) [96] is the standard model of computation [30, 45, 62, 68, 77, 88].

We introduce a new computational model, called the *ex-machine*, and reexamine the first question. The ex-machine model bifurcates the first question into two questions. *What is computation? What can computation compute?* An ex-machine computation adds two special types of instructions to the Turing machine instructions. The name ex-machine — derived from the latin *extra machinam* — was chosen because ex-machine computation generates new dynamical behaviors that one may no longer recognize as a machine.

The *meta instruction* is one type of special instruction. When an ex-machine executes a meta instruction, the meta instruction can add new states and add new instructions or replace instructions. The meta instruction enables the complexity [87, 82] of an ex-machine to increase during its execution, unlike a typical machine (e.g., the inclined plane, lever, pulley, wedge, wheel and axle, Archimedean screw, Galilean telescope or bicycle).

The *quantum random instruction* is the other special instruction. It can be physically realized with a quantum random number generator [44, 63, 71, 76, 95, 92, 93, 100]. Due to the *quantum random instructions*, the execution behavior of two ex-machines may be distinct, even though the two ex-machines start their execution with the same input on the tape, the same instructions, the same initial states, and so on. Two distinct identical ex-machines may exhibit different execution behaviors even when started with identical initial conditions. When this property of the quantum random instructions is combined with the appropriate use of meta instructions, two identical machines with the same initial conditions can evolve to two different ex-machines as the execution of each respective machine proceeds.

Some of the ex-machine programs provided here compute beyond the Turing barrier. A countable set of ex-machines are explicitly defined. Every one of these ex-machines can evolve to compute a Turing incomputable language with probability measure 1, whenever the quantum random measurements (trials) behave like unbiased Bernoulli trials. (A Turing machine cannot compute a Turing incomputable language.)

In 1936, Alan Turing posed the halting problem and proved that the halting problem for Turing machines is unsolvable by a Turing machine [30, 68, 77, 96]. Consider the ex-machine halting problem: Given an enumeration $\mathcal{E}_a(i) = (\mathfrak{M}_i, T_i)$ of all Turing machines \mathfrak{M}_i and initial tapes T_i , each finitely bounded and containing only blank symbols outside the bounds, does there exist an ex-machine \mathfrak{X} that has at least one evolutionary path $\mathfrak{X} \rightarrow \mathfrak{X}_1 \rightarrow \mathfrak{X}_2 \rightarrow \dots \rightarrow \mathfrak{X}_m$, so at stage m , the ex-machine \mathfrak{X}_m can correctly determine for $0 \leq i \leq m$ whether \mathfrak{M}_i 's execution

on tape T_i eventually halts? We demonstrate an ex-machine $\mathfrak{Q}(x)$ that has one such evolutionary path. At stage m , the self-modifying ex-machine's evolutionary path $\mathfrak{Q}(h_{\mathcal{E}_a}(0) \ x) \rightarrow \mathfrak{Q}(h_{\mathcal{E}_a}(0) \ h_{\mathcal{E}_a}(1) \ x) \rightarrow \dots \mathfrak{Q}(h_{\mathcal{E}_a}(0) \ h_{\mathcal{E}_a}(1) \ \dots \ h_{\mathcal{E}_a}(m) \ x)$ has used a finite amount of computational resources and measured a finite amount of quantum randomness.

Consider the Goldbach Conjecture [50] and the Riemann Hypothesis [75], which are both famous, unsolved math problems. Each of these problems can be expressed as an instance of Turing's halting problem with a particular Turing machine. (See machine instructions 3 and [97].) A large scale, physical realization of an ex-machine and further research might present an opportunity to study these mathematical problems and other difficult problems with new computational and conceptual tools.

1.1 Related Work – Computation

The rest of the introduction discusses some related results on computation using quantum randomness, and the theory of quantum randomness. Some related work on computation is in [41] and [43]. In [41], a parallel computational machine, called the active element machine, uses its meta commands and quantum randomness to construct a computational procedure that behaves like a quantum black box. Using quantum randomness as a source of unpredictability and the meta commands to self-modify the active element machine, this procedure emulates a universal Turing machine so that an outside observer is unable to comprehend what Turing machine instructions are being executed by the emulation of the universal Turing machine.

In [43], based on a Turing machine's states and alphabet symbols, a transformation ϕ was defined from the Turing machine's instructions to a finite number of affine functions in the two dimensional plane $\mathbb{Q} \times \mathbb{Q}$, where \mathbb{Q} is the rational numbers. Now for the details: let states $Q = \{q_1, \dots, q_{|Q|}\}$, alphabet $A = \{a_1, \dots, a_{|A|}\}$, a halt state h that is not in Q , and program $\eta : Q \times A \rightarrow Q \cup \{h\} \times A \times \{-1, +1\}$ be a Turing machine. This next part defines a one-to-one mapping ϕ from Turing program η to a finite set of affine functions, whose domain is a bounded subset of $\mathbb{Q} \times \mathbb{Q}$. Set $B = |A| + |Q| + 1$. Define symbol value function $\nu : \{h\} \cup Q \cup A \rightarrow \mathbb{N}$ as $\nu(h) = 0$, $\nu(a_i) = i$ and $\nu(q_i) = i + |A|$.

T_k is the alphabet symbol in the k th tape square. ϕ maps right computational step $\eta(q, T_k) = (r, \alpha, +1)$ to the affine function $f(x, y) = \left(Bx - B^2\nu(T_k), \frac{1}{B}y + B\nu(r) + \nu(\alpha) - \nu(q) \right)$. During this computational step, state q moves to state r . Alphabet symbol α replaces T_k on tape square k , and the tape head moves to tape square $k + 1$, one square to the right.

Similarly, ϕ maps left computational step $\eta(q, T_k) = (r, \alpha, -1)$ to the affine function $g(x, y) = \left(\frac{1}{B}x + B\nu(T_{k-1}) + \nu(\alpha) - \nu(T_k), By + B\nu(r) - B^2\nu(q) - B\nu(T_{k-1}) \right)$. ϕ maps machine configuration $(q, k, T) \in Q \times \mathbb{Z} \times A^{\mathbb{Z}}$ to the point $\phi(q, k, T) = \left(\sum_{j=-1}^{\infty} \nu(T_{k+j+1})B^{-j}, B\nu(q) + \sum_{j=0}^{\infty} \nu(T_{k-j-1})B^{-j} \right)$ in the $\mathbb{Q} \times \mathbb{Q}$ plane. Point $\phi(q, k, T)$ is in $\mathbb{Q} \times \mathbb{Q}$ because of the finitely bounded tape condition for Turing machines: only a finite number of tape squares contain non-blank symbols, so the tail of each infinite sum is a geometric series.

Each affine function's domain is a subset of some unit square $\{(x, y) \in \mathbb{Q} \times \mathbb{Q} : m \leq x \leq m + 1 \text{ and } n \leq y \leq n + 1\}$, where m and n are integers. Via the ϕ transformation, a finitely bounded initial tape and initial state of the Turing machine are mapped to an initial point with rational coordinates in one of the unit squares. Hence, ϕ transforms Turing's halting problem to the following dynamical systems problem. If machine configuration (q, k, T) halts after n computational steps, then the orbit of $\phi(q, k, T)$ exits one of the unit squares on the n th iteration. If machine configuration (r, j, S) is immortal (i.e., never halts), then the orbit of $\phi(r, j, S)$ remains in these finite number of unit squares forever.

Dynamical system $\frac{dx}{dt} = F(x, y)$ and $\frac{dy}{dt} = G(x, y)$ is *autonomous* if the independent variable t does not appear in F and G . A discrete, autonomous dynamical system is comprised of a function $f : X \rightarrow X$, where X is a

topological space and the orbits $\mathcal{O}(f, p) = \{f^k(p) : p \in X \text{ and } k \in \mathbb{N}\}$ are studied.

Consider the following augmentation of the discrete, autonomous dynamical system (f, X) . After the 1st iteration, f is perturbed to f_1 where $f \neq f_1$ and after the second iteration f_1 is perturbed to f_2 so that $f_2 \neq f_1$ and $f_2 \neq f$ and so on where $f_i \neq f_j$ for all $i \neq j$. Then the dynamical system $(f_1, f_2, \dots, f_k, \dots, X)$ is a *discrete, non-autonomous* dynamical system [40].

For a particular Turing machine, set X equal to the union of all the unit squares induced by ϕ and define f based on the finite number of affine functions, resulting from the ϕ transformation. In terms of dynamical systems theory, the ϕ transformation shows that *each Turing machine is a discrete, autonomous dynamical system*. In [43], we stated that an active element machine using quantum randomness was a non-autonomous dynamical system capable of generating non-Turing computational behaviors; however, no new specific machines exhibiting novel behaviors were provided, except for a reference to procedure 2 in [41]. In this sense, our research is a continuation of [41, 43], but arguably provides a more transparent computational model for studying what can be computed and sheds more light on Turing's halting problem.

1.2 Related Work – Quantum Randomness

Some other related work pertains to the theory of quantum randomness. The classic EPR paper [36] presented a paradox that led Einstein, Podolsky and Rosen to conclude that quantum mechanics is an incomplete theory and should be supplemented with additional variables. They believed that the statistical predictions of quantum mechanics were correct, but only as a consequence of the statistical distributions of these hidden variables. Moreover, they believed that the specification of these hidden variables could predetermine the result of measuring any observable of the system.

Due to an ambiguity in the EPR argument, Bohr [8] explained that no paradox or contradiction can be derived from the assumption that Schrodinger's wave function [83] contains a complete description of physical reality. Namely, in the quantum theory, it is impossible to control the interaction between the object being observed and the measurement apparatus. Per Heisenberg's uncertainty principle [52], momentum is transferred between them during position measurements, and the object is displaced during momentum measurements. Based on the link between the wave function and the probability amplitude, first proposed by Born [10], Bohr's response set the stage for the problem of hidden variables and the development of quantum mechanics as a statistical scientific theory.

In [6], Bohm and Aharanov advocated a Stern-Gerlach magnet [46, 47, 48] example to address the hidden variables problem. Using a gedankenexperiment [7] of Bohm, Bell showed that no local hidden variable theory can reproduce all of the statistical predictions of quantum mechanics and maintain local realism [4]. Clauser, Horne, Shimony and Holt derived a new form of Bell's inequality [24], called the CHSH inequality, along with a proposed physically-realizable experiment to test their inequality.

In [85], their experiment tests the CHSH inequality. Using entangled photon pairs, their experiment found a loophole-free [61] violation of local realism. They estimated the degree to which a local realistic system could predict their measurement choices, and obtained a smallest adjusted p -value equal to 2.3×10^{-7} . Hence, they rejected the hypothesis that local realism governed their experiment. Recently, a quantum randomness expander has been constructed, based on the CHSH inequality [71].

By taking into account the algebraic structure of quantum mechanical observables, Kochen and Specker [58] provided a proof for the nonexistence of hidden variables. In [95], Svozil proposed three criteria for building quantum random number generators based on beam splitters: (A) Have three or more mutually exclusive outcomes correspond to the invocation of Hilbert spaces with dimension at least 3; (B) Use pure states in conjugated bases for preparation and detection; (C) Use entangled singlet (unique) states to eliminate bias.

By extending the theory of Kochen and Specker, Calude and Svozil developed an initial Turing incomputable

theory of quantum randomness [13] — applicable to beam splitters — that has been recently advanced further by Abbott, Calude, and Svozil [14, 15, 16]. A more comprehensive summary of their work will be provided in section 3.

2 The Ex-Machine

We introduce a *quantum random, self-modifiable machine* that adds two special types of instructions to the Turing machine [96]. Before the quantum random and meta instructions are defined, we present some preliminary notation, the standard instructions, and a Collatz machine example.

\mathbb{Z} denotes the integers. \mathbb{N} and \mathbb{N}^+ are the non-negative and positive integers, respectively. The finite set $Q = \{0, 1, 2, \dots, n-1\} \subset \mathbb{N}$ represents the ex-machine states. This representation of the ex-machine states helps specify how new states are added to Q when a meta instruction is executed. Let $\mathfrak{A} = \{a_1, \dots, a_n\}$, where each a_i represents a distinct symbol. The set $A = \{0, 1, \#\} \cup \mathfrak{A}$ consists of alphabet (tape) symbols, where $\#$ is the blank symbol and $\{0, 1, \#\} \cap \mathfrak{A}$ is the empty set. In some ex-machines, $A = \{0, 1, \#, Y, N, \mathbf{a}\}$, where $a_1 = Y$, $a_2 = N$, $a_3 = \mathbf{a}$. In some ex-machines, $A = \{0, 1, \#\}$, where \mathfrak{A} is the empty set. The alphabet symbols are read from and written on the tape. The ex-machine tape T is a function $T : \mathbb{Z} \rightarrow A$ with an initial condition: before the ex-machine starts executing, there exists an $N > 0$ so that $T(k) = \#$ when $|k| > N$. In other words, before the ex-machine starts executing, all tape squares contain blank symbols, except for a finite number of tape squares. When this initial condition holds for tape T , we say that tape T is *finitely bounded*.

2.1 Standard Instructions

Definition 2.1. Execution of Standard Instructions

The standard ex-machine instructions \mathcal{S} satisfy $\mathcal{S} \subset Q \times A \times Q \times A \times \{-1, 0, 1\}$ and a uniqueness condition: If $(q_1, \alpha_1, r_1, a_1, y_1) \in \mathcal{S}$ and $(q_2, \alpha_2, r_2, a_2, y_2) \in \mathcal{S}$ and $(q_1, \alpha_1, r_1, a_1, y_1) \neq (q_2, \alpha_2, r_2, a_2, y_2)$, then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$. A standard instruction $I = (q, a, r, \alpha, y)$ is similar to a Turing machine tuple [30, 74, 96]. When the ex-machine is in state q and the tape head is scanning alphabet symbol $a = T(k)$ at tape square k , instruction I is executed as follows:

- The ex-machine state moves from state q to state r .
- The ex-machine replaces alphabet symbol a with alphabet symbol α so that $T(k) = \alpha$. The rest of the tape remains unchanged.
- If $y = -1$, the ex-machine moves its tape head one square to the left on the tape and is subsequently scanning the alphabet symbol $T(k-1)$ in tape square $k-1$.
- If $y = +1$, the ex-machine moves its tape head one square to the right on the tape and is subsequently scanning the alphabet symbol $T(k+1)$ in tape square $k+1$.
- If $y = 0$, the ex-machine does not moves its tape head and is subsequently scanning the alphabet symbol $T(k) = \alpha$ in tape square k .

Remark 2.1. A Turing machine [96] has a finite set of states Q , a finite alphabet A , a finitely bounded tape, and a finite set of standard ex-machine instructions that are executed according to definition 2.1. In other words, an ex-machine that uses only standard instructions is computationally equivalent to a Turing machine. Hence, an ex-machine with only standard instructions will be called a standard machine or a Turing machine.

The Collatz conjecture has an interesting relationship to Turing's halting problem, which will be discussed further in section 7. Furthermore, there is a generalization of the Collatz function that is unsolvable for a standard machine [25].

Definition 2.2. Collatz Conjecture

Define the Collatz function $f : \mathbb{N}^+ \rightarrow \mathbb{N}^+$, where $f(n) = \frac{n}{2}$ when n is even and $f(n) = 3n + 1$ when n is odd. Zero iterations of f is $f^0(n) = n$. k iterations of f is represented as $f^k(n)$. The orbit of n with respect to f is $\mathcal{O}(f, n) = \{f^k(n) : k \in \mathbb{N}\}$. Observe that $f(5) = 16$, $f^2(5) = 8$, $f^3(5) = 4$, $f^4(5) = 2$, $f^5(5) = 1$, so $\mathcal{O}(f, 5) = \{5, 16, 8, 4, 2, 1\}$. The Collatz conjecture states that for any positive integer n , $\mathcal{O}(f, n)$ contains 1.

We specify a Turing machine that for each n computes the orbit $\mathcal{O}(f, n)$. The standard machine halts if the orbit $\mathcal{O}(f, n)$ contains 1. Set $A = \{0, 1, \#, E\}$. Set $Q = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, p, q\}$ where $a = 0$, $b = 1$, $c = 2, \dots, n = 13$, $p = 14$, and $q = 15$.

Machine instructions 1 shows a list of standard instructions that compute $\mathcal{O}(f, n)$. The initial tape is $\# \# 1^n \#$, where it is understood that the remaining tape squares, beyond the leftmost $\#$ and rightmost $\#$, contain only blank symbols. The space means the tape head is scanning the $\#$ adjacent to the leftmost 1. The initial state is q .

Machine Instructions 1. Collatz Machine

```
;; Comments follow two semicolons.
(q, #, a, #, 1)
(q, 0, p, 0, 1)
(q, 1, p, 1, 1)

(a, #, p, #, 1)
(a, 0, p, 0, 1)
(a, 1, b, 1, 1)

(b, #, h, #, -1) ;; Valid halt # 1#. The Collatz orbit reached 1.
(b, 0, p, 0, 1)
(b, 1, c, 1, 1)

(c, #, e, #, -1)
(c, 0, p, 0, 1)
(c, 1, d, 1, 1)

(d, #, k, #, -1)
(d, 0, p, 0, 1)
(d, 1, c, 1, 1)

;; n / 2 computation
(e, #, g, #, 1)
(e, 0, p, 0, 1)
(e, 1, f, 0, -1)

(f, #, g, #, 1)
(f, 0, p, 0, 1)
(f, 1, f, 1, -1)

(g, #, j, #, -1)
(g, 0, g, 1, 1)
(g, 1, i, #, 1)

(i, #, p, #, 1)
(i, 0, e, 0, -1)
(i, 1, i, 1, 1)

(j, #, a, #, 1)
(j, 0, p, 0, 1)
(j, 1, j, 1, -1)
```

```

;; 3n + 1 computation
(k, #, n, #, 1)
(k, 0, k, 0, -1)
(k, 1, l, 0, 1)

(l, #, m, 0, 1)
(l, 0, l, 0, 1)
(l, 1, p, 1, 1)

(m, #, k, 0, -1)
(m, 0, p, 0, 1)
(m, 1, p, 1, 1)

;; Start n / 2 computation
(n, #, f, 0, -1)
(n, 0, n, 1, 1)
(n, 1, p, 1, 1)

;; HALT with ERROR. Alphabet symbol E represents an error.
(p, #, h, E, 0)
(p, 0, h, E, 0)
(p, 1, h, E, 0)

```

With input $\# \#1^n\#$, the execution of the Collatz machine halts (i.e., moves to the halting state h) if the orbit $\mathcal{O}(f, n)$ reaches 1. Below shows the Collatz machine executing the first ten instructions with initial tape $\# \#11111\#$ and initial state q . Each row shows the current tape and machine state after the instruction in that row has been executed. The complete execution of the Collatz machine is shown in the appendix 8. It computes $\mathcal{O}(f, 5)$.

STATE	TAPE	TAPE HEAD	INSTRUCTION	COMMENT
a	## 11111#####	1	(q, #, a, #, 1)	
b	##1 1111#####	2	(a, 1, b, 1, 1)	
c	##11 111#####	3	(b, 1, c, 1, 1)	
d	##111 11#####	4	(c, 1, d, 1, 1)	
c	##1111 1#####	5	(d, 1, c, 1, 1)	
d	##11111 #####	6	(c, 1, d, 1, 1)	
k	##1111 1#####	5	(d, #, k, #, -1)	Compute $3*5 + 1$
l	##11110 #####	6	(k, 1, l, 0, 1)	
m	##111100 #####	7	(l, #, m, 0, 1)	
k	##11110 00###	6	(m, #, k, 0, -1)	

2.2 Quantum Random Instructions

The quantum random instructions \mathcal{R} are subsets of $Q \times A \times Q \times \{-1, 0, 1\} = \{(q, a, r, y) : q, r \text{ are in } Q \text{ and } a \text{ in } A \text{ and } y \text{ in } \{-1, 0, 1\}\}$ that satisfy a uniqueness condition defined below.

Definition 2.3. Execution of Quantum Random Instructions

The quantum random instructions \mathcal{R} satisfy $\mathcal{R} \subset Q \times A \times Q \times \{-1, 0, 1\}$ and the following uniqueness condition: If $(q_1, \alpha_1, r_1, y_1) \in \mathcal{R}$ and $(q_2, \alpha_2, r_2, y_2) \in \mathcal{R}$ and $(q_1, \alpha_1, r_1, y_1) \neq (q_2, \alpha_2, r_2, y_2)$, then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$. When the tape head is scanning alphabet symbol a and the ex-machine is in state q , the quantum random instruction (q, a, r, y) executes as follows:

- The ex-machine measures a quantum random source that returns a random bit $b \in \{0, 1\}$. (It is assumed that the quantum measurements satisfy unbiased Bernoulli trial axioms 1 and 2.)
- On the tape, alphabet symbol a is replaced with random bit b .
(This is why A always contains both symbols 0 and 1.)

- The ex-machine state changes to state r .
- The ex-machine moves its tape head left if $y = -1$, right if $y = +1$, or the tape head does not move if $y = 0$.

Repeated independent trials are called *quantum random Bernoulli trials* [37] if there are only two possible outcomes for each trial (i.e., quantum random measurement) and the probability of each outcome remains constant for all trials. *Unbiased* means the probability of both outcomes is the same. Below are the formal definitions.

Axiom 1. *Unbiased Trials.*

Consider the bit sequence $(x_1x_2\dots)$ in the infinite product space $\{0,1\}^{\mathbb{N}}$. A single outcome x_i of a bit sequence $(x_1x_2\dots)$ generated by quantum randomness is unbiased. The probability of measuring a 0 or a 1 are equal: $P(x_i = 1) = P(x_i = 0) = \frac{1}{2}$.

Axiom 2. *Stochastic Independence.*

History has no effect on the next quantum random measurement. Each outcome x_i is independent of the history. No correlation exists between previous or future outcomes. This is expressed in terms of the conditional probabilities: $P(x_i = 1 \mid x_1 = b_1, \dots, x_{i-1} = b_{i-1}) = \frac{1}{2}$ and $P(x_i = 0 \mid x_1 = b_1, \dots, x_{i-1} = b_{i-1}) = \frac{1}{2}$ for each $b_i \in \{0,1\}$.

In order to not detract from the formal description of the ex-machine, section 3 provides a physical basis for the axioms and a discussion of quantum randomness.

Machine instructions 2 lists a random walk machine that has only standard instructions and quantum random instructions. Alphabet $A = \{0, 1, \#, E\}$. The states are $Q = \{0, 1, 2, 3, 4, 5, 6, h\}$, where the halting state $h = 7$. A valid initial tape contains only blank symbols; that is, $\# \#\#$. The valid initial state is 0.

There are three quantum random instructions: $(0, \#, 0, 0)$, $(1, \#, 1, 0)$ and $(4, \#, 4, 0)$. The random instruction $(0, \#, 0, 0)$ is executed first. If the quantum random source measures a 1, the machine jumps to state 4 and the tape head moves to the right of tape square 0. If the quantum random source measures a 0, the machine jumps to state 1 and the tape head moves to the left of tape square 0. Instructions containing alphabet symbol E provide error checking for an invalid initial tape or initial state; in this case, the machine halts with an error.

Machine Instructions 2. Random Walk

```
;; Comments follow two semicolons.
(0, #, 0, 0)
(0, 0, 1, 0, -1)
(0, 1, 4, 1, 1)

;; Continue random walk to the left of tape square 0
(1, #, 1, 0)
(1, 0, 1, 0, -1)
(1, 1, 2, #, 1)

(2, 0, 3, #, 1)
(2, #, h, E, 0)
(2, 1, h, E, 0)

;; Go back to state 0. Numbers of random 0's = number of random 1's.
(3, #, 0, #, -1)

;; Go back to state 1. Numbers of random 0's > number of random 1's.
(3, 0, 1, 0, -1)
(3, 1, h, E, 0)
```

```

;; Continue random walk to the right of tape square 0
(4, #, 4, 0)
(4, 1, 4, 1, 1)
(4, 0, 5, #, -1)

(5, 1, 6, #, -1)
(5, #, h, E, 0)
(5, 0, h, E, 0)

;; Go back to state 0. Numbers of random 0's = number of random 1's.
(6, #, 0, #, 1)

;; Go back to state 4. Numbers of random 1's > number of random 0's.
(6, 1, 4, 1, 1)

(6, 0, h, E, 0)

```

Below are 31 computational steps of the ex-machine's first execution. This random walk machine never halts when the initial tape is blank and the initial state is 0. The first quantum random instruction executed is (0, #, 0, 0). The quantum random source measured a 0, so the execution of this instruction is shown as (0, #, 0, 0_{qr}, 0). The second quantum random instruction executed is (1, #, 1, 0). The quantum random source measured a 1, so the execution of instruction (1, #, 1, 0) is shown as (1, #, 1, 1_{qr}, 0).

1st Execution of Random Walk Machine. Computational Steps 1-31.

STATE	TAPE	TAPE HEAD	INSTRUCTION
0	##### 0##	0	(0, #, 0, 0 _{qr} , 0)
1	#### #0##	-1	(0, 0, 1, 0, -1)
1	#### 10##	-1	(1, #, 1, 1 _{qr} , 0)
2	##### 0##	0	(1, 1, 2, #, 1)
3	##### ##	1	(2, 0, 3, #, 1)
0	##### ###	0	(3, #, 0, #, -1)
0	##### 0##	0	(0, #, 0, 0 _{qr} , 0)
1	#### #0##	-1	(0, 0, 1, 0, -1)
1	#### 00##	-1	(1, #, 1, 0 _{qr} , 0)
1	### #00##	-2	(1, 0, 1, 0, -1)
1	### 000##	-2	(1, #, 1, 0 _{qr} , 0)
1	## #000##	-3	(1, 0, 1, 0, -1)
1	## 1000##	-3	(1, #, 1, 1 _{qr} , 0)
2	### 000##	-2	(1, 1, 2, #, 1)
3	##### 00##	-1	(2, 0, 3, #, 1)
1	### #00##	-2	(3, 0, 1, 0, -1)
1	### 100##	-2	(1, #, 1, 1 _{qr} , 0)
2	##### 00##	-1	(1, 1, 2, #, 1)
3	##### 0##	0	(2, 0, 3, #, 1)
1	##### #0##	-1	(3, 0, 1, 0, -1)
1	##### 10##	-1	(1, #, 1, 1 _{qr} , 0)
2	##### 0##	0	(1, 1, 2, #, 1)
3	##### ##	1	(2, 0, 3, #, 1)
0	##### ###	0	(3, #, 0, #, -1)
0	##### 0##	0	(0, #, 0, 0 _{qr} , 0)
1	#### #0##	-1	(0, 0, 1, 0, -1)
1	#### 00##	-1	(1, #, 1, 0 _{qr} , 0)
1	### #00##	-2	(1, 0, 1, 0, -1)
1	### 000##	-2	(1, #, 1, 0 _{qr} , 0)
1	## #000##	-3	(1, 0, 1, 0, -1)
1	## 1000##	-3	(1, #, 1, 1 _{qr} , 0)

Below are the first 31 steps of the ex-machine's second execution. The first quantum random instruction executed is (0, #, 0, 0). The quantum random bit measured was 1, so the result of this instruction is shown as (0, #, 0, 1_{qr}, 0). The second quantum random instruction executed is (1, #, 1, 0), which measured a 0, so the result of this instruction is shown as (1, #, 1, 0_{qr}, 0).

2nd Execution of Random Walk Machine. Computational Steps 1-31.

STATE	TAPE	TAPE HEAD	INSTRUCTION
0	## 1#####	0	(0, #, 0, 1_qr, 0)
4	##1 #####	1	(0, 1, 4, 1, 1)
4	##1 0####	1	(4, #, 4, 0_qr, 0)
5	## 1#####	0	(4, 0, 5, #, -1)
6	# #####	-1	(5, 1, 6, #, -1)
0	## #####	0	(6, #, 0, #, 1)
0	## 1#####	0	(0, #, 0, 1_qr, 0)
4	##1 #####	1	(0, 1, 4, 1, 1)
4	##1 1####	1	(4, #, 4, 1_qr, 0)
4	##11 #####	2	(4, 1, 4, 1, 1)
4	##11 1###	2	(4, #, 4, 1_qr, 0)
4	##111 ###	3	(4, 1, 4, 1, 1)
4	##111 1##	3	(4, #, 4, 1_qr, 0)
4	##1111 ##	4	(4, 1, 4, 1, 1)
4	##1111 0#	4	(4, #, 4, 0_qr, 0)
5	##111 1##	3	(4, 0, 5, #, -1)
6	##11 1###	2	(5, 1, 6, #, -1)
4	##111 ###	3	(6, 1, 4, 1, 1)
4	##111 0##	3	(4, #, 4, 0_qr, 0)
5	##11 1###	2	(4, 0, 5, #, -1)
6	##1 1####	1	(5, 1, 6, #, -1)
4	##11 #####	2	(6, 1, 4, 1, 1)
4	##11 0###	2	(4, #, 4, 0_qr, 0)
5	##1 1####	1	(4, 0, 5, #, -1)
6	## 1#####	0	(5, 1, 6, #, -1)
4	##1 #####	1	(6, 1, 4, 1, 1)
4	##1 0####	1	(4, #, 4, 0_qr, 0)
5	## 1#####	0	(4, 0, 5, #, -1)
6	# #####	-1	(5, 1, 6, #, -1)
0	## #####	0	(6, #, 0, #, 1)
0	## 0#####	0	(0, #, 0, 0_qr, 0)
1	# #0#####	-1	(0, 0, 1, 0, -1)

The first and second executions of the random walk ex-machine verify our statement in the introduction: in contrast with the Turing machine, the execution behavior of the same ex-machine may be distinct at two different instances, even though each instance of the ex-machine starts its execution with the same input on the tape, the same initial states and same initial instructions. Hence, the ex-machine is a discrete, non-autonomous dynamical system.

2.3 Meta Instructions

Meta instructions are the second type of special instructions. The execution of a meta instruction enables the ex-machine to self-modify its instructions. This means that an ex-machine's meta instructions can add new states, add new instructions or replace instructions. Formally, the meta instructions \mathcal{M} satisfy $\mathcal{M} \subset \{(q, a, r, \alpha, y, J) : q \in Q \text{ and } r \in Q \cup \{|Q|\} \text{ and } a, \alpha \in A \text{ and instruction } J \in \mathcal{S} \cup \mathcal{R}\}$.

Define $\mathcal{I} = \mathcal{S} \cup \mathcal{R} \cup \mathcal{M}$, as the set of standard, quantum random, and meta instructions. To help describe how a meta instruction modifies \mathcal{I} , the *unique state, scanning symbol condition* is defined: for any two distinct instructions chosen from \mathcal{I} at least one of the first two coordinates must differ. More precisely, all 6 of the following uniqueness conditions must hold.

1. If $(q_1, \alpha_1, r_1, \beta_1, y_1)$ and $(q_2, \alpha_2, r_2, \beta_2, y_2)$ are both in \mathcal{S} , then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$.
2. If $(q_1, \alpha_1, r_1, \beta_1, y_1) \in \mathcal{S}$ and $(q_2, \alpha_2, r_2, \beta_2, y_2) \in \mathcal{R}$ or vice versa, then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$.
3. If $(q_1, \alpha_1, r_1, y_1)$ and $(q_2, \alpha_2, r_2, y_2)$ are both in \mathcal{R} , then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$.
4. If $(q_1, \alpha_1, r_1, y_1) \in \mathcal{R}$ and $(q_2, \alpha_2, r_2, a_2, y_2, J_2) \in \mathcal{M}$ or vice versa, then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$.
5. If $(q_1, \alpha_1, r_1, \beta_1, y_1) \in \mathcal{S}$ and $(q_2, \alpha_2, r_2, a_2, y_2, J_2) \in \mathcal{M}$ or vice versa, then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$.
6. If $(q_1, \alpha_1, r_1, a_1, y_1, J_1)$ and $(q_2, \alpha_2, r_2, a_2, y_2, J_2)$ are both in \mathcal{M} , then $q_1 \neq q_2$ or $\alpha_1 \neq \alpha_2$.

Before a valid machine execution starts, it is assumed that the standard, quantum random and meta instructions $\mathcal{S} \cup \mathcal{R} \cup \mathcal{M}$ always satisfy the unique state, scanning symbol condition. This condition assures that there is no ambiguity on what instruction should be executed when the machine is in state q and is scanning tape symbol a . Furthermore, the execution of a meta instruction preserves this uniqueness condition.

Definition 2.4. Execution of Meta Instructions

A meta instruction (q, a, r, α, y, J) in \mathcal{M} is executed as follows.

- The first five coordinates (q, a, r, α, y) are executed as a standard instruction according to definition 2.1 with one caveat. State q may be expressed as $|Q| - c_1$ and state r may be expressed as $|Q|$ or $|Q| - c_2$, where $0 < c_1, c_2 \leq |Q|$. When (q, a, r, α, y) is executed, if q is expressed as $|Q| - c_1$, the value of q is instantiated to the current value of $|Q|$ minus c_1 . Similarly, if r is expressed as $|Q|$ or $|Q| - c_2$, the value of state r is instantiated to the current value of $|Q|$ or $|Q|$ minus c_2 , respectively.
- Subsequently, instruction J modifies \mathcal{I} , where instruction J has one of the two forms: $J = (q, a, r, \alpha, y)$ or $J = (q, a, r, y)$.
- For both forms, if $\mathcal{I} \cup \{J\}$ still satisfies the unique state, scanning symbol condition, then \mathcal{I} is updated to $\mathcal{I} \cup \{J\}$.
- Otherwise, there is an instruction I in \mathcal{I} whose first two coordinates q, a , are equal to instruction J 's first two coordinates. In this case, instruction J replaces instruction I in \mathcal{I} . That is, \mathcal{I} is updated to $\mathcal{I} \cup \{J\} - \{I\}$.

In regard to definition 2.4, example 1 shows how instruction I is added to \mathcal{I} and how new states are instantiated and added to Q .

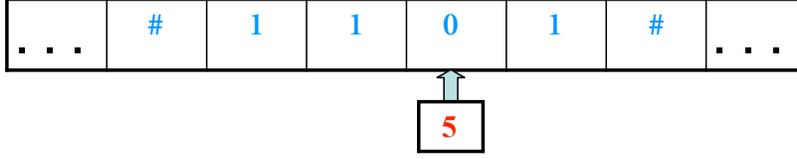
Example 1. Adding New States

Consider the meta instruction $(q, a_1, |Q| - 1, \alpha_1, y_1, J)$, where $J = (|Q| - 1, a_2, |Q|, \alpha_2, y_2)$. After the standard instruction $(q, a_1, |Q| - 1, \alpha_1, y_1)$ is executed, this meta instruction adds one new state $|Q|$ to the machine states Q and also adds the instruction J , instantiated with the current value of $|Q|$. Figure 1 shows the execution of this meta instruction for the specific values $Q = \{0, 1, 2, 3, 4, 5, 6, 7\}$, $A = \{\#, 0, 1\}$, $q = 5$, $a_1 = 0$, $\alpha_1 = 1$, $y_1 = 0$, $a_2 = 1$, $\alpha_2 = \#$, and $y_2 = -1$. States and alphabet symbols are shown in red and blue, respectively.

States $Q = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Alphabet $A = \{\#, 0, 1\}$.

Meta Instruction $(5, 0, |Q| - 1, 1, 0, J)$ where

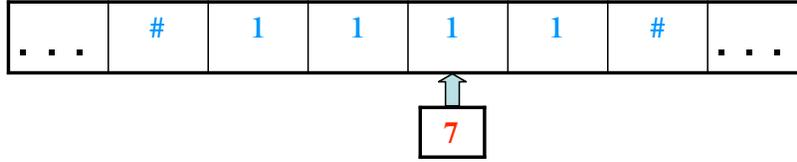
$$J = (|Q| - 1, 1, |Q|, \#, -1).$$



Instruction $(5, 0, 7, 1, 0)$ is executed, since $|Q| = 8$.

$J = (7, 1, 8, \#, -1)$ is added to the standard instructions.

States Q are updated to $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$.



States $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. Alphabet $A = \{\#, 0, 1\}$.

Instruction $J = (7, 1, 8, \#, -1)$ is executed.

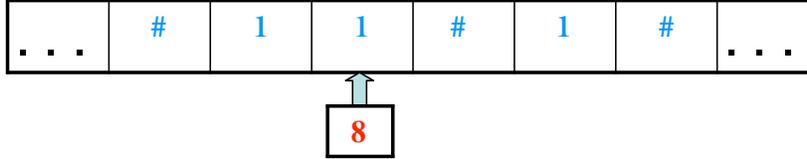


Figure 1: Meta Instruction Execution

Let \mathfrak{X} be an ex-machine. The instantiation of $|Q| - 1$ and $|Q|$ in a meta instruction I invokes *self-reflection* about \mathfrak{X} 's current number of states, at the moment when \mathfrak{X} executes I . This simple type of self-reflection poses no obstacles in physical realizations. In particular, a LISP implementation [69] along with quantum random bits measured from [100] simulates all executions of the ex-machines provided herein.

Definition 2.5. Simple Meta Instructions

A simple meta instruction has one of the forms $(q, a, |Q| - c_2, \alpha, y)$, $(q, a, |Q|, \alpha, y)$, $(|Q| - c_1, a, r, \alpha, y)$, $(|Q| - c_1, a, |Q| - c_2, \alpha, y)$, $(|Q| - c_1, a, |Q|, \alpha, y)$, where $0 < c_1, c_2 \leq |Q|$. The expressions $|Q| - c_1$, $|Q| - c_2$ and $|Q|$ are instantiated to a state based on the current value of $|Q|$ when the instruction is executed.

In this paper, ex-machines only self-reflect with the symbols $|Q| - 1$ and $|Q|$.

Example 2. Execution of Simple Meta Instructions

Let $A = \{0, 1, \#\}$ and $Q = \{0\}$. ex-machine \mathfrak{X} has 3 simple meta instructions.

$(|Q|-1, \#, |Q|-1, 1, 0)$
 $(|Q|-1, 1, |Q|, 0, 1)$
 $(|Q|-1, 0, |Q|, 0, 0)$

With an initial blank tape and starting state of 0, the first four computational steps are shown below. In the first step, \mathfrak{X} 's tape head is scanning a $\#$ and the ex-machine state is 0. Since $|Q| = 1$, simple meta instruction $(|Q|-1, \#, |Q|-1, 1, 0)$ instantiates to $(0, \#, 0, 1, 0)$, and executes.

STATE	TAPE	TAPE HEAD	INSTRUCTION	NEW INSTRUCTION
0	## 1###	0	$(0, \#, 0, 1, 0)$	$(0, \#, 0, 1, 0)$
1	##0 ###	1	$(0, 1, 1, 0, 1)$	$(0, 1, 1, 0, 1)$
1	##0 1##	1	$(1, \#, 1, 1, 0)$	$(1, \#, 1, 1, 0)$
2	##00 ##	2	$(1, 1, 2, 0, 1)$	$(1, 1, 2, 0, 1)$

In the second step, the tape head is scanning a 1 and the state is 0. Since $|Q| = 1$, instruction $(|Q|-1, 1, |Q|, 0, 1)$ instantiates to $(0, 1, 1, 0, 1)$, executes and updates $Q = \{0, 1\}$. In the third step, the tape head is scanning a $\#$ and the state is 1. Since $|Q| = 2$, instruction $(|Q|-1, \#, |Q|-1, 1, 0)$ instantiates to $(1, \#, 1, 1, 0)$ and executes. In the fourth step, the tape head is scanning a 1 and the state is 1. Since $|Q| = 2$, instruction $(|Q|-1, 1, |Q|, 0, 1)$ instantiates to $(1, 1, 2, 0, 1)$, executes and updates $Q = \{0, 1, 2\}$. During these four steps, two simple meta instructions create four new instructions and add new states 1 and 2.

Definition 2.6. Finite Initial Conditions

An ex-machine is said to have finite initial conditions if the following four conditions are satisfied before the ex-machine starts executing.

1. The number of states $|Q|$ is finite.
2. The number of alphabet symbols $|A|$ is finite.
3. The number of machine instructions $|\mathcal{I}|$ is finite.
4. The tape is finitely bounded.

It may be useful to think about the initial conditions of an ex-machine as analogous to the boundary value conditions of a differential equation. While trivial to verify, the purpose of remark 2.2 is to assure that computations performed with an ex-machine are physically plausible.

Remark 2.2. Finite Initial Conditions

If the machine starts its execution with finite initial conditions, then after the machine has executed l instructions for any positive integer l , the current number of states $Q(l)$ is finite and the current set of instructions $\mathcal{I}(l)$ is finite. Also, the tape T is still finitely bounded and the number of measurements obtained from the quantum random source is finite.

Proof. The remark follows immediately from definition 2.6 of finite initial conditions and machine instruction definitions 2.1, 2.3, and 2.4. In particular, the execution of one meta instruction adds at most one new instruction and one new state to Q . \square

Definition 2.7 defines new ex-machines that may have evolved from computations of prior ex-machines that have halted. The notion of evolving is useful because the quantum random and meta instructions can self-modify an ex-machine's instructions as it executes. In contrast with the ex-machine, after a Turing machine halts, its instructions have not changed.

This difference motivates the next definition, which is illustrated by the following. Consider an initial ex-machine \mathfrak{X}_0 that has 9 initial states and 15 initial instructions. \mathfrak{X}_0 starts executing on a finitely bounded tape T_0 and halts. When the ex-machine halts, it (now called \mathfrak{X}_1) has 14 states and 24 instructions and the current tape is S_1 . We say that ex-machine \mathfrak{X}_0 with tape T_0 *evolves to* ex-machine \mathfrak{X}_1 with tape S_1 .

Definition 2.7. Evolving an ex-machine

Let $T_0, T_1, T_2 \dots T_{i-1}$ each be a finitely bounded tape. Consider ex-machine \mathfrak{X}_0 with finite initial conditions. \mathfrak{X}_0 starts executing with tape T_0 and evolves to ex-machine \mathfrak{X}_1 with tape S_1 . Subsequently, \mathfrak{X}_1 starts executing with tape T_1 and evolves to \mathfrak{X}_2 with tape S_2 . This means that when ex-machine \mathfrak{X}_1 starts executing on tape T_1 , its instructions are preserved after the halt with tape S_1 . The ex-machine evolution continues until \mathfrak{X}_{i-1} starts executing with tape T_{i-1} and evolves to ex-machine \mathfrak{X}_i with tape S_i . One says that ex-machine \mathfrak{X}_0 with finitely bounded tapes $T_0, T_1, T_2 \dots T_{i-1}$ evolves to ex-machine \mathfrak{X}_i after i halts.

When ex-machine \mathfrak{X}_0 evolves to \mathfrak{X}_1 and subsequently \mathfrak{X}_1 evolves to \mathfrak{X}_2 and so on up to ex-machine \mathfrak{X}_n , then ex-machine \mathfrak{X}_i is called an *ancestor* of ex-machine \mathfrak{X}_j whenever $0 \leq i < j \leq n$. Similarly, ex-machine \mathfrak{X}_j is called a *descendant* of ex-machine \mathfrak{X}_i whenever $0 \leq i < j \leq n$. The sequence of ex-machines $\mathfrak{X}_0 \rightarrow \mathfrak{X}_1 \rightarrow \dots \rightarrow \mathfrak{X}_n \dots$ is called an *evolutionary path*.

3 Quantum Randomness

On a first reading, one may choose to skip this section, by assuming that there is adequate physical justification for axioms 1 and 2. Overall, the ex-machine uses quantum randomness as a computational tool. Hence, part of our goal was to use axioms 1 and 2 for our quantum random instructions, because the axioms are supported by the empirical evidence of various quantum random number generators [1, 5, 60, 63, 76, 92, 93, 100, 101]. In practice, however, a physical implementation of a quantum random number generator can only generate a finite amount of data and only a finite number of statistical tests can be performed on the data. Due to these limitations, one goal of quantum random theory [71, 14, 15, 16, 95], besides general understanding, is to certify the mathematical properties, assumed about actual quantum random number generators, and assure that the theory is a reasonable extension of quantum mechanics [8, 9, 10, 51, 52, 53, 83, 84, 36, 4, 24, 58].

We believe it is valuable to reach both a pragmatic (experimental) and theoretical viewpoint. In pure mathematics, the formal system and the logical steps in the mathematical proofs need to be checked. In our situation, it is possible for a mathematical theory of quantum randomness (or for that matter any theory in physics) to be consistent (i.e., in the sense of mathematics) and have valid mathematical proofs, yet the theory still does not adequately model the observable properties of the underlying physical reality. If just one subtle mistake or oversight is made while deriving a mathematical formalism from some physical assumptions, then the mathematical conclusions arrived at – based on the theory – may represent physical nonsense. Due to the infinite nature of randomness, this branch of science is faced with the challenging situation that the mathematical properties of randomness can only be provably tested with an infinite amount of experimental data and an infinite number of tests. Since we only have the means to collect a finite amount of data and perform a finite amount of statistical tests, we must acknowledge that experimental tests on a quantum random number generator, designed according to a mathematical / physical theory, can only falsify the theory [73] for that class of quantum random number

generators. As more experiments are performed, successful statistical tests calculated on longer sequences of random data may strengthen the empirical evidence, but they cannot scientifically prove the theory. This paragraph provides at least some motivation for some of our pragmatic points, described in the next three paragraphs.

In sections 4 and 6, the mathematical proofs rely upon the property that for any m , all 2^m binary strings are equally likely to occur when a quantum random number generator takes m binary measurements.¹ In terms of the ex-machine computation performed, how one of these binary strings is generated from some particular type of quantum process is not the critical issue.

Furthermore, most of the 2^m binary strings have high Kolmogorov complexity [89, 59, 20]. This fact leads to the following mathematical intuition that enables new computational behaviors: the execution of quantum random instructions working together with meta instructions enables the ex-machine to increase its program complexity [87, 82] as it evolves. In some cases, the increase in program complexity can increase the ex-machine's computational power as the ex-machine evolves. Also, notice the distinction here between the program complexity of the ex-machine and Kolmogorov complexity. The definition of Kolmogorov complexity only applies to standard machines. Moreover, the program complexity (e.g., the Shannon complexity $|Q||A|$ [87]) stays fixed for standard machines. In contrast, an ex-machine's program complexity can increase without bound, when the ex-machine executes quantum random and meta instructions that productively work together. (For example, see ex-machine 1, called $\mathfrak{Q}(x)$.)

With this intuition about complexity in mind, we provide a concrete example. Suppose the quantum random generator demonstrated in [60], certified by the strong Kochen-Specker theorem [95, 14, 15, 16], outputs the 100-bit string $a_0a_1 \dots a_{99} = 101100001010111100110011001110001000111001010101101111000000010011001000011010101101111001101010000$ to ex-machine \mathfrak{X}_1 .

Suppose a distinct quantum random generator using radioactive decay [76] outputs the same 100-bit string $a_0a_1 \dots a_{99}$ to a distinct ex-machine \mathfrak{X}_2 . Suppose \mathfrak{X}_1 and \mathfrak{X}_2 have identical programs with the same initial tapes and same initial state. Even though radioactive decay [28, 29, 78, 79, 80] was discovered over 100 years ago and its physical basis is still phenomenological, the execution behavior of \mathfrak{X}_1 and \mathfrak{X}_2 are indistinguishable for the first 100 executions of their quantum random instructions. In other words, ex-machines \mathfrak{X}_1 and \mathfrak{X}_2 exhibit execution behaviors that are independent of the quantum process that generated these two identical binary strings.

3.1 Mathematical Determinism and Unpredictability

Before some of the deeper theory on quantum randomness is reviewed, we take a step back to view randomness from a broader theoretical perspective. While we generally agree with the philosophy of Eagle [35] that *randomness is unpredictability*, example 3 helps sharpen the differences between *indeterminism* and *unpredictability*.

Example 3. A Mathematical Gedankenexperiment

Our gedankenexperiment demonstrates a deterministic system which exhibits an extreme amount of unpredictability. Some work is needed to define the dynamical system and summarize its mathematical properties before we can present the gedankenexperiment.

Consider the quadratic map $f : \mathbb{R} \rightarrow \mathbb{R}$, where $f(x) = \frac{9}{2}x(1-x)$. Set $I_0 = [0, \frac{1}{3}]$ and $I_1 = [\frac{1}{3}, \frac{2}{3}]$. Set $B = (\frac{1}{3}, \frac{2}{3})$. Define the set $\Lambda = \{x \in [0, 1] : f^n(x) \in I_0 \cup I_1 \text{ for all } n \in \mathbb{N}\}$. 0 is a fixed point of f and $f^2(\frac{1}{3}) = f^2(\frac{2}{3}) = 0$, so the boundary points of B lie in Λ . Furthermore, whenever $x \in B$, then $f(x) < 0$ and $\lim_{n \rightarrow \infty} f^n(x) = -\infty$. This means all orbits that exit Λ head off to $-\infty$.

¹One has to be careful not to misinterpret quantum random axioms 1 and 2. For example, the Champernowne sequence 01 00 01 10 11 000 001 010 011 100 101 110 111 0000 ... is sometimes cited as a sequence that is Borel normal, yet still Turing computable. However, based on the mathematics of random walks [37], the Champernowne sequence catastrophically fails the expected number of changes of sign as $n \rightarrow \infty$. Since all 2^m strings are equally likely, the expected value of changes of sign follows from the reflection principle and simple counting arguments, as shown in III.5 of [37].

The inverse image $f^{-1}(B)$ is two open intervals $B_0 \subset I_0$ and $B_1 \subset I_1$ such that $f(B_0) = f(B_1) = B$. Topologically, B_0 behaves like Cantor's open middle third of I_0 and B_1 behaves like Cantor's open middle third of I_1 . Repeating the inverse image indefinitely, define the set $H = B \cup \bigcup_{k=1}^{\infty} f^{-k}(B)$. Now $H \cup \Lambda = [0, 1]$ and $H \cap \Lambda = \emptyset$.

Using dynamical systems notation, set $\Sigma_2 = \{0, 1\}^{\mathbb{N}}$. Define the shift map $\sigma : \Sigma_2 \rightarrow \Sigma_2$, where $\sigma(a_0 a_1 \dots) = (a_1 a_2 \dots)$. For each x in Λ , x 's trajectory in I_0 and I_1 corresponds to a unique point in Σ_2 : define $h : \Lambda \rightarrow \Sigma_2$ as $h(x) = (a_0 a_1 \dots)$ such that for each $n \in \mathbb{N}$, set $a_n = 0$ if $f^n(x) \in I_0$ and $a_n = 1$ if $f^n(x) \in I_1$.

For any two points $(a_0 a_1 \dots)$ and $(b_0 b_1 \dots)$ in Σ_2 , define the metric $d((a_0 a_1 \dots), (b_0 b_1 \dots)) = \sum_{i=0}^{\infty} |a_i - b_i| 2^{-i}$. Via the standard topology on \mathbb{R} inducing the subspace topology on Λ , it is straightforward to verify that h is a homeomorphism from Λ to Σ_2 . Moreover, $h \circ f = \sigma \circ h$, so h is a topological conjugacy. The set H and the topological conjugacy h enable us to verify that Λ is a Cantor set. This means that Λ is uncountable, totally disconnected, compact and every point of Λ is a limit point of Λ .

We are ready to pose our *mathematical gedankenexperiment*. We make the following assumption about our mathematical observer. When our observer takes a physical measurement of a point x in Λ , she measures a 0 if x lies in I_0 and measures a 1 if x lies in I_1 . We assume that she cannot make her observation any more accurate based on our idealization that is analogous to the following: measurements at the quantum level have limited resolution due to the wavelike properties of matter [33, 34, 10, 51, 52, 83, 39]. Similarly, at the second observation, our observer measures a 0 if $f(x)$ lies in I_0 and 1 if $f(x)$ lies in I_1 . Our observer continues to make these observations until she has measured whether $f^{k-1}(x)$ is in I_0 or in I_1 . Before making her $k+1$ st observation, can our observer make an effective prediction whether $f^k(x)$ lies in I_0 or I_1 that is correct for more than 50% of her predictions?

The answer is no when $h(x)$ is a generic point (i.e., in the sense of Lebesgue measure) in Σ_2 . Set \mathcal{R} to the Martin-Löf random points in Σ_2 . Then \mathcal{R} has Lebesgue measure 1 in Σ_2 [37, 21], so its complement $\Sigma_2 - \mathcal{R}$ has Lebesgue measure 0. For any x such that $h(x)$ lies in \mathcal{R} , then our observer cannot predict the orbit of x with a Turing machine. Hence, via the topological conjugacy h , we see that for a generic point x in Λ , x 's orbit between I_0 and I_1 is Martin-Löf random – even though f is mathematically deterministic and f is a Turing computable function.

Overall, the dynamical system (f, Λ) is mathematically deterministic and each real number x in Λ has a definite value. However, due to the lack of resolution in the observer's measurements, the orbit of generic point x is unpredictable – in the sense of Martin-Löf random.

3.2 Quantum Random Theory

A deeper theory on quantum randomness stems from the seminal EPR paper [36]. Einstein, Podolsky and Rosen propose a necessary condition for a complete theory of quantum mechanics: *Every element of physical reality must have a counterpart in the physical theory*. Furthermore, they state that elements of physical reality must be found by the results of experiments and measurements. While mentioning that there might be other ways of recognizing a physical reality, EPR propose the following as a reasonable criterion for a complete theory of quantum mechanics:

If, without in any way disturbing a system, one can predict with certainty (i.e., with probability equal to unity) the value of a physical quantity, then there exists an element of physical reality corresponding to this physical quantity.

They consider a quantum-mechanical description of a particle, having one degree of freedom. After some analysis, they conclude that a *definite value* of the coordinate, for a particle in the state given by $\psi = e^{\frac{2\pi i}{h} p_0 x}$, is

not predictable, but may be obtained only by a direct measurement. However, such a measurement disturbs the particle and changes its state. They remind us that in quantum mechanics, *when the momentum of the particle is known, its coordinate has no physical reality*. This phenomenon has a more general mathematical condition that if the operators corresponding to two physical quantities, say A and B , do not commute, then a precise knowledge of one of them precludes a precise knowledge of the other. Hence, EPR reach the following conclusion:

(I) The quantum-mechanical description of physical reality given by the wave function is not complete.

OR

(II) When the operators corresponding to two physical quantities (e.g., position and momentum) do not commute (i.e. $AB \neq BA$), the two quantities cannot have the same simultaneous reality.

EPR justifies this conclusion by reasoning that if both physical quantities had a simultaneous reality and consequently definite values, then these definite values would be part of the complete description. Moreover, if the wave function provides a complete description of physical reality, then the wave function would contain these definite values and the definite values would be predictable.

From their conclusion of I OR II, EPR assumes the negation of I – that the wave function does give a complete description of physical reality. They analyze two systems that interact over a finite interval of time. And show by a thought experiment of measuring each system via wave packet reduction that it is possible to assign two different wave functions to the same physical reality. Upon further analysis of two wave functions that are eigenfunctions of two non-commuting operators, they arrive at the conclusion that two physical quantities, with non-commuting operators can have simultaneous reality. From this contradiction or paradox (depending on one's perspective), they conclude that the quantum-mechanical description of reality is not complete.

In [8], Bohr responds to the EPR paper. Via an analysis involving single slit experiments and double slit (two or more) experiments, Bohr explains how during position measurements that momentum is transferred between the object being observed and the measurement apparatus. Similarly, Bohr explains that during momentum measurements the object is displaced. Bohr also makes a similar observation about time and energy: *“it is excluded in principle to control the energy which goes into the clocks without interfering essentially with their use as time indicators”*. Because at the quantum level it is impossible to control the interaction between the object being observed and the measurement apparatus, Bohr argues for a *“final renunciation of the classical ideal of causality”* and a *“radical revision of physical reality”*.

From his experimental analysis, Bohr concludes that the meaning of EPR's expression *without in any way disturbing the system* creates an ambiguity in their argument. Bohr states: *“There is essentially the question of an influence on the very conditions which define the possible types of predictions regarding the future behavior of the system. Since these conditions constitute an inherent element of the description of any phenomenon to which the term physical reality can be properly attached, we see that the argumentation of the mentioned authors does not justify their conclusion that quantum-mechanical description is essentially incomplete.”* Overall, the EPR versus Bohr-Born-Heisenberg position set the stage for the problem of hidden variables in the theory of quantum mechanics.

The Kochen-Specker [90, 58] approach – to a hidden variable theory in quantum mechanics – is addressed independently of any reference to locality or non-locality [4]. Instead, they assume a stronger condition than locality: hidden variables are only associated with the quantum system being measured. No hidden variables are associated with the measurement apparatus. This is the physical (non-formal) notion of *non-contextuality*.

In their theory, a set of observables are defined, where in the case of quantum mechanics, the observables (more general) are represented by the self-adjoint operators on a separable Hilbert space. The Kochen-Specker theorem [90, 58] proves that it is impossible for a non-contextual hidden variable theory to assign values to finite sets of

observables, which is also consistent with the theory of quantum mechanics. More precisely, the Kochen-Specker theorem demonstrates a contradiction between the following two assumptions:

- (\mathcal{A}_1) The set of observables in question have pre-assigned definite values. Due to complementarity, the observables may not be all simultaneously co-measurable, where the formal definition of co-measurable means that the observables commute.
- (\mathcal{A}_2) The measurement outcomes of observables are non-contextual. This means the outcomes are independent of the other co-measurable observables that are measured along side them, along with the requirement that in any “complete” set of mutually co-measurable yes-no propositions, exactly one proposition should be assigned the value “yes”.

Making assumption \mathcal{A}_2 more precise, the mutually co-measurable yes-no propositions are represented by mutually orthogonal projectors spanning the Hilbert space.

The Kochen-Specker theorem does not explicitly identify the observables that violate at least one of the assumptions \mathcal{A}_1 or \mathcal{A}_2 . The original Kochen-Specker theorem was not developed with a goal of locating the particular observable(s) that violate assumptions \mathcal{A}_1 or \mathcal{A}_2 .

In [14], Abbott, Calude and Svozil (ACS) advance beyond the Kochen-Specker theory, but also preserve the quantum logic formalism of von Neumann [98, 99] and Kochen-Specker [56, 57]. Their work can be summarized as follows:

1. They explicitly formalize the physical notions of value definiteness (indefiniteness) and contextuality.
2. They sharpen in what sense the Kochen-Specker and Bell-like theorems imply the violation of the non-contextuality assumption \mathcal{A}_2 .
3. They provide collections of observables that do not produce Kochen-Specker contradictions.
4. They propose the reasonable statement that *quantum random number generators² should be certified by value indefiniteness, based on the particular observables utilized for that purpose*. Hence, their extension of the Kochen-Specker theory needs to locate the violations of non-classicality.

The key intuition for quantum random number generators that are designed according to the ACS protocol is that value indefiniteness implies unpredictability. One of the primary strengths of ACS theory over the Kochen-Specker and Bell-type theorems is that it helps identify the particular observables that are value indefinite. The identification of value indefinite observables helps design a quantum random number generator, based on mathematics with reasonable physical assumptions rather than ad hoc arguments. In particular, their results assure that, if quantum mechanics is correct, the particular observables – used in the measurements that produce the random number sequences – are provably value indefinite. In order for a quantum random number generator to capture the value indefiniteness during its measurements, their main idea is to identify pairs of projection observables that satisfy the following: if one of the projection observables is assigned the value 1 by an admissible assignment function such that the projector observables \mathcal{O} are non-contextual, then the other observable in the pair must be value indefinite.

² There are quantum random number expanders [71, 85], based on the Bell inequalities [4, 24] and non-locality. We believe random number expanders are better suited for cryptographic applications where an active adversary may be attempting to subvert the generation of a cryptographic key. In this paper, advancing cryptography is not one of our goals.

Now, some of their formal definitions are briefly reviewed to clarify how the ACS corollary helps design a protocol for a dichotomic quantum random bit generator, operating in a three-dimensional Hilbert space. Vectors $|\psi\rangle$ lie in the Hilbert space \mathbb{C}^n , where \mathbb{C} is the complex numbers. The *observables* $\mathcal{O} \subset \{P_\psi : |\psi\rangle \in \mathbb{C}^n\}$ are a non-empty subset of the projection operators $P_\psi = \frac{|\psi\rangle\langle\psi|}{\langle\psi|\psi\rangle}$, that project onto the linear subspace of \mathbb{C}^n , spanned by non-zero vector $|\psi\rangle$. They only consider pure states. The set of *measurement contexts* over \mathcal{O} is the set $\mathcal{C} \subset \{\{P_1, P_2, \dots, P_n\} \mid P_i \in \mathcal{O} \text{ and } \langle i|j\rangle = 0 \text{ for } i \neq j\}$. A *context* $C \in \mathcal{C}$ is a maximal set of *compatible projection observables*, where compatible means the observables can be simultaneously measured.

Let $\nu : \{(o, C) \mid o \in \mathcal{O}, C \in \mathcal{C} \text{ and } o \in C\} \xrightarrow{\nu} \{0, 1\}$ be a partial function. For some $o, o' \in \mathcal{O}$ and $C, C' \in \mathcal{C}$, then $\nu(o, C) = \nu(o', C')$ means both $\nu(o, C)$ and $\nu(o', C')$ are defined and they have equal values in $\{0, 1\}$. The expression $\nu(o, C) \neq \nu(o', C')$ means either $\nu(o, C)$ or $\nu(o', C')$ are not defined or they are both defined but have different values. ν is called an *assignment function* and ν formally expresses the notion of a hidden variable. ν specifies in advance the result obtained from the measurement of an observable.

An observable $o \in C$ is *value definite* in the context C with respect to ν if $\nu(o, C)$ is defined. Otherwise, o is *value indefinite* in C . If o is value definite in all contexts $C \in \mathcal{C}$ for which $o \in C$, then o is called value definite with respect to ν . If o is value indefinite in all contexts C , then o is called *value indefinite* with respect to ν .

The set \mathcal{O} is *value definite* with respect to ν if every observable $o \in \mathcal{O}$ is value definite with respect to ν . This formal definition of value definiteness corresponds to the classical notion of determinism. Namely, ν assigns a definite value to an observable, which expresses that we are able to predict in advance the value obtained via measurement.

An observable $o \in \mathcal{O}$ is *non-contextual* with respect to ν if for all contexts $C, C' \in \mathcal{C}$ such that $o \in C$ and $o \in C'$, then $\nu(o, C) = \nu(o, C')$. Otherwise, ν is *contextual*. The set of observables \mathcal{O} is *non-contextual* with respect to ν if every observable $o \in \mathcal{O}$ which is not value indefinite (i.e. value definite in some context) is non-contextual with respect to ν . Otherwise, the set of observables \mathcal{O} is *contextual*. Non-contextuality corresponds to the classical notion that the value obtained via measurement is independent of other compatible observables measured alongside it. If an observable o is non-contextual, then it is value definite. However, this is false for sets of observables: \mathcal{O} can be non-contextual but not value definite if \mathcal{O} contains an observable that is value indefinite.

An assignment function ν is *admissible* if the following two conditions hold for $C \in \mathcal{C}$:

1. If there exists an $o \in C$ with $\nu(o, C) = 1$, then $\nu(o', C) = 0$ for all $o' \in C - \{o\}$.
2. If there exists an $o \in C$ such that $\nu(o', C) = 0$, for all $o' \in C - \{o\}$, then $\nu(o, C) = 1$

Admissibility is analogous to a *two-valued measure* used in quantum logic [2, 3, 103, 54, 94]. These definitions lead us to the ACS corollary which helps design a protocol for a quantum random bit generator that relies upon value indefiniteness.

ACS Corollary. *Let $|a\rangle, |b\rangle$ in \mathbb{C}^3 be unit vectors such that $\sqrt{\frac{5}{14}} \leq |\langle a|b\rangle| \leq \frac{3}{\sqrt{14}}$. Then there exists a set of projection observables \mathcal{O} containing P_a and P_b and a set of contexts \mathcal{C} over \mathcal{O} such that there is no admissible assignment function under which \mathcal{O} is non-contextual, P_a has the value 1 and P_b is value definite.*

The ACS experimental protocol starts with a spin-1 source and consists of two sequential measurements. Spin-1 particles are prepared in the $S_z = 0$ state. (From their eigenstate assumption, this operator has a definite value.) Specifically, the first measurement puts the particle in the $S_z = 0$ eigenstate of the spin operator S_z . Since the preparation state is an eigenstate of the $S_x = 0$ projector observable with eigenvalue 0, this outcome has a definite value and theoretically cannot be reached. The second measurement is performed in the eigenbasis of the S_x operator with two outcomes $S_x = \pm 1$.

The $S_x = \pm 1$ outcomes can be assigned 0 and 1, respectively. Moreover, since $\langle S_z = 0 | S_x = \pm 1 \rangle = \frac{1}{\sqrt{2}}$, the ACS corollary implies that neither of the $S_x = \pm 1$ outcomes can have a pre-determined definite value. This ACS

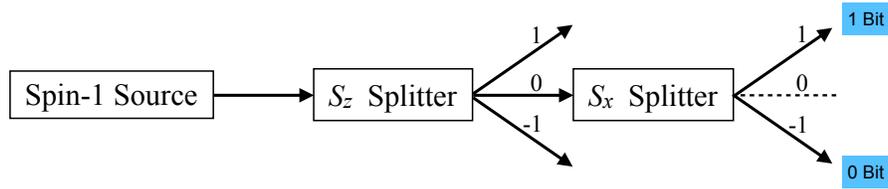


Figure 2: ACS Experimental Protocol. Quantum Observables Rendering Random Bits via Value Indefiniteness.

design provides two key properties: (1) Bits 0 and 1 are generated by value indefiniteness. (2) Bits 0 and 1 are generated independently with a 50/50 probability. Quantum random axioms 1 and 2 only require the second property. It is worth noting, however, that the first property (value indefiniteness) helps sharpen the results in section 4.

In [60], a quantum random number generator was built, empirically tested and implemented the ACS protocol that is shown in figure 2. During testing, the bias of the bits showed a 50.001% mean frequency of obtaining a 0 outcome and a standard deviation of 0.1% that is consistent with a bucket size of 999302 bits. The entropy for unbiased random numbers obtained from 10 Gigabits of raw data was 7.999999 per byte. The ideal value is, of course, 8. The data passed all standard NIST and Diehard statistical test suites [70, 31]. Furthermore, the quantum random bits were analyzed with a test [17] directly related to algorithmic randomness: the raw bits generated from [60] were used to test the primality of Carmichael numbers smaller than 54×10^7 with the Solovay-Strassen probabilistic algorithm. The metric is the minimum number of random bits needed to confirm compositeness. Ten sequences of raw quantum random bits of length 2^{29} demonstrated a significant advantage over sequences of the same length, produced from three modern pseudo-random generators – Random123, PCG and xorshiro128+.

In [15], Abbott, Calude and Svozil show that the occurrence of quantum randomness due to quantum indeterminacy is not a fluke. They show that the breakdown of non-contextual hidden variable theories occurs almost everywhere and prove that quantum indeterminacy (i.e. contextuality or value indefiniteness) is a global property. They prove that after one arbitrary observable is fixed so that it occurs with certainty, almost all (Lebesgue measure one) remaining observables are value indefinite.

4 Computing Ex-Machine Languages

A class of ex-machines are defined as evolutions of the fundamental ex-machine $\Omega(x)$, whose 15 initial instructions are listed under ex-machine 1. These ex-machines compute languages L that are subsets of $\{\mathbf{a}\}^* = \{\mathbf{a}^n : n \in \mathbb{N}\}$. The expression \mathbf{a}^n represents a string of n consecutive \mathbf{a} 's. For example, $\mathbf{a}^5 = \mathbf{aaaaa}$ and \mathbf{a}^0 is the empty string.

Define the set of languages

$$\mathcal{L} = \bigcup_{L \subset \{\mathbf{a}\}^*} \{L\}.$$

For each function $f : \mathbb{N} \rightarrow \{0, 1\}$, definition 4.1 defines a unique language in \mathcal{L} .

Definition 4.1. *Language L_f*

Consider any function $f : \mathbb{N} \rightarrow \{0, 1\}$. This means f is a member of the set $\{0, 1\}^{\mathbb{N}}$. Function f induces the language $L_f = \{\mathbf{a}^n : f(n) = 1\}$. In other words, for each non-negative integer n , string \mathbf{a}^n is in the language L_f if and only if $f(n) = 1$.

Trivially, L_f is a language in \mathfrak{L} . Moreover, these functions f generate all of \mathfrak{L} .

$$\text{Remark 4.1. } \mathfrak{L} = \bigcup_{f \in \{0,1\}^{\mathbb{N}}} \{L_f\}$$

In order to define the halting syntax for the language in \mathfrak{L} that an ex-machine computes, choose alphabet set $A = \{\#, 0, 1, \mathbb{N}, \mathbb{Y}, \mathbf{a}\}$.

Definition 4.2. *Language L in \mathfrak{L} that ex-machine \mathfrak{X} computes*

Let \mathfrak{X} be an ex-machine. The language L in \mathfrak{L} that \mathfrak{X} computes is defined as follows. A valid initial tape has the form $\# \# \mathbf{a}^n \#$. The valid initial tape $\# \# \#$ represents the empty string. After machine \mathfrak{X} starts executing with initial tape $\# \# \mathbf{a}^n \#$, string \mathbf{a}^n is in \mathfrak{X} 's language if ex-machine \mathfrak{X} halts with tape $\# \mathbf{a}^n \# \mathbb{Y} \#$. String \mathbf{a}^n is not in \mathfrak{X} 's language if \mathfrak{X} halts with tape $\# \mathbf{a}^n \# \mathbb{N} \#$.

The use of special alphabet symbols \mathbb{Y} and \mathbb{N} – to decide whether \mathbf{a}^n is in the language or not in the language – follows [62].

For a particular string $\# \# \mathbf{a}^m \#$, some ex-machine \mathfrak{X} could first halt with $\# \mathbf{a}^m \# \mathbb{N} \#$ and in a second computation with input $\# \# \mathbf{a}^m \#$ could halt with $\# \mathbf{a}^m \# \mathbb{Y} \#$. This oscillation of halting outputs could continue indefinitely and in some cases the oscillation can be aperiodic. In this case, \mathfrak{X} 's language would not be well-defined according to definition 4.2. These types of ex-machines will be avoided in this paper.

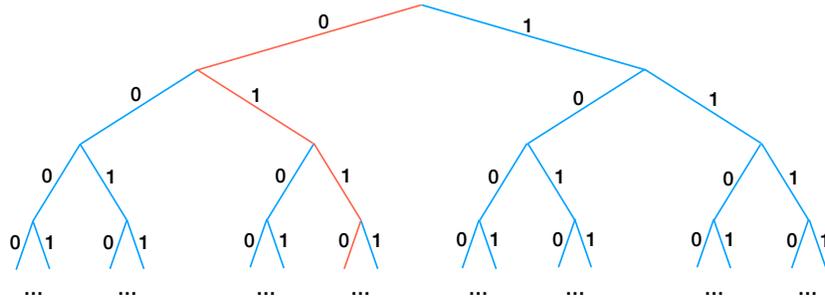
There is a subtle difference between $\mathfrak{Q}(x)$ and an ex-machine \mathfrak{X} whose halting output never stabilizes. In contrast to the Turing machine, two different instances of the ex-machine $\mathfrak{Q}(x)$ can evolve to two different machines and compute distinct languages according to definition 4.2. However, after $\mathfrak{Q}(x)$ has evolved to a new machine $\mathfrak{Q}(a_0 a_1 \dots a_m x)$ as a result of a prior execution with input tape $\# \# \mathbf{a}^m \#$, then for each i with $0 \leq i \leq m$, machine $\mathfrak{Q}(a_0 a_1 \dots a_m x)$ always halts with the same output when presented with input tape $\# \# \mathbf{a}^i \#$. In other words, $\mathfrak{Q}(a_0 a_1 \dots a_m x)$'s halting output stabilizes on all input strings \mathbf{a}^i where $0 \leq i \leq m$. Furthermore, it is the ability of $\mathfrak{Q}(x)$ to exploit the non-autonomous behavior of its two quantum random instructions that enables an evolution of $\mathfrak{Q}(x)$ to compute languages that are Turing incomputable.

We designed ex-machines that compute subsets of $\{\mathbf{a}\}^*$ rather than subsets of $\{0,1\}^*$ because the resulting specification of $\mathfrak{Q}(x)$ is much simpler and more elegant. It is straightforward to list a standard machine that bijectively translates each \mathbf{a}^n to a binary string in $\{0,1\}^*$ as follows. The empty string in $\{\mathbf{a}\}^*$ maps to the empty string in $\{0,1\}^*$. Let ψ represent this translation map. Hence, $\mathbf{a} \xrightarrow{\psi} 0$, $\mathbf{aa} \xrightarrow{\psi} 1$, $\mathbf{aaa} \xrightarrow{\psi} 00$, $\mathbf{a}^4 \xrightarrow{\psi} 01$, $\mathbf{a}^5 \xrightarrow{\psi} 10$, $\mathbf{a}^6 \xrightarrow{\psi} 11$, $\mathbf{a}^7 \xrightarrow{\psi} 000$, and so on. Similarly, an inverse translation standard machine computes the inverse of ψ . Hence $0 \xrightarrow{\psi^{-1}} \mathbf{a}$, $1 \xrightarrow{\psi^{-1}} \mathbf{aa}$, $00 \xrightarrow{\psi^{-1}} \mathbf{aaa}$, and so on. The translation and inverse translation computations immediately transfer any results about the ex-machine computation of subsets of $\{\mathbf{a}\}^*$ to corresponding subsets of $\{0,1\}^*$ via ψ . In particular, the following remark is relevant for our discussion.

Remark 4.2. Every subset of $\{\mathbf{a}\}^*$ is computable by some ex-machine if and only if every subset of $\{0,1\}^*$ is computable by some ex-machine.

Proof. The remark immediately follows from the fact that the translation map ψ and the inverse translation map ψ^{-1} are computable with a standard machine. \square

When the quantum randomness in \mathfrak{Q} 's two quantum random instructions satisfy axiom 1 (unbiased bernoulli trials) and axiom 2 (stochastic independence), for each $n \in \mathbb{N}$, all 2^n finite paths of length n in the infinite, binary tree of figure 3 are equally likely. (Feller [37, 38] covers random walks.) Moreover, there is a one-to-one correspondence between a function $f : \mathbb{N} \rightarrow \{0,1\}$ and an infinite downward path in the infinite binary tree of figure 3. The beginning of an infinite downward path is shown in red, and starts as $(0, 1, 1, 0, \dots)$.

Figure 3: Infinite Binary Tree. A Graphical Representation of $\{0, 1\}^{\mathbb{N}}$.

Based on this one-to-one correspondence between functions $f : \mathbb{N} \rightarrow \{0, 1\}$ and downward paths in the infinite binary tree, an examination of $\Omega(x)$'s execution behavior will show that $\Omega(x)$ can evolve to compute any language L_f in \mathfrak{L} when quantum random instructions $(x, \#, x, 0)$ and $(x, a, t, 0)$ satisfy axioms 1 and 2.

Ex-Machine 1. $\Omega(x)$

$A = \{\#, 0, 1, N, Y, a\}$. States $Q = \{0, h, n, y, t, v, w, x, 8\}$ where halting state $h = 1$, and states $n = 2, y = 3, t = 4, v = 5, w = 6, x = 7$. The initial state is always 0. The letters represent states instead of explicit numbers because these states have special purposes. (Letters are used solely for the reader's benefit.) State n indicates NO that the string is not in the language. State y indicates YES that the string is in the language. State x is used to generate a new random bit; this random bit determines the string corresponding to the current value of $|Q| - 1$. The fifteen instructions of $\Omega(x)$ are shown below.

```
(0, #, 8, #, 1)
(8, #, x, #, 0)
(y, #, h, Y, 0)
(n, #, h, N, 0)

(x, #, x, 0)
(x, a, t, 0)

(x, 0, v, #, 0, (|Q|-1, #, n, #, 1) )
(x, 1, w, #, 0, (|Q|-1, #, y, #, 1) )

(t, 0, w, a, 0, (|Q|-1, #, n, #, 1) )
(t, 1, w, a, 0, (|Q|-1, #, y, #, 1) )

(v, #, n, #, 1, (|Q|-1, a, |Q|, a, 1) )

(w, #, y, #, 1, (|Q|-1, a, |Q|, a, 1) )
(w, a, |Q|, a, 1, (|Q|-1, a, |Q|, a, 1) )

(|Q|-1, a, x, a, 0)
(|Q|-1, #, x, #, 0)
```

With initial state 0 and initial tape $\# \#aaaa\#\#$, an execution of machine $\Omega(x)$ is shown below.

STATE	TAPE	HEAD	INSTRUCTION	NEW INSTRUCTION
8	## aaaa###	1	(0, #, 8, #, 1)	
x	## aaaa###	1	(8, a, x, a, 0)	
t	## 1aaa###	1	(x, a, t, 1_qr, 0)	
w	## aaaa###	1	(t, 1, w, a, 0, (Q -1, #, y, #, 1))	(8, #, y, #, 1)
9	##a aaa###	2	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(8, a, 9, a, 1)
x	##a aaa###	2	(9, a, x, a, 0)	(9, a, x, a, 0)
t	##a 1aa###	2	(x, a, t, 1_qr, 0)	
w	##a aaa###	2	(t, 1, w, a, 0, (Q -1, #, y, #, 1))	(9, #, y, #, 1)
10	##aa aa###	3	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(9, a, 10, a, 1)
x	##aa aa###	3	(10, a, x, a, 0)	(10, a, x, a, 0)
t	##aa 0a###	3	(x, a, t, 0_qr, 0)	
w	##aa aa###	3	(t, 0, w, a, 0, (Q -1, #, n, #, 1))	(10, #, n, #, 1)
11	##aaa a###	4	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(10, a, 11, a, 1)
x	##aaa a###	4	(11, a, x, a, 0)	(11, a, x, a, 0)
t	##aaa 1###	4	(x, a, t, 1_qr, 0)	
w	##aaa a###	4	(t, 1, w, a, 0, (Q -1, #, y, #, 1))	(11, #, y, #, 1)
12	##aaaa ###	5	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(11, a, 12, a, 1)
x	##aaaa ###	5	(12, #, x, #, 0)	(12, #, x, #, 0)
x	##aaaa 0##	5	(x, #, x, 0_qr, 0)	
v	##aaaa ###	5	(x, 0, v, #, 0, (Q -1, #, n, #, 1))	(12, #, n, #, 1)
n	##aaaa# ##	6	(v, #, n, #, 1, (Q -1, a, Q , a, 1))	(12, a, 13, a, 1)
h	##aaaa# N#	6	(n, #, h, N, 0)	

During this execution, $\Omega(x)$ replaces instruction (8, #, x, #, 0) with (8, #, y, #, 1). Meta instruction (w, a, |Q|, a, 1, (|Q|-1, a, |Q|, a, 1)) executes and replaces (8, a, x, a, 0) with new instruction (8, a, 9, a, 1). Also, simple meta instruction (|Q|-1, a, x, a, 0) temporarily added instructions (9, a, x, a, 0), (10, a, x, a, 0), and (11, a, x, a, 0).

Subsequently, these new instructions were replaced by (9, a, 10, a, 1), (10, a, 11, a, 1), and (11, a, 12, a, 1), respectively. Similarly, simple meta instruction (|Q|-1, #, x, #, 0) added instruction (12, #, x, #, 0) and this instruction was replaced by instruction (12, #, n, #, 1). Lastly, instructions (9, #, y, #, 1), (10, #, n, #, 1), (11, #, y, #, 1), and (12, a, 13, a, 1) were added.

Furthermore, five new states 9, 10, 11, 12 and 13 were added to Q . After this computation halts, the machine states are $Q = \{0, h, n, y, t, v, w, x, 8, 9, 10, 11, 12, 13\}$ and the resulting ex-machine evolved to has 24 instructions. It is called $\Omega(11010 x)$.

Ex-Machine 2. $\Omega(11010 x)$

(0, #, 8, #, 1)

(y, #, h, Y, 0)
(n, #, h, N, 0)

(x, #, x, 0)
(x, a, t, 0)

(x, 0, v, #, 0, (|Q|-1, #, n, #, 1))
(x, 1, w, #, 0, (|Q|-1, #, y, #, 1))

(t, 0, w, a, 0, (|Q|-1, #, n, #, 1))
(t, 1, w, a, 0, (|Q|-1, #, y, #, 1))

(v, #, n, #, 1, (|Q|-1, a, |Q|, a, 1))

(w, #, y, #, 1, (|Q|-1, a, |Q|, a, 1))
(w, a, |Q|, a, 1, (|Q|-1, a, |Q|, a, 1))

(|Q|-1, a, x, a, 0)
(|Q|-1, #, x, #, 0)

(8, #, y, #, 1)
 (8, a, 9, a, 1)

(9, #, y, #, 1)
 (9, a, 10, a, 1)

(10, #, n, #, 1)
 (10, a, 11, a, 1)

(11, #, y, #, 1)
 (11, a, 12, a, 1)

(12, #, n, #, 1)
 (12, a, 13, a, 1)

New instructions (8, #, y, #, 1), (9, #, y, #, 1), and (11, #, y, #, 1) help $\Omega(11010 x)$ compute that the empty string, **a** and **aaa** are in its language, respectively. Similarly, the new instructions (10, #, n, #, 1) and (12, #, n, #, 1) help $\Omega(11010 x)$ compute that **aa** and **aaaa** are not in its language, respectively.

The zeroeth, first, and third 1 in $\Omega(11010 x)$'s name indicate that the empty string, **a** and **aaa** are in $\Omega(11010 x)$'s language. The second and fourth 0 indicate strings **aa** and **aaaa** are not in its language. The symbol x indicates that all strings a^n with $n \geq 5$ have not yet been determined whether they are in $\Omega(11010 x)$'s language or not in its language.

Starting at state 0, ex-machine $\Omega(11010 x)$ computes that the empty string is in $\Omega(11010 x)$'s language.

STATE	TAPE	TAPE HEAD	INSTRUCTION
8	## ###	1	(0, #, 8, #, 1)
y	### ##	2	(8, #, y, #, 1)
h	### Y#	2	(y, #, h, Y, 0)

Starting at state 0, ex-machine $\Omega(11010 x)$ computes that string **a** is in $\Omega(11010 x)$'s language.

STATE	TAPE	TAPE HEAD	INSTRUCTION
8	## a###	1	(0, #, 8, #, 1)
9	##a ###	2	(8, a, 9, a, 1)
y	##a# ##	3	(9, #, y, #, 1)
h	##a# Y#	3	(y, #, h, Y, 0)

Starting at state 0, $\Omega(11010 x)$ computes that string **aa** is not in $\Omega(11010 x)$'s language.

STATE	TAPE	TAPE HEAD	INSTRUCTION
8	## aa###	1	(0, #, 8, #, 1)
9	##a a###	2	(8, a, 9, a, 1)
10	##aa ###	3	(9, a, 10, a, 1)
n	##aa# ##	4	(10, #, n, #, 1)
h	##aa# N#	4	(n, #, h, N, 0)

Starting at state 0, $\Omega(11010 x)$ computes that **aaa** is in $\Omega(11010 x)$'s language.

STATE	TAPE	TAPE HEAD	INSTRUCTION
8	## aaa###	1	(0, #, 8, #, 1)
9	##a aa###	2	(8, a, 9, a, 1)
10	##aa a###	3	(9, a, 10, a, 1)
11	##aaa ###	4	(10, a, 11, a, 1)
y	##aaa# ##	5	(11, #, y, #, 1)
h	##aaa# Y#	5	(y, #, h, Y, 0)

Starting at state 0, $\Omega(11010 x)$ computes that **aaaa** is not in $\Omega(11010 x)$'s language.

STATE	TAPE	TAPE HEAD	INSTRUCTION
8	## aaaa####	1	(0, #, 8, #, 1)
9	##a aaa####	2	(8, a, 9, a, 1)
10	##aa aa####	3	(9, a, 10, a, 1)
11	##aaa a####	4	(10, a, 11, a, 1)
12	##aaaa ####	5	(11, a, 12, a, 1)
n	#####	6	(12, #, n, #, 1)
h	##### N##	6	(n, #, h, N, 0)

Note that for each of these executions, no new states were added and no instructions were added or replaced. Thus, for all subsequent executions, ex-machine $\Omega(11010 x)$ computes that the empty string, **a** and **aaa** are in its language. Similarly, strings **aa** and **aaaa** are not in $\Omega(11010 x)$'s language for all subsequent executions of $\Omega(11010 x)$.

Starting at state 0, we examine an execution of ex-machine $\Omega(11010 x)$ on input tape # **#####**.

STATE	TAPE	HEAD	INSTRUCTION	NEW INSTRUCTION
8	#####	1	(0, #, 8, #, 1)	
9	##a #####	2	(8, a, 9, a, 1)	
10	##aa #####	3	(9, a, 10, a, 1)	
11	##aaa #####	4	(10, a, 11, a, 1)	
12	##### aaa###	5	(11, a, 12, a, 1)	
13	##### aa###	6	(12, a, 13, a, 1)	
x	##### aa###	6	(13, a, x, a, 0)	
t	##### 0a###	6	(x, a, t, 0_qr, 0)	
w	##### aa###	6	(t, 0, w, a, 0, (Q -1, #, n, #, 1))	(13, #, n, #, 1)
14	##### a###	7	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(13, a, 14, a, 1)
x	##### a###	7	(14, a, x, a, 0)	(14, a, x, a, 0)
t	##### 1###	7	(x, a, t, 1_qr, 0)	
w	##### a###	7	(t, 1, w, a, 0, (Q -1, #, y, #, 1))	(14, #, y, #, 1)
15	##### ###	8	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(14, a, 15, a, 1)
x	##### ###	8	(15, #, x, #, 0)	(15, #, x, #, 0)
x	##### 1##	8	(x, #, x, 1_qr, 0)	
w	##### ###	8	(x, 1, w, #, 0, (Q -1, #, y, #, 1))	(15, #, y, #, 1)
y	##### ##	9	(w, #, y, #, 1, (Q -1, a, Q , a, 1))	(15, a, 16, a, 1)
h	##### Y#	9	(y, #, h, Y, 0)	

Overall, during this execution ex-machine $\Omega(11010 x)$ evolved to ex-machine $\Omega(11010 011 x)$. Three quantum random instructions were executed. The first quantum random instruction (x, a, t, 0) measured a 0 so it is shown above as (x, a, t, 0_qr, 0). The result of this 0 bit measurement adds the instruction (13, #, n, #, 1), so that in all subsequent executions of ex-machine $\Omega(11010 011 x)$, string **a⁵** is not in $\Omega(11010 011 x)$'s language. Similarly, the second quantum random instruction (x, a, t, 0) measured a 1 so it is shown above as (x, a, t, 1_qr, 0). The result of this 1 bit measurement adds the instruction (14, #, y, #, 1), so that in all subsequent executions, string **a⁶** is in $\Omega(11010 011 x)$'s language. Finally, the third quantum random instruction (x, #, x, 0) measured a 1 so it is shown above as (x, #, x, 1_qr, 0). The result of this 1 bit measurement adds the instruction (15, #, y, #, 1), so that in all subsequent executions, string **a⁷** is in $\Omega(11010 011 x)$'s language.

Lastly, starting at state 0, we examine a distinct execution of ex-machine $\Omega(11010 x)$ on input tape # **#####**. A distinct execution of $\Omega(11010 x)$ evolves to ex-machine $\Omega(11010 000 x)$.

STATE	TAPE	HEAD	INSTRUCTION	NEW INSTRUCTION
8	## aaaaaa###	1	(0, #, 8, #, 1)	
9	##a aaaaaa###	2	(8, a, 9, a, 1)	
10	##aa aaaaa###	3	(9, a, 10, a, 1)	
11	###aa aaaa###	4	(10, a, 11, a, 1)	
12	##### aaa###	5	(11, a, 12, a, 1)	
13	##### aa###	6	(12, a, 13, a, 1)	
x	##### aa###	6	(13, a, x, a, 0)	
t	##### 0a###	6	(x, a, t, 0_qr, 0)	
w	##### aa###	6	(t, 0, w, a, 0, (Q -1, #, n, #, 1))	(13, #, n, #, 1)
14	##### a###	7	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(13, a, 14, a, 1)
x	##### a###	7	(14, a, x, a, 0)	(14, a, x, a, 0)
t	##### 0###	7	(x, a, t, 0_qr, 0)	
w	##### a###	7	(t, 0, w, a, 0, (Q -1, #, n, #, 1))	(14, #, n, #, 1)
15	##### ###	8	(w, a, Q , a, 1, (Q -1, a, Q , a, 1))	(14, a, 15, a, 1)
x	##### ###	8	(15, #, x, #, 0)	(15, #, x, #, 0)
x	##### 0##	8	(x, #, x, 0_qr, 0)	
v	##### ###	8	(x, 0, v, #, 0, (Q -1, #, n, #, 1))	(15, #, n, #, 1)
n	##### ##	9	(v, #, n, #, 1, (Q -1, a, Q , a, 1))	(15, a, 16, a, 1)
h	##### N#	9	(n, #, h, N, 0)	

Based on our previous examination of ex-machine $\Omega(x)$ evolving to $\Omega(11010x)$ and then subsequently $\Omega(11010x)$ evolving to $\Omega(11010011x)$, ex-machine 3 specifies $\Omega(a_0a_1\dots a_mx)$ in terms of initial states and initial instructions.

Ex-Machine 3. $\Omega(a_0a_1\dots a_mx)$

Let $m \in \mathbb{N}$. Set $Q = \{0, h, n, y, t, v, w, x, 8, 9, 10, \dots, m+8, m+9\}$. For $0 \leq i \leq m$, each a_i is 0 or 1. ex-machine $\Omega(a_0a_1\dots a_mx)$'s instructions are shown below. Symbol $b_8 = y$ if $a_0 = 1$. Otherwise, symbol $b_8 = n$ if $a_0 = 0$. Similarly, symbol $b_9 = y$ if $a_1 = 1$. Otherwise, symbol $b_9 = n$ if $a_1 = 0$. And so on until reaching the second to the last instruction $(m+8, \#, b_{m+8}, \#, 1)$, symbol $b_{m+8} = y$ if $a_m = 1$. Otherwise, symbol $b_{m+8} = n$ if $a_m = 0$.

(0, #, 8, #, 1)

(y, #, h, Y, 0)
(n, #, h, N, 0)

(x, #, x, 0)
(x, a, t, 0)

(x, 0, v, #, 0, (|Q|-1, #, n, #, 1))
(x, 1, w, #, 0, (|Q|-1, #, y, #, 1))

(t, 0, w, a, 0, (|Q|-1, #, n, #, 1))
(t, 1, w, a, 0, (|Q|-1, #, y, #, 1))

(v, #, n, #, 1, (|Q|-1, a, |Q|, a, 1))

(w, #, y, #, 1, (|Q|-1, a, |Q|, a, 1))
(w, a, |Q|, a, 1, (|Q|-1, a, |Q|, a, 1))

(|Q|-1, a, x, a, 0)
(|Q|-1, #, x, #, 0)

(8, #, b_8 , #, 1)
(8, a, 9, a, 1)

$(9, \#, b_9, \#, 1)$
 $(9, a, 10, a, 1)$

 $(10, \#, b_{10}, \#, 1)$
 $(10, a, 11, a, 1)$

 \dots
 $(i+8, \#, b_{i+8}, \#, 1)$
 $(i+8, a, i+9, a, 1)$

 \dots
 $(m+7, \#, b_{m+7}, \#, 1)$
 $(m+7, a, m+8, a, 1)$

 $(m+8, \#, b_{m+8}, \#, 1)$
 $(m+8, a, m+9, a, 1)$

Lemma 4.1. *Whenever i satisfies $0 \leq i \leq m$, string \mathbf{a}^i is in $\mathfrak{Q}(a_0 a_1 \dots a_m x)$'s language if $a_i = 1$; string \mathbf{a}^i is not in $\mathfrak{Q}(a_0 a_1 \dots a_m x)$'s language if $a_i = 0$. Whenever $n > m$, it has not yet been determined whether string \mathbf{a}^n is in $\mathfrak{Q}(a_0 a_1 \dots a_m x)$'s language or not in its language.*

Proof. When $0 \leq i \leq m$, the first consequence follows immediately from the definition of \mathbf{a}^i being in $\mathfrak{Q}(a_0 a_1 \dots a_m x)$'s language and from ex-machine 3. In instruction $(i+8, \#, b_{i+8}, \#, 1)$ the state value of b_{i+8} is y if $a_i = 1$ and b_{i+8} is n if $a_i = 0$.

For the indeterminacy of strings \mathbf{a}^n when $n > m$, ex-machine $\mathfrak{Q}(a_0 \dots a_m x)$ executes its last instruction $(m+8, a, m+9, a, 1)$ when it is scanning the m th a in \mathbf{a}^n . Subsequently, for each a on the tape to the right of $\#a^m$, ex-machine $\mathfrak{Q}(a_0 \dots a_m x)$ executes the quantum random instruction $(x, a, t, 0)$.

If the execution of $(x, a, t, 0)$ measures a 0, the two meta instructions $(t, 0, w, a, 0, (|Q|-1, \#, n, \#, 1))$ and $(w, a, |Q|, a, 1, (|Q|-1, a, |Q|, a, 1))$ are executed. If the next alphabet symbol to the right is an a , then a new standard instruction is executed that is instantiated from the simple meta instruction $(|Q|-1, a, x, a, 0)$. If the tape head was scanning the last a in \mathbf{a}^n , then a new standard instruction is executed that is instantiated from the simple meta instruction $(|Q|-1, \#, x, \#, 0)$.

If the execution of $(x, a, t, 0)$ measures a 1, the two meta instructions $(t, 1, w, a, 0, (|Q|-1, \#, y, \#, 1))$ and $(w, a, |Q|, a, 1, (|Q|-1, a, |Q|, a, 1))$ are executed. If the next alphabet symbol to the right is an a , then a new standard instruction is executed that is instantiated from the simple meta instruction $(|Q|-1, a, x, a, 0)$. If the tape head was scanning the last a in \mathbf{a}^n , then a new standard instruction is executed that is instantiated from the simple meta instruction $(|Q|-1, \#, x, \#, 0)$.

In this way, for each a on the tape to the right of $\#a^m$, the execution of the quantum random instruction $(x, a, t, 0)$ determines whether each string \mathbf{a}^{m+k} , satisfying $1 \leq k \leq n - m$, is in or not in $\mathfrak{Q}(a_0 a_1 \dots a_n x)$'s language.

After the execution of $(|Q|-1, \#, x, \#, 0)$, the tape head is scanning a blank symbol, so the quantum random instruction $(x, \#, x, 0)$ is executed. If a 0 is measured by the quantum random source, the meta instructions $(x, 0, v, \#, 0, (|Q|-1, \#, n, \#, 1))$ and $(v, \#, n, \#, 1, (|Q|-1, a, |Q|, a, 1))$ are executed. Then the last instruction executed is $(n, \#, h, N, 0)$ which indicates that \mathbf{a}^n is not in $\mathfrak{Q}(a_0 a_1 \dots a_n x)$'s language.

If the execution of $(x, \#, x, 0)$ measures a 1, the meta instructions $(x, 1, w, \#, 0, (|Q|-1, \#, y, \#, 1))$ and $(w, \#, y, \#, 1, (|Q|-1, a, |Q|, a, 1))$ are executed. Then the last instruction executed is $(y, \#, h, Y, 0)$ which indicates that \mathbf{a}^n is in $\mathfrak{Q}(a_0 a_1 \dots a_n x)$'s language.

During the execution of the instructions, for each a on the tape to the right of $\#a^m$, $\mathfrak{Q}(a_0 a_1 \dots a_m x)$ evolves to $\mathfrak{Q}(a_0 a_1 \dots a_n x)$ according to the specification in ex-machine 3, where one substitutes n for m . \square

Remark 4.3. When the binary string $a_0a_1 \dots a_m$ is presented as input, the ex-machine instructions for $\Omega(a_0a_1 \dots a_m x)$, specified in ex-machine 3, are constructible (i.e., can be printed) with a standard machine.

In contrast with lemma 4.1, $\Omega(a_0a_1 \dots a_m x)$'s instructions are not executable with a standard machine when the input tape $\# \#a^i\#$ satisfies $i > m$ because meta and quantum random instructions are required. Thus, remark 4.3 distinguishes the construction of $\Omega(a_0a_1 \dots a_m x)$'s instructions from the execution of $\Omega(a_0a_1 \dots a_m x)$'s instructions.

Proof. When given a finite list $(a_0 \ a_1 \ \dots \ a_m)$, where each a_i is 0 or 1, the code listing below constructs $\Omega(a_0a_1 \dots a_m x)$'s instructions. Starting with comment `;; Qx_builder.lsp`, the code listing is expressed in a dialect [69] of LISP. The LISP language [65, 66, 67] originated from the lambda calculus [22, 23, 55]. The appendix of [96] outlines a proof that the lambda calculus is computationally equivalent to a standard machine. The following 3 instructions print the ex-machine instructions for $\Omega(11010 x)$, listed in ex-machine 2.

```
(set 'a0_a1_dots_am (list 1 1 0 1 0) )
(set 'Qx_machine (build_Qx_machine a0_a1_dots_am) )
(print_xmachine Qx_machine)
```

□

```
;; Qx_builder.lsp
(define (make_qr_instruction q_in a_in q_out move)
  (list (string q_in) (string a_in) (string q_out) (string move)) )

(define (make_instruction q_in a_in q_out a_out move)
  (list (string q_in) (string a_in)
        (string q_out) (string a_out) (string move)) )

(define (make_meta_instruction q_in a_in q_out a_out move_standard r_in b_in r_out b_out move_meta)
  (list (string q_in) (string a_in)
        (string q_out) (string a_out) (string move_standard)
        (make_instruction r_in b_in r_out b_out move_meta) ) )

(define (initial_Qx_machine)
  (list
    (make_instruction "0" "#" "8" "#" 1)
    (make_instruction "8" "#" "x" "#" 0)

    (make_instruction "y" "#" "h" "Y" 0) ;; This means string a^i is in the language.
    (make_instruction "n" "#" "h" "N" 0) ;; This means string a^i is NOT in the language.

    (make_qr_instruction "x" "#" "x" 0)
    (make_qr_instruction "x" "a" "t" 0)

    (make_meta_instruction "x" "0" "v" "#" 0 "|Q|-1" "#" "n" "#" 1)
    (make_meta_instruction "x" "1" "w" "#" 0 "|Q|-1" "#" "y" "#" 1)

    (make_meta_instruction "t" "0" "w" "a" 0 "|Q|-1" "#" "n" "#" 1)
    (make_meta_instruction "t" "1" "w" "a" 0 "|Q|-1" "#" "y" "#" 1)

    (make_meta_instruction "v" "#" "n" "#" 1 "|Q|-1" "a" "|Q|" "a" 1)
    (make_meta_instruction "w" "#" "y" "#" 1 "|Q|-1" "a" "|Q|" "a" 1)
    (make_meta_instruction "w" "a" "|Q|" "a" 1 "|Q|-1" "a" "|Q|" "a" 1)

    (make_instruction "|Q|-1" "a" "x" "a" 0)
    (make_instruction "|Q|-1" "#" "x" "#" 0)
  ))
```

```

(define (add_instruction instruction q_list)
  (append q_list (list instruction) ) )

(define (check_a0_a1_dots_am a0_a1_dots_am)
  (if (list? a0_a1_dots_am)
      (dolist (a_i a0_a1_dots_am)
        (if (member a_i (list 0 1) )
            true
            (begin
              (println "ERROR! (build_Qx_machine a0_a1_dots_am). a_i = " a_i)
              (exit)
            )
        )
      )
      a0_a1_dots_am )
  nil
))

;;;;;;;;;;;;;;;;;;;;;;;;; BUILD MACHINE Q(a0 a1 . . . am x)
;; a0_a1_dots_am has to be a list of 0's and 1's or '()
(define (build_Qx_machine a0_a1_dots_am)
  (let
    ( (Qx_machine (initial_Qx_machine))
      (|Q| 8)
      (b_|Q| nil)
      (ins1 nil)
      (ins2 nil)
    )
    (set 'check (check_a0_a1_dots_am a0_a1_dots_am) )

    ;; if nil OR check is an empty list, remove instruction (8, #, x, #, 0)
    (if (or (not check) (empty? check) )
        true
        (set 'Qx_machine (append (list (Qx_machine 0)) (rest (rest Qx_machine))))
    )

    (if (list? a0_a1_dots_am)
        (dolist (a_i a0_a1_dots_am)
          (if (= a_i 1)
              (set 'b_|Q| "y")
              (set 'b_|Q| "n") )

          (set 'ins1 (make_instruction |Q| "#" b_|Q| "#" 1) )
          (set 'Qx_machine (add_instruction ins1 Qx_machine) )

          (set 'ins2 (make_instruction |Q| "a" (+ |Q| 1) "a" 1))
          (set 'Qx_machine (add_instruction ins2 Qx_machine) )
          (set '|Q| (+ |Q| 1) )
        )
    )

    Qx_machine
  ))

```

```

(define (print_elements instruction)
  (let
    ( (idx_ub (min 4 (- (length instruction) 1)) )
      (i 0)
    )
    (for (i 0 idx_ub)
      (print (instruction i) )
      (if (< i idx_ub) (print ", ")
        )
    )
  ))

(define (print_instruction instruction)
  (print "(")
  (if (<= (length instruction) 5)
    (print_elements instruction)
    (begin
      (print_elements instruction)
      (print ", (")
      (print_elements (instruction 5) )
      (print ") " ) )
    )
  (println ")")
)

(define (print_xmachine x_machine)
  (println)
  (dolist (instruction x_machine)
    (print_instruction instruction))
  (println)
)

```

Definition 4.3. Define \mathfrak{U} as the union of $\mathfrak{Q}(x)$ and all ex-machines $\mathfrak{Q}(a_0 \dots a_m x)$ for each $m \in \mathbb{N}$ and for each $a_0 \dots a_m$ in $\{0, 1\}^{m+1}$. In other words,

$$\mathfrak{U} = \{\mathfrak{Q}(x)\} \cup \bigcup_{m=0}^{\infty} \bigcup_{a_0 \dots a_m \in \{0,1\}^{m+1}} \{\mathfrak{Q}(a_0 a_1 \dots a_m x)\}.$$

Theorem 4.2. Each language L_f in \mathfrak{L} can be computed by the evolving sequence of ex-machines $\mathfrak{Q}(x)$, $\mathfrak{Q}(f(0)x)$, $\mathfrak{Q}(f(0)f(1)x)$, \dots , $\mathfrak{Q}(f(0)f(1) \dots f(n)x)$, \dots .

Proof. The theorem follows from ex-machine 1, ex-machine 3 and lemma 4.1. \square

Corollary 4.3. Given function $f : \mathbb{N} \rightarrow \{0, 1\}$, for any arbitrarily large n , the evolving sequence of ex-machines $\mathfrak{Q}(f(0)f(1) \dots f(n)x)$, $\mathfrak{Q}(f(0)f(1) \dots f(n)f(n+1)x)$, \dots computes language L_f .

Corollary 4.4. Moreover, for each n , all ex-machines $\mathfrak{Q}(x)$, $\mathfrak{Q}(f(0)x)$, $\mathfrak{Q}(f(0)f(1)x)$, \dots , $\mathfrak{Q}(f(0)f(1) \dots f(n)x)$ combined have used only a finite amount of tape, finite number of states, finite number of instructions, finite number of executions of instructions and only a finite amount of quantum random information measured by the quantum random instructions.

Proof. For each n , the finite use of computational resources follows immediately from remark 2, definition 7 and the specification of ex-machine 3. \square

A set X is called countable if there exists a bijection between X and \mathbb{N} . Since the set of all Turing machines is countable and each Turing machine only recognizes a single language *most (in the sense of Cantor's heirarchy of infinities [19]) languages L_f are not computable with a Turing machine.* More precisely, the cardinality of the set of languages L_f computable with a Turing machine is \aleph_0 , while the cardinality of the set of all languages \mathcal{L} is \aleph_1 .

For each non-negative integer n , define the language tree $\mathcal{L}(a_0a_1 \dots a_n) = \{L_f : f \in \{0,1\}^{\mathbb{N}} \text{ and } f(i) = a_i \text{ for } i \text{ satisfying } 0 \leq i \leq n\}$. Define the corresponding subset of $\{0,1\}^{\mathbb{N}}$ as $\mathcal{S}(a_0a_1 \dots a_n) = \{f \in \{0,1\}^{\mathbb{N}} : f(i) = a_i \text{ for } i \text{ satisfying } 0 \leq i \leq n\}$. Let Ψ denote this 1-to-1 correspondence, where $\mathcal{L} \xleftrightarrow{\Psi} \{0,1\}^{\mathbb{N}}$ and $\mathcal{L}(a_0a_1 \dots a_n) \xleftrightarrow{\Psi} \mathcal{S}(a_0a_1 \dots a_n)$.

Since the two quantum random axioms 1 and 2 are satisfied, each finite path $f(0)f(1) \dots f(n)$ is equally likely and there are 2^{n+1} of these paths. Thus, each path of length $n+1$ has probability $2^{-(n+1)}$. These uniform probabilities on finite strings of the same length can be extended to the Lebesgue measure μ on probability space $\{0,1\}^{\mathbb{N}}$ [37, 38]. Hence, each subset $\mathcal{S}(a_0a_1 \dots a_n)$ has measure $2^{-(n+1)}$. That is, $\mu(\mathcal{S}(a_0a_1 \dots a_n)) = 2^{-(n+1)}$ and $\mu(\{0,1\}^{\mathbb{N}}) = 1$. Via the Ψ correspondence between each language tree $\mathcal{L}(a_0a_1 \dots a_n)$ and subset $\mathcal{S}(a_0a_1 \dots a_n)$, uniform probability measure μ induces a uniform probability measure ν on \mathcal{L} , where $\nu(\mathcal{L}(a_0a_1 \dots a_n)) = 2^{-(n+1)}$ and $\nu(\mathcal{L}) = 1$.

Theorem 4.5. *For functions $f : \mathbb{N} \rightarrow \{0,1\}$, the probability that language L_f is Turing incomputable has measure 1 in (ν, \mathcal{L}) .*

Proof. The Turing machines are countable and therefore the number of functions $f : \mathbb{N} \rightarrow \{0,1\}$ that are Turing computable is countable. Hence, via the Ψ correspondence, the Turing computable languages L_f have measure 0 in \mathcal{L} . \square

Moreover, the Martin-Löf random sequences $f : \mathbb{N} \rightarrow \{0,1\}$ have Lebesgue measure 1 in $\{0,1\}^{\mathbb{N}}$ and are a proper subset of the Turing incomputable sequences. See [12, 64].

Corollary 4.6. *$\Omega(x)$ is not a Turing machine. Each ex-machine $\Omega(a_0a_1 \dots a_m x)$ in \mathfrak{U} is not a Turing machine.*

Proof. $\Omega(x)$ can evolve to compute Turing incomputable languages on a set of probability measure 1 with respect to (ν, \mathcal{L}) . Also, $\Omega(a_0a_1 \dots a_m x)$ can evolve to compute Turing incomputable languages on a set of measure $2^{-(m+1)}$ with respect to (ν, \mathcal{L}) . In contrast, each Turing machine only recognizes a single language, which has measure 0. In fact, the measure of all Turing computable languages is 0 in \mathcal{L} . \square

Remark 4.4. The statements in theorem 4.5 and corollary 4.6 can be sharpened when deeper results are obtained for the quantum random source [14, 15, 16, 60] used by the quantum random instructions.

5 Some $\Omega(x)$ Observations Based on Cantor and Gödel

At first glance, the results from the prior section may seem paradoxical. Even though there are only a countable number of initial ex-machines in \mathfrak{U} , the ex-machines evolving from $\Omega(x)$ can compute languages L_f where each $f : \mathbb{N} \rightarrow \{0,1\}$ corresponds to a particular instance selected from an uncountable number of infinite paths in the infinite binary tree (i.e., $\{0,1\}^{\mathbb{N}}$ is uncountable [18]). With initial state 0 and initial tape $\# \# \mathbf{a}^n \#$, for every n and m with $n > m$, each ex-machine $\Omega(a_0a_1 \dots a_m x)$ has an uncountably infinite number of possible execution behaviors. On the other hand, a Turing machine with the same initial state 0 and initial tape $\# \# \mathbf{a}^n \#$ always has exactly one execution behavior. Hence, a Turing machine can only have a countable number of execution behaviors for all initial tapes $\# \# \mathbf{a}^n \#$, where $n > m$.

It may seem peculiar that the countable set \mathcal{U} of ex-machines can evolve to compute an uncountable number of languages L_f . However, there is an analogous phenomenon in elementary analysis that mathematicians routinely accept. The rational numbers \mathbb{Q} are countable. The set $\mathbb{Q} \cap [0, 1]$ is dense in the closed interval $[0, 1]$ of real numbers. Any real number $r \in [0, 1]$ can be expressed as $r = \sum_{i=1}^{\infty} b_i 2^{-i}$, where each $b_i \in \{0, 1\}$. Set the m th rational number $q_m = \sum_{i=1}^m b_i 2^{-i}$. Then $\lim_{m \rightarrow \infty} q_m = r$. Thus, each real number can be realized as a sequence of rational numbers, even though the real numbers are uncountable. Furthermore, each rational number in that sequence is representable with a finite amount of information (bits). Similar to the sequence of rationals q_m converging to a real number, each language L_f can be computed (i.e., realized) by the evolving sequence of ex-machines $\mathcal{Q}(x)$, $\mathcal{Q}(f(0)x)$, $\mathcal{Q}(f(0)f(1)x)$, \dots , $\mathcal{Q}(f(0)f(1)\dots f(n)x)$, \dots , where for each n , all ex-machines $\mathcal{Q}(x)$, $\mathcal{Q}(f(0)x)$, $\mathcal{Q}(f(0)f(1)x)$, \dots , $\mathcal{Q}(f(0)f(1)\dots f(n)x)$ have used only a finite amount of tape, finite number of states, finite number of instructions, finite number of executions of instructions and a finite amount of quantum random information has been measured.

Finally, our attention turns to an insightful remark by Gödel, entitled *A philosophical error in Turing's work* [49]. Gödel states:

Turing in his [1936 [96], section 9] gives an argument which is supposed to show that mental procedures cannot go beyond mechanical procedures. However, this argument is inconclusive. What Turing disregards completely is the fact that *mind, its use, is not static, but constantly developing*, i.e., that we understand abstract terms more and more precisely as we go on using them, and that more and more abstract terms enter the sphere of our understanding. There may exist systematic methods of actualizing this development, which could form part of the procedure. Therefore, although at each stage the number and precision of the abstract terms at our disposal may be *finite*, both (and, therefore, also Turing's number of *distinguishable states of mind*) may *converge toward infinity* in the course of the application of the procedure. Note that something like this indeed seems to happen in the process of forming stronger and stronger axioms of infinity in set theory. This process, however, today is far from being sufficiently understood to form a well-defined procedure which could actually be carried out (and would yield a non-recursive number-theoretic function).

Although we make no claim whatsoever that the execution of $\mathcal{Q}(x)$ functions anything like a mental procedure, Gödel attributes some properties to mental procedures that are strikingly similar to the ex-machine. First, the ex-machine $\mathcal{Q}(a_0 a_1 \dots a_m x)$ is not static and constantly develops each time it is queried with a string \mathbf{a}^n such that $n > m$. (String \mathbf{a}^n is longer than any prior string that an ancestor of $\mathcal{Q}(a_0 a_1 \dots a_m x)$ has executed upon.)

After $\mathcal{Q}(a_0 a_1 \dots a_n x)$'s computation halts, the resulting ex-machine always has a finite number of states and a finite number of instructions, so at each stage of the evolution, the ex-machine is finite. Lastly, consider Gödel's comment: "(and, therefore, also Turing's number of *distinguishable states of mind*) may *converge toward infinity* in the course of the application of the procedure". Gödel's insight seems to foresee the ex-machine's ability to add new states; moreover, to compute a Turing incomputable language L_f , any ex-machine $\mathcal{Q}(f(0)f(1)\dots f(n)x)$ must have an evolutionary path that has an unbounded number of states.

6 An Ex-Machine Halting Problem

In [96], Alan Turing posed the halting problem for Turing machines. Does there exist a Turing machine \mathcal{D} that can determine for any given Turing machine \mathcal{M} and finitely bounded initial tape T whether \mathcal{M} 's execution on

tape T eventually halts? In the same paper [96], Turing proved that no single Turing machine could solve his halting problem.

Next, we explain what Turing's seminal result relies upon in terms of abstract computational resources. Turing's result means that there does not exist a single Turing machine \mathcal{H} – regardless of the size of \mathcal{H} 's finite state set Q and finite alphabet set A – so that when this special machine \mathcal{H} is presented with any Turing machine \mathcal{M} with a finitely bounded initial tape T and initial state q_0 , then \mathcal{H} can execute a finite number of computational steps, halt and correctly determine whether \mathcal{M} halts or does not halt with a tape T and initial state q_0 . In terms of *definability*, the statement of Turing's halting problem ranges over all possible Turing machines and all possible finitely bounded initial tapes. This means: for each tape T and machine \mathcal{M} , there are finite initial conditions imposed on tape T and machine \mathcal{M} . However, as tape T and machine \mathcal{M} range over all possibilities, the computational resources required for tape T and machine \mathcal{M} are unbounded. Thus, the computational resources required by \mathcal{H} are unbounded as its input ranges over all finitely bounded initial tapes T and machines \mathcal{M} .

The previous paragraph provides some observations about Turing's halting problem because any philosophical objection to $\mathcal{Q}(x)$'s unbounded computational resources during an evolution should also present a similar philosophical objection to Turing's assumptions in his statement and proof of his halting problem. Notice that corollary 4.4 supports our claim.

Since $\mathcal{Q}(x)$ and every other ex-machine $\mathcal{Q}(a_0a_1 \dots a_m x)$ in \mathfrak{U} is not a Turing machine, there is a natural extension of Turing's halting problem. Does there exist an ex-machine $\mathcal{G}(x)$ such that for any given Turing machine \mathcal{M} and finite initial tape T , then $\mathcal{G}(x)$ can sometimes compute whether \mathcal{M} 's execution on tape T will eventually halt? Before we call this the *ex-machine halting problem*, the phrase *can sometimes compute whether* must be defined so that this problem is well-posed. A reasonable definition requires some work.

From the universal Turing machine / enumeration theorem [32, 77], there exists a Turing computable enumeration $\mathcal{E} : \mathbb{N} \rightarrow \{\text{all Turing machines } \mathcal{M}\} \times \{\text{Each of } \mathcal{M}'\text{'s states as an initial state}\}$ of every Turing machine. Similar to ex-machines, for each machine \mathcal{M} , the set $\{\text{Each of } \mathcal{M}'\text{'s states as an initial state}\}$ can be realized as a finite subset $\{0, \dots, n-1\}$ of \mathbb{N} . Since $\mathcal{E}(n)$ is an ordered pair, the phrase “Turing machine $\mathcal{E}(n)$ ” refers to the first coordinate of $\mathcal{E}(n)$. Similarly, the “initial state $\mathcal{E}(n)$ ” refers to the second coordinate of $\mathcal{E}(n)$.

Recall that the Turing machine halting problem is equivalent to the blank-tape halting problem. (See pages 150-151 of [68]). For our discussion, the blank-tape halting problem translates to: for each Turing machine $\mathcal{E}(n)$, does Turing machine $\mathcal{E}(n)$ halt when $\mathcal{E}(n)$ begins its execution with a blank initial tape and initial state $\mathcal{E}(n)$?

Lemma 4.1 implies that the same initial ex-machine can evolve to two different ex-machines; furthermore, these two ex-machines will never compute the same language no matter what descendants they evolve to. For example, $\mathcal{Q}(0 x)$ and $\mathcal{Q}(1 x)$ can never compute the same language in \mathfrak{L} . Hence, *sometimes* means that for each n , there exists an evolution of $\mathcal{G}(x)$ to $\mathcal{G}(a_0x)$, and then to $\mathcal{G}(a_0a_1x)$ and so on up to $\mathcal{G}(a_0a_1 \dots a_n x) \dots$, where for each i with $0 \leq i \leq n$, then $\mathcal{G}(a_0a_1 \dots a_n x)$ correctly computes whether Turing machine $\mathcal{E}(n)$ – executing on an initial blank tape with initial state $\mathcal{E}(n)$ – halts or does not halt.

In the prior sentence, the word *computes* means that $\mathcal{G}(a_0a_1 \dots a_i x)$ halts after a finite number of instructions have been executed and the halting output written by $\mathcal{G}(a_0a_1 \dots a_i x)$ on its tape indicates whether machine $\mathcal{E}(n)$ halts or does not halt. For example, if the input tape is $\# \mathbf{a}^i \#$, then enumeration machine $\mathcal{M}_{\mathcal{E}}$ writes the representation of $\mathcal{E}(i)$ on the tape, and then $\mathcal{G}(a_0a_1 \dots a_m x)$ with $m \geq i$ halts with $\# \mathbf{Y} \#$ written to the right of the representation for machine $\mathcal{E}(i)$. Alternatively $\mathcal{G}(a_0a_1 \dots a_m x)$ with $m \geq i$ halts with $\# \mathbf{N} \#$ written to the right of the representation for machine $\mathcal{E}(i)$. The word *correctly* means that ex-machine $\mathcal{G}(a_0a_1 \dots a_m x)$ halts with $\# \mathbf{Y} \#$ written on the tape if machine $\mathcal{E}(i)$ halts and ex-machine $\mathcal{G}(a_0a_1 \dots a_m x)$ halts with $\# \mathbf{N} \#$ written on the tape if machine $\mathcal{E}(i)$ does not halt.

Next, our goal is to transform the ex-machine halting problem to a form so that the results from the previous section can be applied. Choose the alphabet as $\mathcal{A} = \{\#, 0, 1, a, A, B, M, N, S, X, Y\}$. As before, for each Turing

machine, it is helpful to identify the set of machine states Q as a finite subset of \mathbb{N} . Let $\mathcal{M}_{\mathcal{E}}$ be the Turing machine that computes a Turing computable enumeration³ as $\mathcal{E}_a : \mathbb{N} \rightarrow \{\mathcal{A}\}^* \times \mathbb{N}$, where the tape $\# \# \mathbf{a}^n \#$ represents natural number n . Each $\mathcal{E}_a(n)$ is an ordered pair where the first coordinate is the Turing machine and the second coordinate is an initial state chosen from one of $\mathcal{E}_a(n)$'s states.

Remark 6.1. For each $n \in \mathbb{N}$, with blank initial tape and initial state $\mathcal{E}_a(n)$, then Turing machine $\mathcal{E}_a(n)$ either halts or does not halt.

Proof. The execution behavior of Turing machine computation is unambiguous. For each n , there are only two possibilities. \square

For our particular instance of \mathcal{E}_a , define the *halting function* $h_{\mathcal{E}_a} : \mathbb{N} \rightarrow \{0, 1\}$ as follows. For each n , set $h_{\mathcal{E}_a}(n) = 1$, whenever Turing machine $\mathcal{E}_a(n)$ with blank initial tape and initial state $\mathcal{E}_a(n)$ halts. Otherwise, set $h_{\mathcal{E}_a}(n) = 0$, if Turing machine $\mathcal{E}_a(n)$ with blank initial tape and initial state $\mathcal{E}_a(n)$ does not halt. Remark 6.1 implies that function $h_{\mathcal{E}_a}(n)$ is well-defined. Via the halting function $h_{\mathcal{E}_a}(n)$ and definition 4.1, define the *halting language* $L_{h_{\mathcal{E}_a}}$.

Theorem 6.1. *The ex-machine $\Omega(x)$ has an evolutionary path that computes halting language $L_{h_{\mathcal{E}_a}}$. This evolutionary path is $\Omega(h_{\mathcal{E}_a}(0) x) \rightarrow \Omega(h_{\mathcal{E}_a}(0) h_{\mathcal{E}_a}(1) x) \rightarrow \dots \Omega(h_{\mathcal{E}_a}(0) h_{\mathcal{E}_a}(1) \dots h_{\mathcal{E}_a}(m) x) \dots$*

Proof. Theorem 6.1 follows from the previous discussion, including the definition of halting function $h_{\mathcal{E}_a}(n)$ and halting language $L_{h_{\mathcal{E}_a}}$ and theorem 4.2. \square

6.1 Some Observations Based on Theorem 6.1

Although theorem 6.1 provides an affirmative answer to the ex-machine halting problem, in practice, a particular execution of $\Omega(x)$ will not, from a probabilistic perspective, evolve to compute $L_{h(\mathcal{E}_a)}$. For example, the probability is 2^{-128} that a particular execution of $\Omega(x)$ will evolve to $\Omega(a_0 a_1 \dots a_{127} x)$ so that $\Omega(a_0 a_1 \dots a_{99} x)$ correctly computes whether each string $\mathbf{a}^0, \mathbf{a}, \mathbf{a}^2 \dots \mathbf{a}^{127}$ is a member of $L_{h(\mathcal{E}_a)}$ or not a member of $L_{h(\mathcal{E}_a)}$.

Furthermore, theorem 6.1 provides no general method for infallibly testing (proving) that an evolution of $\Omega(x)$ to some new machine $\Omega(a_0 a_1 \dots a_m x)$ satisfies $a_i = h_{\mathcal{E}_a}(i)$ for each $0 \leq i \leq m$. We also know that any such general testing method that works for all natural numbers m would require at least an ex-machine (or some computational object more powerful than an ex-machine if that object exists) because any general testing method cannot be implemented with a standard machine. Otherwise, if such a testing method could be executed by a standard machine, then this special standard machine could be used to solve Turing's halting problem: this is logically impossible due to Turing's proof of the unsolvability of the halting problem with a standard machine.

Despite all of this, it is still *logically possible* for this evolution to happen, since $\Omega(x)$ can in principle evolve to compute any language L_f in \mathfrak{L} . In other words, every infinite, downward path in the infinite binary tree of figure 3 is possible and equally likely. Clearly, $\Omega(x)$ is *not an "intelligent" ex-machine*, by any reasonable definition of "intelligent", since evolutionary path $\Omega(h_{\mathcal{E}_a}(0) x) \rightarrow \Omega(h_{\mathcal{E}_a}(0) h_{\mathcal{E}_a}(1) x) \rightarrow \dots \Omega(h_{\mathcal{E}_a}(0) h_{\mathcal{E}_a}(1) \dots h_{\mathcal{E}_a}(m) x) \dots$ relies solely on blind luck.

In some ways, theorem 6.1 has an analogous result in pure mathematics. The Brouwer fixed point theorem [11] guarantees that a continuous map from an n -simplex to an n -simplex has at least one fixed point and demonstrates the power of algebraic topology [91]. However, the early proofs were indirect and provided no constructive methods for finding the fixed point(s). The parallel here is that theorem 6.1 guarantees that an

³Chapter 7 of [68] provides explicit details of encoding quintuples with a particular universal Turing machine. Alphabet \mathcal{A} was selected so that it is compatible with this encoding. A careful study of chapter 7 should provide a clear path of how $\mathcal{M}_{\mathcal{E}}$'s instructions can be specified to implement \mathcal{E}_a .

evolutionary path exists, but the proof provides no general method for infallibly testing that for an evolution up to stage m , then $\mathfrak{Q}(a_0 a_1 \dots a_m x)$ satisfies $a_i = h_{\mathcal{E}_a}(i)$ for each $0 \leq i \leq m$.

Algorithmic methods for finding fixed points were developed about 60 years later [81, 102]. The part of the analogy that has not yet played out could break down due to the extreme ramifications of reaching large enough sizes of m , whereby currently intractable problems in mathematics could be proven. However, this really depends upon the computing speeds and the ex-machine learning and mathematical methods developed over the next few centuries. We believe that deeper, ex-machine learning and mathematical methods could have a larger impact than hardware advances because a clever proof can save a large number of mechanical steps over a mediocre proof. And the clever use of a prior theorem or symmetry in a new proof can save an infinite number of computational steps.

With the history of the Brouwer fixed point theorem in mind, the logical possibility, demonstrated by theorem 6.1, suggests that there might be an opportunity to develop new problem solving methods and apply more advanced ex-machines to key instances of the halting problem. As far as more advanced ex-machines, a broad research direction is to explore the use of populations of ex-machines that evolve and also communicate formal languages with each other, analogous to the methods that human mathematicians use in their mathematical research.

The Goldbach conjecture states that every even number greater than 2 is the sum of two primes. This conjecture seems to be an intractable problem in number theory as Goldbach posed it [50] in the year 1742. Despite its apparent intractability, a fairly simple Turing machine can be specified; namely, proving that the following Goldbach machine never halts provides a proof of the conjecture.

Machine Instructions 3. A Goldbach Machine

```

set n = 2
set g = true
set prime_list = (2)
while (g == true)
{
    set n = n + 2
    set g = false
    for each p in prime_list
    {
        set x = n - p
        if (x is a member of prime_list) set g = true
    }

    if (n-1 is prime) store n-1 in prime_list
}

print ("Even number " n " is not the sum of two primes.")
print ("The Goldbach conjecture is false!")
halt

```

We wrap up this section with some advice from mathematician George Pólya. Pólya emphasizes the use of heuristics for mathematical problem solving, which may shed further light on what we broadly have in mind for ex-machines that help solve intractable math problems. Advanced ex-machines may help human mathematicians with the conception of a proof. A propos to our discussion, Pólya [72] distinguishes between conceiving of a proof versus verifying a proof:

The following pages are written somewhat concisely, but simply as possible, and are based on a long and serious study of methods of solution. This sort of study, called heuristic by some writers, is not in fashion nowadays but has a long past and, perhaps, some future.

Studying the methods of solving problems, we perceive another face of mathematics. Yes, mathematics has two faces; it is the rigorous science of Euclid but it is also something else. Mathematics presented in the Euclidean way appears as a systematic, deductive science; but mathematics in the making appears as an experimental, inductive science. Both aspects are as old as the science of mathematics itself.

7 Two Research Problems

Pólya expresses a broad vision, but without some concrete research problems aimless wandering is likely. We propose two mathematical problems, where the goal is to express each one as a halting problem for a single Turing machine. Furthermore, each problem has a different strategy for teaching us more about new ex-machine computation that advances beyond $\Omega(x)$'s blind luck.

Mathematical Problem 1.

Specify explicit initial instructions of an ex-machine that can evolve to compute a proof that $\sqrt{2}$ is irrational. One reason for proposing this problem is that we already know the correct answer. Another reason is that the human proof is short and clever. One possible ex-machine approach follows the traditional method of constructing a proof by contradiction based on the theorems about odd and even natural numbers.

A second approach is more involved. However, new techniques, gained from this approach, may be applicable to other instances of halting problems. Consider the pseudocode below that computes the $\sqrt{2}$, by executing a bisection algorithm on the curve $y = x^2 - 2$.

Machine Instructions 4. A $\sqrt{2}$ Standard Machine

```

Set l = 1
Set u = 2
Set a = true

while (a == true)
{
  set x = (l + u) / 2

  if ( (x * x) > 2)
    set l = x
  else
    set u = x

  if (the tape representation of x so far has a periodic sequence)
    and (the periodicity will continue indefinitely)
  {
    set a = false
  }
}
halt

```

The bisection approach searches for a periodic sequence of writing the tape symbols. The critical part is the instruction `if (the tape representation of x has a periodic sequence) and (the periodicity will continue indefinitely)`. How does an ex-machine evolve rules to recognize a periodic sequence of symbols (written on the tape by the $\sqrt{2}$ standard machine) and also guarantee that the periodic sequence on the tape will repeat indefinitely? In other words, the *ex-machine evolves rules that adequately represent knowledge about the dynamics of the $\sqrt{2}$ machine's instructions*.

A consecutive repeating state cycle in a Turing machine occurs when a finite sequence of standard machine instructions $\{I_i\}$ is executed by the Turing machine two consecutive times: $I_1 \rightarrow I_2 \rightarrow \dots I_k \rightarrow I_1 \rightarrow I_2 \dots I_k$ and the machine configuration before the first instruction I_1 is executed equals the machine configuration after the instruction I_k has completed its execution a second time.

In [42], the main theorem shows that consecutive repeating state cycles characterize the periodic points of a Turing machine. A periodic point that does not reach a halting state indicates that the Turing machine execution is immortal (i.e., never halts). Can this consecutive repeating state cycle theorem or an extension of this theorem be used to help an ex-machine find a proof? If the standard $\sqrt{2}$ machine writes symbols on the tape in a periodic sequence, this indicates that $\sqrt{2}$ is rational. If an ex-machine can construct rules which prove that the standard $\sqrt{2}$ machine never halts, then these ex-machine rules provide a proof that the $\sqrt{2}$ is irrational.

Mathematical Problem 2.

Transform Collatz machine 1's execution of each individual orbit $\mathcal{O}(f, n)$ into a single ex-machine computation that collectively makes a determination about all individual orbits. That is, find an ex-machine computation that evolves to a decision whether 1 is in $\mathcal{O}(f, n)$ for all $n \in \mathbb{N}$. Is it possible to accomplish this with an ex-machine computation? If it is impossible, why?

Consider the augmentation of Collatz machine 1 to an enumerated Collatz machine \mathcal{E} . The standard machine \mathcal{E} iterates over the odd numbers 3, 5, 7, \dots . \mathcal{E} first writes # #111# on the input tape and hands this computation over to Collatz machine 1. After Collatz machine 1 halts at 1, then \mathcal{E} updates the input tape to # #11111#, representing 5, and hands this to the Collatz machine again. After the Collatz machine halts at 1, then \mathcal{E} updates the input tape to # #1111111#, and so on. If the Collatz conjecture is true, this execution of \mathcal{E} never halts and \mathcal{E} iterates over every odd number.

At least part of the challenge with machine \mathcal{E} seems to be that there could exist some n such that n 's Collatz orbit reaches a periodic attractor that does not contain 1. Another possibility is that there exists some u whose Collatz orbit aperiodically oscillates and never reaches 1. In this case, u 's orbit does not have an upper bound. That is, $\sup \mathcal{O}(f, u) = \infty$. In both cases, the orbit of n and the orbit of u do not halt at 1. If the conjecture is true, how does one distinguish these two different types of immortal orbits from the enumerated Collatz machine that halts at 1 for each odd input, but is also immortal?

Is it possible to transform (either by human ingenuity or by ex-machine evolution or a combination) this enumerated Collatz machine \mathcal{E} into a non-vacuous, explicit Turing machine so that an immortal orbit proves or disproves that the Collatz conjecture is true? If this transformation exists, does there exist an ex-machine that can construct this transformation? Perhaps, the answers to these types of questions will provide some insight on a famous remark by mathematician Erdős about the Collatz conjecture. *Mathematics is not ready for such problems.*

References

- [1] C. Abellan, W. Amaya, M. Jofre, M. Curty, A. Acin, J. Capmany, V. Pruneri and M.W. Mitchell. Ultra-fast quantum randomness generation by accelerated phase diffusion in a pulsed laser diode. *Optics Express*. **22**(2), 2014, pp. 1645-1654.
- [2] Václav Alda. On 0-1 measures for projectors I. *Aplikace Matematiky*. **25**, 1980, pp. 373-374.
- [3] Václav Alda. On 0-1 measures for projectors II. *Aplikace Matematiky*. **26**, 1981, pp. 57-58.
- [4] John Bell. On the Einstein Podolsky Rosen Paradox. *Physics*. **1**, 1964, pp. 195-200.
- [5] Peter Bierhorst, Emanuel Knill, Scott Glancy, Yanbao Zhang, Alan Mink, Stephen Jordan, Andrea Rommal, Yi-Kai Liu, Bradley Christensen, Sae Woo Nam, Martin J. Stevens, and Lynden K. Shalm. Experimentally generated randomness certified by the impossibility of superluminal signals. *Nature*. **556**, 2018, pp. 223-226
- [6] David Bohm and Yakir Aharonov. Discussion of Experimental Proof for the Paradox of Einstein, Podolsky, and Rosen. *Physical Review*. **108**(4), November 1957, pp. 1070-1076.
- [7] David Bohm. Quantum Theory. Prentice-Hall, 1951, pp. 614-615.
- [8] Neil Bohr. Can Quantum-Mechanical Description of Physical Reality be Considered Complete? *Physical Review*. **48**, October 15, 1935, pp. 696-702.
- [9] Max Born and Pascual Jordan. Zur Quantenmechanik. *Zeitschrift für Physik*. **34**, 1925, pp. 858-888
- [10] Max Born. Quantenmechanik der Stoßvorgänge. *Zeitschrift für Physik*. **38**, 1926, pp. 803-827.
- [11] Luitzen Egbertus Jan Brouwer. Über ein eindeutige, stetige Transformationen von Flächen in sich. *Math Ann*. **69**. 1910, pp. 176-180.
- [12] Cristian S. Calude. *Information and Randomness. An Algorithmic Perspective*. Second Edition, Springer, 2002.
- [13] Cristian Calude and Karl Svozil. Quantum randomness and value indefiniteness. *Advanced Science Letters* **1**(2), 2008, pp. 165-168.
- [14] Alistair Abbott, Cristian Calude, Conder, J., Karl Svozil. Strong Kochen-Specker theorem and incomputability of quantum randomness. *Physical Review A*. **86**, 062109, 2012, pp. 1-11.
- [15] Alistair Abbott, Cristian Calude, Karl Svozil. Value Indefinite Observables are Almost Everywhere. *Physical Review A*. **89**, 032109, 2014.
- [16] Alistair Abbott, Cristian Calude, Karl Svozil. On the Unpredictability of Individual Quantum Measurement Outcomes. (Editors: Beklemishev L., Blass A., Dershowitz N., Finkbeiner B., Schulte W.) *Fields of Logic and Computation II. Lecture Notes in Computer Science*. **9300**, Springer, 2015.
- [17] Alistair Abbott, Cristian Calude, Michael Dineen, and Nan Huang. Experimental evidence of the superiority of quantum randomness over pseudo-randomness. 2017.
- [18] Georg Cantor. Ueber eine elementare Frage der Mannigfaltigkeitslehre. *Jahresbericht der Deutschen Mathematiker-Vereinigung*. Teubner 1891, pp. 75-78.

- [19] Georg Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers*. Translated by Philip E. Jourdain. New York: Dover Publications, 1955.
- [20] Gregory J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*. **13**, 1966, pp. 547-569.
- [21] Gregory J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations. *Journal of the ACM*. **16**, 1969, pp. 145-159.
- [22] Alonzo Church. An Unsolvable Problem of Elementary Number Theory. *The American Journal of Mathematics*. **58**, 1936, pp. 345-363.
- [23] Alonzo Church. The Calculi of Lambda-Conversion. *Annals of Mathematics*. **6**, Princeton University Press, 1941.
- [24] John Clauser, Michael Horne, Abner Shimony, and Richard Holt. Proposed Experiment to Test Local Hidden-Variable Theories. *Physical Review Letters*. **23**(15), October 13, 1969, pp. 880-884.
- [25] John Horton Conway. Unpredictable iterations. *Proceedings of the Number Theory Conference*. University of Colorado, Boulder, CO, 1972, pp. 4952.
- [26] Stephen A. Cook. The P versus NP Problem. Manuscript, 12 pages.
- [27] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, ACM*. 1971, pp. 151-158.
- [28] Pierre Curie et Marie Curie. Sur la radioactivit e provoqu ee par les rayons de Becquerel. *C.R. Acad. Sci. Paris*. **129**: November, 1899, pp. 714-716.
- [29] Pierre Curie et Marie Curie. Les nouvelles substances radioactives et les rayons quelles emettent. *Rapports pr esent es au Congres international de Physique* Gauthier-Villars, Paris: vol. III, 1900, pp. 79-114.
- [30] Martin Davis. *Computability and Unsolvability*. Dover, 1982.
- [31] George Marsaglia. Diehard Statistical Tests. 1996.
- [32] Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer. 2010, pp. 9-10.
- [33] Louis de Broglie. Recherches sur la th eorie des quanta. *Ph.D. Thesis*. Paris, 1924.
- [34] Louis de Broglie. Recherches sur la th eorie des quanta. *Ann. de Physique*. **10**, 3, 22, 1925.
- [35] Antony Eagle. Randomness Is Unpredictability. *British Journal of Philosophy of Science*. **56**, 2005, pp. 749-790.
- [36] Albert Einstein, Boris Podolsky and Nathan Rosen. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Physical Review*. **47**, 1935, pp. 777-780.
- [37] William Feller. *An Introduction to Probability Theory and Its Applications*. Volume 1, Third Edition. John Wiley & Sons, New York, 1968; pp. 202-211, ISBN 0 471 25708-7.
- [38] William Feller. *An Introduction to Probability Theory and Its Applications*. Volume 2, Third Edition. John Wiley & Sons, New York, 1966.

- [39] Richard Feynman. *The Feynman Lectures on Physics*. Volume III. Addison-Wesley Publishing, 1963.
- [40] Michael Stephen Fiske. *Non-autonomous Dynamical Systems Applicable to Neural Computation*. Northwestern University, 1996.
- [41] Michael Stephen Fiske. Turing Incomputable Computation. *Turing-100 Proceedings. Alan Turing Centenary*. EasyChair, **10**, 2012, pp. 66-91.
- [42] Michael Stephen Fiske. Consecutive Repeating State Cycles Determine Periodic Points in a Turing Machine. *Selected Topics in Nonlinear Dynamics and Theoretical Electrical Engineering*. **459**, 2013, pp. 393-417.
- [43] Michael Stephen Fiske. Quantum Random Active Element Machine. *Unconventional Computation and Natural Computation*. LNCS 7956. Springer, 2013, pp. 252-254.
- [44] Christian Gabriel, Christoffer Wittmann, Denis Sych, Ruifang Dong, Wolfgang Mauerer, Ulrik L Andersen, Christoph Marquardt, and Gerd Leuchs. A generator for unique quantum random numbers based on vacuum states. *Nature Photonics*. **4**, 2010, pp. 711-715.
- [45] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.
- [46] Walther Gerlach and Otto Stern. Der experimentelle Nachweis der Richtungsquantelung im Magnetfeld. *Zeitschrift für Physik*. **9**, 1922, pp. 349-352.
- [47] Walther Gerlach and Otto Stern. Das magnetische Moment des Silberatoms. *Zeitschrift für Physik*. **9**, 1922, pp. 353-355.
- [48] Walther Gerlach and Otto Stern. Der experimentelle Nachweis des magnetischen Moments des Silberatoms. *Zeitschrift für Physik*. **8**, 1922, pp. 110-111.
- [49] Kurt Gödel. *Collected Works. Volume II: Publications 1938-1974*. Edited by S. Fefferman, J Dawson, S. Kleene, G. Moore, R. Solovay, and J. Heijenoort. Oxford University Press, 1972, pp. 305-306.
- [50] Christian Goldbach. Letter from Goldbach to Euler. June 7, 1742.
- [51] Werner Heisenberg. Über quantentheoretische Umdeutung kinematischer und mechanischer Beziehungen. *Zeitschrift für Physik*. September, 1925.
- [52] Werner Heisenberg. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Zeitschrift für Physik*. **43**(3-4), 1927, pp. 172-198.
- [53] Werner Heisenberg. *The Physical Principles of the Quantum Theory*. University of Chicago Press, 1930.
- [54] Gudrum Kalmbach. *Measures and Hilbert Lattices*. World Scientific, Singapore, 1986.
- [55] S.C. Kleene. A theory of positive integers in formal logic. *American Journal of Mathematics*. **57**, 1935, pp. 153-173 and pp. 219-244.
- [56] Simon Kochen and Ernst Specker. Logical structures arising in quantum theory. *Symposium on the Theory of Models, Proceedings of the 1963 International Symposium at Berkeley*, North Holland, Amsterdam, 1965, pp. 177-189.

- [57] Simon Kochen and Ernst Specker. The calculus of partial propositional functions. *Proceedings of the 1964 International Congress for Logic, Methodology and Philosophy of Science, Jerusalem*, North Holland, Amsterdam, 1965, pp. 45-57.
- [58] Simon Kochen and Ernst Specker. The problem of hidden variables in quantum mechanics. *Journal of Mathematics and Mechanics*. **17**, 1967, pp. 59-87.
- [59] Andrey N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems Information Transmission*. **1**(1), 1965, pp. 1-7.
- [60] Anatoly Kulikov, Markus Jerger, Anton Potočník, Andreas Wallraff, and Arkady Fedorov. Realization of a Quantum Random Generator Certified with the Kochen-Specker Theorem. *Physical Review Letter*. **119**, 240501, December 11, 2017.
- [61] Jan-Ake Larsson. Loopholes in Bell Inequality Tests of Local Realism. *Journal of Physical Review A*. **47**, 424003, 2014.
- [62] Harry R. Lewis and Christos Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.
- [63] Hai-Qiang Ma, Yuejian Xie, and Ling-An Wu. Random number generation based on the time of arrival of single photons. *Applied Optics*. **44**, 2005, pp. 77607763.
- [64] Per Martin-Löf. The definition of random sequences. *Information and Control*. 1966, **9**, pp. 602-619.
- [65] John McCarthy. Recursive functions of symbolic expressions. *Communications of the ACM*, **3**, 1960, pp. 184-195.
- [66] John McCarthy. *The LISP 1.5 Programmer's Manual*. MIT Press, Cambridge, 1962.
- [67] John McCarthy. A basis for a mathematical theory of computation. *Computer Programming and Formal Systems*, Braffort and Hirschberg (Eds.), North-Holland, Amsterdam, 1963, pp. 33-70.
- [68] Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
- [69] Lutz Mueller. *newLISP*. <http://www.newlisp.org> 2018.
- [70] A. Rukhin, J. Soto, J. Nechvatal, et al. NIST Special Publication. 2010.
- [71] S. Pironio, A. Acin, S. Massar, A. Boyer de la Giroday, D. N. Matsukevich, P. Maunz, S. Olmschenk, D. Hayes, L. Luo, T. A. Manning, and C. Monroe. Random numbers certified by Bell's theorem. *Nature*. **464**, April 2010, pp. 1021-1024.
- [72] George Pólya. *How to Solve It. A New Aspect of Mathematical Method*. Princeton Science Library, Second Edition, 1957.
- [73] Karl Popper. *Logik der Forschung. Zur Erkenntnistheorie der modernen Naturwissenschaft*. Mohr Siebeck Verlag, 1934.
- [74] Emil L. Post. Recursive Unsolvability of a Problem of Thue. *The Journal of Symbolic Logic*, **12**, 1947, pp. 1-11.
- [75] Bernhard Riemann. Ueber die Anzahl der Primzahlen unter einer gegebenen Grösse. November 1859.

- [76] Markus Rohe. A True-Random Generator Based On Radioactive Decay. *Security and Cryptography Research Group. Saarland University*, 2003, pp. 1-36.
- [77] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.
- [78] Ernest Rutherford. Radioactivity produced in substances by the action of thorium compounds. *Philosophical Magazine*. S.5, **49**: February 1900, pp. 161-192.
- [79] Ernest Rutherford and F. Soddy. The cause and nature of radioactivity - Part I. *Philosophical Magazine*. S.6, **4**: September 1902, pp. 370-396.
- [80] Ernest Rutherford and F. Soddy. The cause and nature of radioactivity - Part II. *Philosophical Magazine*. S.6, **4**: November 1902, pp. 569-585.
- [81] Herbert Scarf. The Approximation of Fixed Points of a Continuous Mapping. *SIAM Journal of Applied Mathematics*. **15**(5), September 1967, pp. 1328-1343.
- [82] Alfred A. Schmitt. The State Complexity of Turing Machines. *Information and Control*. **17**, 1970, pp. 217-225.
- [83] Erwin Schrodinger. An Undulatory Theory of the Mechanics of Atoms and Molecules. *Physical Review*. **28**(6), December 1926, 1049-1070.
- [84] Erwin Schrodinger. *Naturwissenschaften*. **23**, 1935.
- [85] Lynden Shalm, et al. A Strong Loophole-Free Test of Local Realism. *Physical Review Letters*. **115**, 250402, December 16, 2015.
- [86] Claude Shannon. The Mathematical Theory of Communication. *Bell System Technical Journal*. **27**, 379, 632, 1948.
- [87] Claude Shannon. A Universal Turing Machine with Two Internal States. *Automata Studies*. (editors, C.E. Shannon and John McCarthy). Princeton University Press, 1956, pp. 129-153.
- [88] Michael Sipser. *Introduction to the Theory of Computation*. Third Edition. Cengage Learning, 2013.
- [89] Ray Solomonoff. A Preliminary Report on a General Theory of Inductive Inference. Report V-131, Cambridge, MA, Zator Company, November, 1960.
- [90] Ernst Specker. Die Logik nicht gleichzeitig entscheidbarer Aussagen. *Dialectica*. **14**, 1960, pp. 239-246.
- [91] Samuel Eilenberg and Norman Steenrod. *Foundations of Algebraic Topology*. Princeton University Press, 1952.
- [92] Andre Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. Optical quantum random number generator. *Journal of Modern Optics*. **47**, 2000, pp. 595598.
- [93] Mario Stipcevic and Rupert Ursin. An On-Demand Optical Quantum Random Number Generator with In-Future Action and Ultra-Fast Response. *Nature. Scientific Reports*. June 2015, pp. 1-8.
- [94] Karl Svozil and Josef Tkadlec. Greechie diagrams, nonexistence of measures in quantum logics and Kochen-Specker type constructions. *Journal of Mathematical Physics*. **37**, 1996, pp. 53805401.

- [95] Karl Svozil. Three criteria for quantum random-number generators based on beam splitters. *Physical Review A*, **79**, 054306, May 2009.
- [96] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. Series 2* **42** (Parts 3 and 4), 1936, pp. 230-265. A correction, *ibid.* **43**, 1937, pp. 544-546.
- [97] Alan M. Turing. Systems of Logic Based on Ordinals. *Proceedings of the London Mathematical Society*. **45**(2), 1939, pp. 161-228.
- [98] John von Neumann. *Mathematical Foundations of Quantum Mechanics*. Princeton University Press, 1955.
- [99] Garrett Birkhoff and John von Neumann. The logic of quantum mechanics. *Annals of Mathematics*. **37**, 1936, pp. 823-843.
- [100] Michael Wahl, Matthias Leifgen, Michael Berlin, Tino Rohlicke1, Hans-Jurgen Rahn, and Oliver Benson. An ultrafast quantum random number generator with provably bounded output bias based on photon arrival time measurements. *Applied Physics Letters*. **98**, 171105, 2011.
- [101] Jie Yang, Jinlu Liu, Qi Su, Zhengyu Li, Fan Fan, Bingjie Xu and Hong Guo. A 5.4 Gbps real time quantum random number generator with compact implementation. *Optics Express*. **24**(24), 2016, pp. 27475-27481.
- [102] R. B. Kellogg, T. Y. Li, and J. Yorke. A Constructive Proof of the Brouwer Fixed-Point Theorem and Computational Results. *SIAM Journal on Numerical Analysis*. **13**(4), 1976, pp. 473-483.
- [103] Neal Zierler and Michael Schlessinger. Boolean Embeddings of Orthomodular Sets and Quantum Logic. *Duke Mathematical Journal*. **32**, 1965, pp. 251-262.

8 Appendix – Execution of the Collatz Machine on $n = 5$

Each row shows the current tape and machine state after the instruction in that row has been executed. Before the machine starts executing, the initial tape is # 11111# and the initial state is q. The space indicates the location of the tape head.

STATE	TAPE	TAPE HEAD	INSTRUCTION	COMMENTS
a	## 11111#####	1	(q, #, a, #, 1)	
b	##1 1111#####	2	(a, 1, b, 1, 1)	
c	##11 111#####	3	(b, 1, c, 1, 1)	
d	##111 11#####	4	(c, 1, d, 1, 1)	
c	##1111 1#####	5	(d, 1, c, 1, 1)	
d	##11111 #####	6	(c, 1, d, 1, 1)	
k	##11111 1#####	5	(d, #, k, #, -1)	Compute 3*5+1
l	##11110 #####	6	(k, 1, l, 0, 1)	
m	##111100 #####	7	(l, #, m, 0, 1)	
k	##11110 00#####	6	(m, #, k, 0, -1)	
k	##1111 000#####	5	(k, 0, k, 0, -1)	
k	##111 1000#####	4	(k, 0, k, 0, -1)	
l	##1110 000#####	5	(k, 1, l, 0, 1)	
l	##11100 00#####	6	(l, 0, l, 0, 1)	
l	##111000 0#####	7	(l, 0, l, 0, 1)	
l	##1110000 #####	8	(l, 0, l, 0, 1)	
m	##11100000 #####	9	(l, #, m, 0, 1)	
k	##1110000 00#####	8	(m, #, k, 0, -1)	
k	##1110000 000#####	7	(k, 0, k, 0, -1)	
k	##11100 0000#####	6	(k, 0, k, 0, -1)	
k	##1110 00000#####	5	(k, 0, k, 0, -1)	
k	##111 000000#####	4	(k, 0, k, 0, -1)	
k	##11 1000000#####	3	(k, 0, k, 0, -1)	
l	##110 000000#####	4	(k, 1, l, 0, 1)	
l	##1100 00000#####	5	(l, 0, l, 0, 1)	
l	##11000 0000#####	6	(l, 0, l, 0, 1)	
l	##110000 000#####	7	(l, 0, l, 0, 1)	
l	##1100000 00#####	8	(l, 0, l, 0, 1)	
l	##11000000 0#####	9	(l, 0, l, 0, 1)	
l	##110000000 #####	10	(l, 0, l, 0, 1)	
m	##1100000000 #####	11	(l, #, m, 0, 1)	
k	##1100000000 00####	10	(m, #, k, 0, -1)	
k	##11000000 000###	9	(k, 0, k, 0, -1)	
k	##1100000 0000###	8	(k, 0, k, 0, -1)	
k	##1100000 00000###	7	(k, 0, k, 0, -1)	
k	##11000 000000###	6	(k, 0, k, 0, -1)	
k	##1100 0000000###	5	(k, 0, k, 0, -1)	
k	##110 00000000###	4	(k, 0, k, 0, -1)	
k	##11 000000000###	3	(k, 0, k, 0, -1)	
k	##1 1000000000###	2	(k, 0, k, 0, -1)	
l	##10 000000000###	3	(k, 1, l, 0, 1)	
l	##100 00000000###	4	(l, 0, l, 0, 1)	
l	##1000 00000000###	5	(l, 0, l, 0, 1)	
l	##10000 00000000###	6	(l, 0, l, 0, 1)	
l	##100000 00000###	7	(l, 0, l, 0, 1)	
l	##1000000 0000###	8	(l, 0, l, 0, 1)	
l	##10000000 000###	9	(l, 0, l, 0, 1)	
l	##100000000 00###	10	(l, 0, l, 0, 1)	
l	##1000000000 0###	11	(l, 0, l, 0, 1)	
l	##10000000000 ###	12	(l, 0, l, 0, 1)	
m	##100000000000 ##	13	(l, #, m, 0, 1)	
k	##10000000000 00#	12	(m, #, k, 0, -1)	
k	##1000000000 000#	11	(k, 0, k, 0, -1)	
k	##100000000 0000#	10	(k, 0, k, 0, -1)	
k	##10000000 00000#	9	(k, 0, k, 0, -1)	
k	##1000000 000000#	8	(k, 0, k, 0, -1)	

STATE	TAPE	TAPE HEAD	INSTRUCTION	COMMENTS
k	##100000 0000000####	7	(k, 0, k, 0, -1)	
k	##10000 00000000####	6	(k, 0, k, 0, -1)	
k	##1000 000000000####	5	(k, 0, k, 0, -1)	
k	##100 0000000000####	4	(k, 0, k, 0, -1)	
k	##10 00000000000####	3	(k, 0, k, 0, -1)	
k	##1 000000000000####	2	(k, 0, k, 0, -1)	
k	## 1000000000000####	1	(k, 0, k, 0, -1)	
l	##0 000000000000####	2	(k, 1, l, 0, 1)	
l	##00 0000000000####	3	(l, 0, l, 0, 1)	
l	##000 000000000####	4	(l, 0, l, 0, 1)	
l	##0000 00000000####	5	(l, 0, l, 0, 1)	
l	##00000 0000000####	6	(l, 0, l, 0, 1)	
l	##000000 000000####	7	(l, 0, l, 0, 1)	
l	##0000000 00000####	8	(l, 0, l, 0, 1)	
l	##00000000 0000####	9	(l, 0, l, 0, 1)	
l	##000000000 000####	10	(l, 0, l, 0, 1)	
l	##0000000000 00####	11	(l, 0, l, 0, 1)	
l	##00000000000 0####	12	(l, 0, l, 0, 1)	
l	##000000000000 0####	13	(l, 0, l, 0, 1)	
l	##0000000000000 #####	14	(l, 0, l, 0, 1)	
m	##00000000000000 ##	15	(l, #, m, 0, 1)	
k	##0000000000000 00##	14	(m, #, k, 0, -1)	
k	##0000000000000 000##	13	(k, 0, k, 0, -1)	
k	##00000000000 0000##	12	(k, 0, k, 0, -1)	
k	##0000000000 00000##	11	(k, 0, k, 0, -1)	
k	##000000000 000000##	10	(k, 0, k, 0, -1)	
k	##00000000 0000000##	9	(k, 0, k, 0, -1)	
k	##0000000 00000000##	8	(k, 0, k, 0, -1)	
k	##000000 000000000##	7	(k, 0, k, 0, -1)	
k	##00000 000000000##	6	(k, 0, k, 0, -1)	
k	##0000 0000000000##	5	(k, 0, k, 0, -1)	
k	##000 00000000000##	4	(k, 0, k, 0, -1)	
k	##00 000000000000##	3	(k, 0, k, 0, -1)	
k	##0 0000000000000##	2	(k, 0, k, 0, -1)	
k	## 00000000000000##	1	(k, 0, k, 0, -1)	
k	# 00000000000000##	0	(k, 0, k, 0, -1)	
n	## 00000000000000##	1	(k, #, n, #, 1)	
n	##1 0000000000000##	2	(n, 0, n, 1, 1)	
n	##11 00000000000##	3	(n, 0, n, 1, 1)	
n	##111 0000000000##	4	(n, 0, n, 1, 1)	
n	##1111 000000000##	5	(n, 0, n, 1, 1)	
n	##11111 00000000##	6	(n, 0, n, 1, 1)	
n	##111111 0000000##	7	(n, 0, n, 1, 1)	
n	##1111111 000000##	8	(n, 0, n, 1, 1)	
n	##11111111 00000##	9	(n, 0, n, 1, 1)	
n	##111111111 0000##	10	(n, 0, n, 1, 1)	
n	##1111111111 0000##	11	(n, 0, n, 1, 1)	
n	##11111111111 000##	12	(n, 0, n, 1, 1)	
n	##111111111111 00##	13	(n, 0, n, 1, 1)	
n	##1111111111111 0##	14	(n, 0, n, 1, 1)	
n	##11111111111111 0##	15	(n, 0, n, 1, 1)	
n	##111111111111111 ##	16	(n, 0, n, 1, 1)	
f	##11111111111111 10#	15	(n, #, f, 0, -1)	Compute 16/2
f	##1111111111111 110#	14	(f, 1, f, 1, -1)	
f	##11111111111 1110#	13	(f, 1, f, 1, -1)	
f	##111111111 11110#	12	(f, 1, f, 1, -1)	
f	##11111111 111110#	11	(f, 1, f, 1, -1)	
f	##1111111 1111110#	10	(f, 1, f, 1, -1)	
f	##111111 11111110#	9	(f, 1, f, 1, -1)	
f	##11111 111111110#	8	(f, 1, f, 1, -1)	
f	##1111 1111111110#	7	(f, 1, f, 1, -1)	
f	##111 11111111110#	6	(f, 1, f, 1, -1)	

STATE	TAPE	TAPE HEAD	INSTRUCTION
f	##1111 11111111110#	5	(f, 1, f, 1, -1)
f	##111 111111111110#	4	(f, 1, f, 1, -1)
f	##11 1111111111110#	3	(f, 1, f, 1, -1)
f	##1 11111111111110#	2	(f, 1, f, 1, -1)
f	## 111111111111110#	1	(f, 1, f, 1, -1)
f	# 1111111111111110#	0	(f, 1, f, 1, -1)
g	## 111111111111110#	1	(f, #, g, #, 1)
i	### 111111111111110#	2	(g, 1, i, #, 1)
i	###1 11111111111110#	3	(i, 1, i, 1, 1)
i	###11 1111111111110#	4	(i, 1, i, 1, 1)
i	###111 1111111111110#	5	(i, 1, i, 1, 1)
i	###1111 111111111110#	6	(i, 1, i, 1, 1)
i	###11111 11111111110#	7	(i, 1, i, 1, 1)
i	###111111 1111111110#	8	(i, 1, i, 1, 1)
i	###1111111 111111110#	9	(i, 1, i, 1, 1)
i	###11111111 11111110#	10	(i, 1, i, 1, 1)
i	###111111111 1111110#	11	(i, 1, i, 1, 1)
i	###1111111111 11110#	12	(i, 1, i, 1, 1)
i	###11111111111 1110#	13	(i, 1, i, 1, 1)
i	###111111111111 110#	14	(i, 1, i, 1, 1)
i	###1111111111111 10#	15	(i, 1, i, 1, 1)
i	###11111111111111 0#	16	(i, 1, i, 1, 1)
e	###1111111111111 10#	15	(i, 0, e, 0, -1)
f	###11111111111 100#	14	(e, 1, f, 0, -1)
f	###11111111111 1100#	13	(f, 1, f, 1, -1)
f	###1111111111 1100#	12	(f, 1, f, 1, -1)
f	###11111111 11100#	11	(f, 1, f, 1, -1)
f	###1111111 111100#	10	(f, 1, f, 1, -1)
f	###111111 1111100#	9	(f, 1, f, 1, -1)
f	###11111 11111100#	8	(f, 1, f, 1, -1)
f	###1111 111111100#	7	(f, 1, f, 1, -1)
f	###111 1111111100#	6	(f, 1, f, 1, -1)
f	###11 11111111100#	5	(f, 1, f, 1, -1)
f	###1 111111111100#	4	(f, 1, f, 1, -1)
f	### 1111111111100#	3	(f, 1, f, 1, -1)
f	## 11111111111100#	2	(f, 1, f, 1, -1)
f	# 111111111111100#	1	(f, 1, f, 1, -1)
g	### 11111111111100#	2	(f, #, g, #, 1)
i	### 11111111111100#	3	(g, 1, i, #, 1)
i	###1 1111111111100#	4	(i, 1, i, 1, 1)
i	###11 111111111100#	5	(i, 1, i, 1, 1)
i	###111 11111111100#	6	(i, 1, i, 1, 1)
i	###1111 1111111100#	7	(i, 1, i, 1, 1)
i	###11111 1111111100#	8	(i, 1, i, 1, 1)
i	###111111 111111100#	9	(i, 1, i, 1, 1)
i	###1111111 11111100#	10	(i, 1, i, 1, 1)
i	###11111111 111100#	11	(i, 1, i, 1, 1)
i	###111111111 11100#	12	(i, 1, i, 1, 1)
i	###1111111111 1100#	13	(i, 1, i, 1, 1)
i	###11111111111 100#	14	(i, 1, i, 1, 1)
i	###111111111111 00#	15	(i, 1, i, 1, 1)
e	###11111111111 100#	14	(i, 0, e, 0, -1)
f	###1111111111 1000#	13	(e, 1, f, 0, -1)
f	###11111111 11000#	12	(f, 1, f, 1, -1)
f	###1111111 111000#	11	(f, 1, f, 1, -1)
f	###111111 1111000#	10	(f, 1, f, 1, -1)
f	###11111 11111000#	9	(f, 1, f, 1, -1)
f	###1111 111111000#	8	(f, 1, f, 1, -1)
f	###111 1111111000#	7	(f, 1, f, 1, -1)
f	###11 11111111000#	6	(f, 1, f, 1, -1)
f	###1 111111111000#	5	(f, 1, f, 1, -1)
f	### 1111111111000#	4	(f, 1, f, 1, -1)

STATE	TAPE	TAPE HEAD	INSTRUCTION
f	#### 11111111110000#	3	(f, 1, f, 1, -1)
f	### #11111111110000#	2	(f, 1, f, 1, -1)
g	#### 11111111110000#	3	(f, #, g, #, 1)
i	##### 1111111110000#	4	(g, 1, i, #, 1)
i	#####1 111111110000#	5	(i, 1, i, 1, 1)
i	#####11 11111110000#	6	(i, 1, i, 1, 1)
i	#####111 1111110000#	7	(i, 1, i, 1, 1)
i	#####1111 111110000#	8	(i, 1, i, 1, 1)
i	#####11111 11110000#	9	(i, 1, i, 1, 1)
i	#####111111 1110000#	10	(i, 1, i, 1, 1)
i	#####1111111 110000#	11	(i, 1, i, 1, 1)
i	#####11111111 10000#	12	(i, 1, i, 1, 1)
i	#####111111111 0000#	13	(i, 1, i, 1, 1)
i	#####1111111111 000#	14	(i, 1, i, 1, 1)
e	#####111111111 1000#	13	(i, 0, e, 0, -1)
f	#####11111111 10000#	12	(e, 1, f, 0, -1)
f	#####1111111 110000#	11	(f, 1, f, 1, -1)
f	#####111111 1110000#	10	(f, 1, f, 1, -1)
f	#####11111 11110000#	9	(f, 1, f, 1, -1)
f	#####1111 111110000#	8	(f, 1, f, 1, -1)
f	#####111 1111110000#	7	(f, 1, f, 1, -1)
f	#####11 11111110000#	6	(f, 1, f, 1, -1)
f	#####1 111111110000#	5	(f, 1, f, 1, -1)
f	##### 1111111110000#	4	(f, 1, f, 1, -1)
f	#### #1111111110000#	3	(f, 1, f, 1, -1)
g	##### 111111110000#	4	(f, #, g, #, 1)
i	##### 111111110000#	5	(g, 1, i, #, 1)
i	#####1 11111110000#	6	(i, 1, i, 1, 1)
i	#####11 1111110000#	7	(i, 1, i, 1, 1)
i	#####111 111110000#	8	(i, 1, i, 1, 1)
i	#####1111 11110000#	9	(i, 1, i, 1, 1)
i	#####11111 1110000#	10	(i, 1, i, 1, 1)
i	#####111111 110000#	11	(i, 1, i, 1, 1)
i	#####1111111 10000#	12	(i, 1, i, 1, 1)
i	#####11111111 0000#	13	(i, 1, i, 1, 1)
e	#####1111111 10000#	12	(i, 0, e, 0, -1)
f	#####111111 100000#	11	(e, 1, f, 0, -1)
f	#####11111 1100000#	10	(f, 1, f, 1, -1)
f	#####1111 11100000#	9	(f, 1, f, 1, -1)
f	#####111 111100000#	8	(f, 1, f, 1, -1)
f	#####11 1111100000#	7	(f, 1, f, 1, -1)
f	#####1 11111100000#	6	(f, 1, f, 1, -1)
f	##### 111111100000#	5	(f, 1, f, 1, -1)
f	#### #111111100000#	4	(f, 1, f, 1, -1)
g	##### 111111100000#	5	(f, #, g, #, 1)
i	##### 11111100000#	6	(g, 1, i, #, 1)
i	#####1 1111100000#	7	(i, 1, i, 1, 1)
i	#####11 111100000#	8	(i, 1, i, 1, 1)
i	#####111 11100000#	9	(i, 1, i, 1, 1)
i	#####1111 1100000#	10	(i, 1, i, 1, 1)
i	#####11111 100000#	11	(i, 1, i, 1, 1)
i	#####111111 00000#	12	(i, 1, i, 1, 1)
e	#####11111 100000#	11	(i, 0, e, 0, -1)
f	#####1111 1000000#	10	(e, 1, f, 0, -1)
f	#####111 11000000#	9	(f, 1, f, 1, -1)
f	#####11 111000000#	8	(f, 1, f, 1, -1)
f	#####1 1111000000#	7	(f, 1, f, 1, -1)
f	##### 11111000000#	6	(f, 1, f, 1, -1)
f	##### #11111000000#	5	(f, 1, f, 1, -1)
g	##### 11111000000#	6	(f, #, g, #, 1)
i	##### 1111000000#	7	(g, 1, i, #, 1)
i	#####1 111000000#	8	(i, 1, i, 1, 1)

STATE	TAPE	TAPE HEAD	INSTRUCTION	COMMENTS
i	#####11 11000000#	9	(i, 1, i, 1, 1)	
i	#####111 1000000#	10	(i, 1, i, 1, 1)	
i	#####1111 000000#	11	(i, 1, i, 1, 1)	
e	#####111 1000000#	10	(i, 0, e, 0, -1)	
f	#####11 10000000#	9	(e, 1, f, 0, -1)	
f	#####1 110000000#	8	(f, 1, f, 1, -1)	
f	##### 1110000000#	7	(f, 1, f, 1, -1)	
f	##### #1110000000#	6	(f, 1, f, 1, -1)	
g	##### 1110000000#	7	(f, #, g, #, 1)	
i	##### 1100000000#	8	(g, 1, i, #, 1)	
i	#####1 10000000#	9	(i, 1, i, 1, 1)	
i	#####11 0000000#	10	(i, 1, i, 1, 1)	
e	#####1 10000000#	9	(i, 0, e, 0, -1)	
f	##### 100000000#	8	(e, 1, f, 0, -1)	
f	##### #100000000#	7	(f, 1, f, 1, -1)	
g	##### 100000000#	8	(f, #, g, #, 1)	
i	##### 00000000#	9	(g, 1, i, #, 1)	
e	##### #00000000#	8	(i, 0, e, 0, -1)	
g	##### 00000000#	9	(e, #, g, #, 1)	
g	#####1 0000000#	10	(g, 0, g, 1, 1)	
g	#####11 000000#	11	(g, 0, g, 1, 1)	
g	#####111 00000#	12	(g, 0, g, 1, 1)	
g	#####1111 0000#	13	(g, 0, g, 1, 1)	
g	#####11111 000#	14	(g, 0, g, 1, 1)	
g	#####111111 00#	15	(g, 0, g, 1, 1)	
g	#####1111111 0#	16	(g, 0, g, 1, 1)	
g	#####11111111 #	17	(g, 0, g, 1, 1)	
j	#####1111111 1#	16	(g, #, j, #, -1)	
j	#####111111 11#	15	(j, 1, j, 1, -1)	
j	#####11111 111#	14	(j, 1, j, 1, -1)	
j	#####1111 1111#	13	(j, 1, j, 1, -1)	
j	#####111 11111#	12	(j, 1, j, 1, -1)	
j	#####11 111111#	11	(j, 1, j, 1, -1)	
j	#####1 1111111#	10	(j, 1, j, 1, -1)	
j	##### 11111111#	9	(j, 1, j, 1, -1)	
j	##### #11111111#	8	(j, 1, j, 1, -1)	
a	##### 11111111#	9	(j, #, a, #, 1)	Completed 16/2.
b	#####1 1111111#	10	(a, 1, b, 1, 1)	
c	#####11 111111#	11	(b, 1, c, 1, 1)	
d	#####111 11111#	12	(c, 1, d, 1, 1)	
c	#####1111 1111#	13	(d, 1, c, 1, 1)	
d	#####11111 111#	14	(c, 1, d, 1, 1)	
c	#####111111 11#	15	(d, 1, c, 1, 1)	
d	#####1111111 1#	16	(c, 1, d, 1, 1)	
c	#####11111111 #	17	(d, 1, c, 1, 1)	
e	#####1111111 1#	16	(c, #, e, #, -1)	Compute 8 / 2
f	#####11111 10#	15	(e, 1, f, 0, -1)	
f	#####11111 110#	14	(f, 1, f, 1, -1)	
f	#####1111 1110#	13	(f, 1, f, 1, -1)	
f	#####111 11110#	12	(f, 1, f, 1, -1)	
f	#####11 111110#	11	(f, 1, f, 1, -1)	
f	#####1 1111110#	10	(f, 1, f, 1, -1)	
f	##### 11111110#	9	(f, 1, f, 1, -1)	
f	##### #11111110#	8	(f, 1, f, 1, -1)	
g	##### 11111110#	9	(f, #, g, #, 1)	
i	##### 1111110#	10	(g, 1, i, #, 1)	
i	#####1 111110#	11	(i, 1, i, 1, 1)	
i	#####11 11110#	12	(i, 1, i, 1, 1)	
i	#####111 1110#	13	(i, 1, i, 1, 1)	
i	#####1111 110#	14	(i, 1, i, 1, 1)	
i	#####11111 10#	15	(i, 1, i, 1, 1)	
i	#####111111 0#	16	(i, 1, i, 1, 1)	

STATE	TAPE	TAPE HEAD	INSTRUCTION	COMMENTS
e	#####11111 10#	15	(i, 0, e, 0, -1)	
f	#####1111 100#	14	(e, 1, f, 0, -1)	
f	#####111 1100#	13	(f, 1, f, 1, -1)	
f	#####11 11100#	12	(f, 1, f, 1, -1)	
f	#####1 111100#	11	(f, 1, f, 1, -1)	
f	##### 1111100#	10	(f, 1, f, 1, -1)	
f	##### #1111100#	9	(f, 1, f, 1, -1)	
g	##### 1111100#	10	(f, #, g, #, 1)	
i	##### 111100#	11	(g, 1, i, #, 1)	
i	#####1 11100#	12	(i, 1, i, 1, 1)	
i	#####11 1100#	13	(i, 1, i, 1, 1)	
i	#####111 100#	14	(i, 1, i, 1, 1)	
i	#####1111 00#	15	(i, 1, i, 1, 1)	
e	#####111 100#	14	(i, 0, e, 0, -1)	
f	#####11 1000#	13	(e, 1, f, 0, -1)	
f	#####1 11000#	12	(f, 1, f, 1, -1)	
f	##### 111000#	11	(f, 1, f, 1, -1)	
f	##### #111000#	10	(f, 1, f, 1, -1)	
g	##### 111000#	11	(f, #, g, #, 1)	
i	##### 11000#	12	(g, 1, i, #, 1)	
i	#####1 1000#	13	(i, 1, i, 1, 1)	
i	#####11 000#	14	(i, 1, i, 1, 1)	
e	#####1 1000#	13	(i, 0, e, 0, -1)	
f	##### 10000#	12	(e, 1, f, 0, -1)	
f	##### #10000#	11	(f, 1, f, 1, -1)	
g	##### 10000#	12	(f, #, g, #, 1)	
i	##### 0000#	13	(g, 1, i, #, 1)	
e	##### #0000#	12	(i, 0, e, 0, -1)	
g	##### 0000#	13	(e, #, g, #, 1)	
g	#####1 000#	14	(g, 0, g, 1, 1)	
g	#####11 00#	15	(g, 0, g, 1, 1)	
g	#####111 0#	16	(g, 0, g, 1, 1)	
g	#####1111 #	17	(g, 0, g, 1, 1)	
j	#####111 1#	16	(g, #, j, #, -1)	
j	#####11 11#	15	(j, 1, j, 1, -1)	
j	#####1 111#	14	(j, 1, j, 1, -1)	
j	##### 1111#	13	(j, 1, j, 1, -1)	
j	##### #1111#	12	(j, 1, j, 1, -1)	Completed 8/2
a	##### 1111#	13	(j, #, a, #, 1)	
b	#####1 111#	14	(a, 1, b, 1, 1)	
c	#####11 11#	15	(b, 1, c, 1, 1)	
d	#####111 1#	16	(c, 1, d, 1, 1)	
c	#####1111 #	17	(d, 1, c, 1, 1)	
e	#####111 1#	16	(c, #, e, #, -1)	Compute 4/2
f	#####11 10#	15	(e, 1, f, 0, -1)	
f	#####1 110#	14	(f, 1, f, 1, -1)	
f	##### 1110#	13	(f, 1, f, 1, -1)	
f	##### #1110#	12	(f, 1, f, 1, -1)	
g	##### 1110#	13	(f, #, g, #, 1)	
i	##### 110#	14	(g, 1, i, #, 1)	
i	#####1 10#	15	(i, 1, i, 1, 1)	
i	#####11 0#	16	(i, 1, i, 1, 1)	
e	#####1 10#	15	(i, 0, e, 0, -1)	
f	##### 100#	14	(e, 1, f, 0, -1)	
f	##### #100#	13	(f, 1, f, 1, -1)	
g	##### 100#	14	(f, #, g, #, 1)	
i	##### 00#	15	(g, 1, i, #, 1)	
e	##### #00#	14	(i, 0, e, 0, -1)	
g	##### 00#	15	(e, #, g, #, 1)	
g	#####1 0#	16	(g, 0, g, 1, 1)	
g	#####11 #	17	(g, 0, g, 1, 1)	

STATE	TAPE	TAPE HEAD	INSTRUCTION	COMMENTS
j	#####1 #	16	(g, #, j, #, -1)	
j	##### 11#	15	(j, 1, j, 1, -1)	
j	##### #11#	14	(j, 1, j, 1, -1)	Completed 4/2
a	##### 11#	15	(j, #, a, #, 1)	
b	#####1 #	16	(a, 1, b, 1, 1)	
c	#####11 #	17	(b, 1, c, 1, 1)	
e	#####1 #	16	(c, #, e, #, -1)	Compute 2/2
f	##### 10#	15	(e, 1, f, 0, -1)	
f	##### #10#	14	(f, 1, f, 1, -1)	
g	##### 10#	15	(f, #, g, #, 1)	
i	##### 0#	16	(g, 1, i, #, 1)	
e	##### #0#	15	(i, 0, e, 0, -1)	
g	##### 0#	16	(e, #, g, #, 1)	
g	#####1 #	17	(g, 0, g, 1, 1)	
j	##### 1#	16	(g, #, j, #, -1)	
j	##### #1#	15	(j, 1, j, 1, -1)	
a	##### 1#	16	(j, #, a, #, 1)	
b	#####1 #	17	(a, 1, b, 1, 1)	
h	##### 1#	16	(b, #, h, #, -1)	n = 5 orbit reaches 1