Turing Incomputable Computation

Michael Stephen Fiske

Aemea Institute, San Francisco, California, U.S.A.
mf@aemea.org

Abstract

A new computing model, called the active element machine (AEM), is presented that demonstrates Turing incomputable computation using quantum random input. The AEM deterministically executes a universal Turing machine (UTM) program $\overline{\eta}$ with random active element firing patterns. These firing patterns are Turing incomputable when the AEM executes a UTM having an unbounded number of computable steps. For an unbounded number of computable steps, if zero information is revealed to an adversary about the AEM's representation of the UTM's state and tape and the quantum random bits that help determine $\overline{\eta}$'s computation and zero information is revealed about the dynamic connections between the active elements, then there does not exist a "reverse engineer" Turing machine that can map the random firing patterns back to the sequence of UTM instructions. This casts a new light on Turing's notion of a *computational procedure*. In practical terms, these methods present an opportunity to build a new class of computing machines where the program's computational steps are hidden. This non-Turing computing behavior may be useful in cybersecurity and in other areas such as machine learning where multiple, dynamic interpretations of firing patterns may be applicable.

1 Introduction

Recent cyberattacks have demonstrated that current approaches to the malware problem (e.g., detection) are inadequate. This is not surprising as malware detection is Turing undecidable [6]. Further, some recent malware implementations use NP problems to encrypt and hide the malware [17]. Two goals guide an alternative approach: (a) Program execution should hide computational steps in order to hinder "reverse engineering" efforts by malware hackers; (b) New computational models should be explored that make it more difficult to hijack the purpose of program execution. The methods explained here pertain to (a).

Turing incomputable ([4], [7], [8]) computation is created using quantum random input. A new computing machine, called the active element machine (AEM), is presented that uses quantum randomness ([42], [43]) to deterministically execute a universal Turing machine (UTM) program $\overline{\eta}$ (table 2) with random firing interpretations. The active element machine computing model was introduced in [18], is summarized in section 2 and is further described in subsection 7.2 of the appendix. Furthermore, the active element firing patterns – computing the execution of the UTM program – are Turing incomputable when the UTM computation is non-halting or when the UTM computation is executed repeatedly on an unbounded number of computable initial configurations that halt.

The incomputability of the firing patterns depends on two quantum randomness axioms.

1. *No bias.* A single outcome x_i of a bit sequence $(x_1 x_2 \dots)$ generated by quantum randomness is unbiased: $P(x_i = 1) = P(x_i = 0) = \frac{1}{2}$.

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

Let $\Omega = \{(b_1 b_2 \dots) : b_i \in \{0,1\}\}$ be the space of infinite sequences of 0's and 1's representing infinite quantum random bit sequences. From these two axioms, in [4] Calude and Szovil conclude that if a quantum system producing the quantum randomness runs under ideal conditions to infinity, then the resulting infinite sequence of 0's and 1's (i.e., sequence in Ω) is Turing incomputable. In other words, no Turing machine can exactly reproduce this infinite sequence of 0's and 1's. In this context, no Turing machine can reproduce an unbounded number of the aforementioned active element firing patterns.

The following definition helps clarify the one-way nature of the random AEM firing patterns.

Definition 1. *Turing machine map back. Let* $g : \mathbb{N} \to \{0,1\}$ *and* $f : \mathbb{N} \to \{0,1,\dots,m\}$ *be functions. A Turing machine maps* g *back to function* f *if conditions* **A** *and* **B** *hold.* **A)** *The initial Turing machine tape has* $g(k)$ *stored on tape square* k *for each* k*.* **B)** *The Turing machine begins execution at tape square 1 and after a finite number of steps writes* $f(1)$ *on tape square 1. For each* k*, the Turing machine visits tape square* k *and after a finite number of steps, it writes* $f(k)$ *on tape square* k *and then moves to tape square* $k+1$*.*

If zero information is revealed to an adversary about the AEM's representation of the UTM's state and tape and the quantum random bits that help determine $\overline{\eta}$'s computation and zero information is revealed about the dynamic connections between the active elements, then there does not exist a "reverse engineer" Turing machine that can map the random firing patterns back to the sequence of UTM instructions when the UTM executes an unbounded number of computable steps. Overall, using quantum randomness, a finite active element machine can deterministically execute any Turing machine with active element firing patterns that are Turing incomputable. This casts a new light on the notion of a *computational procedure* ([5], [45]). In [29], Lewis and Papadimitriou discuss the Church-Turing notion of an algorithm (computational procedure):

> The principle that Turing machines are formal versions of algorithms and that no computational procedure will be considered as an algorithm unless it can be presented as a Turing machine is known as Church's thesis or the Church-Turing Thesis. It is a thesis, not a theorem, because it is not a mathematical result: It simply asserts that a certain informal concept corresponds to a certain mathematical object. It is theoretically possible, however, that Church's thesis could be overthrown at some future date, if someone were to propose an alternative model of computation that was publicly acceptable as fulfilling the requirement of *finite labor at each step* and yet was provably capable of carrying out computations that cannot be carried out by any Turing machine.

The works of [14], [22], [23], [27] and [40] describe methods of *hypercomputation* that currently have no physical realization. In [11] Davis provides counterarguments to the physical realizability of hypercomputation. In [10] he calls hypercomputation "a myth" and appears to dismiss random number generators as a means to enhance the computing power of Turing machines.

> The computing power of Turing machines provided with a random number generator was studied in the classic paper [12]. It turned out that such machines could compute only functions that are already computable by ordinary Turing machines.

In [12], de Leeuw et al. explain that their results depend on the definitions of the machines chosen and the tasks these machines perform.

> The following question will be considered in this paper: Is there anything that can be done by a machine with a random element but not by a deterministic machine?
>
> The question as it stands is, of course, too vague to be amenable to mathematical analysis. In what follows, it must be delimited in two respects. A precise definition of the class of machine to be considered must be given and an equally precise definition must be given of the tasks which they are to perform. It is clear that the nature of our results will depend strongly on these two choices and therefore our answer is not to be interpreted as a complete solution of the originally posed informal question.

What are relevant differences? In [12], the Turing machine reads a random infinite sequence of 0's and 1's off the tape – called a probabilistic machine – but the Turing machine program does not change as a consequence of this random input. (A definition of Turing machine *program* is in section 7.1 of the appendix.) In this paper, during AEM program execution, the `Meta` command changes the AEM architecture (program) based on the quantum random input, which changes the AEM program's computing behavior. In particular, at each computational step of the UTM, the random input helps determine the element parameters and connection parameters of the active elements that collectively compute $\overline{\eta}$; and the `Meta` command helps dynamically change these parameters based on the random input.

Another AEM example with random input also illustrates the differences. In [18], section 7 describes a method – using a finite AEM program, a quantum random input and the `Meta` command – for recognizing a non-Turing binary language $L \subset \{0,1\}^*$. After the AEM has received enough random input, then the AEM can determine whether, for example, if the string 011001 is in the binary language L recognized by the AEM. In other words, the random input helps determine the binary language L and the `Meta` command helps dynamically build the appropriate AEM that recognizes L.

A physical realization of the methods shown here can be implemented using a quantum random generator device [43] with a USB plug connected to a laptop computer executing a finite active element machine program. It is important to note that this physical realization of incomputable firing patterns is a scientific thesis – not a mathematical proof, as this realization depends on quantum theory.

In a cryptographic system, Claude Shannon [38] defines the notion of *perfect secrecy*.

> Perfect Secrecy is defined by requiring of a system that after a cryptogram is intercepted by the enemy the *a posteriori* probabilities of this cryptogram representing various messages be identically the same as the *a priori* probabilities of the same messages before the interception.

Perfect secrecy here means that zero information is released about the state and the tape contents of the universal Turing machine, the quantum random bits that help determine how $\overline{\eta}$ is computed and the dynamic connections of the active element machine. Formally, let $f_{1,j}\ f_{2,j}\ \ldots\ f_{m,j}$ represent the random firing pattern computing $\overline{\eta}$ during the jth computational step and assume an adversary can only eavesdrop on $f_{1,j}\ f_{2,j}\ \ldots\ f_{m,j}$. Let q denote the current state of the UTM, a_k a UTM alphabet symbol and q_k a UTM state. Perfect secrecy means that probabilities $P(q = q_k \mid f_{1,j} = b_1\ \ldots\ f_{m,j} = b_m) = P(q = q_k)$ and $P(T_k = a_k \mid f_{1,j} = b_1\ \ldots\ f_{m,j} = b_m) = P(T_k = a_k)$ for each $b_i \in \{0,1\}$ and each T_k which represents the contents of the kth tape square.

In [28], Kocher et al. present differential power analysis. Differential power analysis obtains information about cryptographic computations executed by register machine hardware, by statistically analyzing the electromagnetic radiation leaked by the hardware during its computation. If a quantum active element computing system is built so that its internal components remain close to perfectly secret, then it could be extremely challenging for an adversary to carry out cyberattacks such as differential power analysis.

1.1 Brief Summary of Prior Computing Models

In [45], the Turing Machine is introduced. A brief review is in the appendix. In [44], Sturgis and Shepherdson show the computational equivalence of the register machine. The works [9], [29], [30], [31], [33] and [41] cover computability theory. Alternative models influenced by neurophysiology are discussed by McCulloch and Pitts in [32], Rosenblatt in [37], Minsky and Papert in [34], Rall in [35], Hertz et al. in [21] and Hopfield in [24], [26] and [25]. In [20] Halang

et al. describe some advantages of using time. Important work on quantum computing models is presented in [1], [2], [13], [15], [16], [19], [30], [31] and [39].

2 An Informal Summary of the Active Element Machine

A formal introduction to the active element machine is in section 7.2. An AEM is composed of computational primitives called active elements. There are three kinds of active elements: Input, Computational and Output active elements. Input active elements process information received from the environment or another active element machine. Computational active elements receive pulses from the input active elements and other computational active elements and transmit new pulses to computational and output active elements. The output active elements receive pulses from the input and computational active elements. Every active element is active in the sense that each one can receive and transmit pulses simultaneously.

Each pulse has an amplitude and a width, representing how long the pulse amplitude lasts as input to the active element receiving the pulse. If active element E_i simultaneously receives pulses with amplitudes summing to a value greater than E_i's threshold and E_i's refractory period has expired, then E_i fires. When E_i fires, it sends pulses to other active elements. If E_i fires at time t, a pulse reaches element E_k at time $t + \tau_{ik}$ where τ_{ik} is the transmission time from element E_i to E_k.

The AEM programming language has 5 commands and 2 special keywords. (See 7.3.) Consider element command `(Element (Time 2) (Name L) (Threshold -3) (Refractory 2) (Last 0))`. At time 2, if active element `L` does not exist, then it is created. Element `L` has its threshold set to -3, its refractory period set to 2, and its last time fired set to 0. After time 2, element `L` exists indefinitely with threshold $= -3$, refractory $= 2$ until a new element command whose name value `L` is executed at a later time; in this case, the parameter values specified in the new command are updated.

A connection command sets the pulse parameters between two elements. Consider command `(Connection (Time 2) (From C) (To L) (Amp -7) (Width 6) (Delay 3))` . At time 2, the connection from active element `C` to active element `L` has its amplitude set to -7, its pulse width set to 6, and transmission time set to 3. A `Fire` command fires an input active element in order to communicate program input to the AEM. `(Fire (Time 3) (Name C))` causes element `C` to fire at time $t = 3$.

A `Program` command enables one command to execute multiple commands. The execution of `(Q (Args 0 E L))` based on the following program definition

```
(Program Q (Args t x y)
  (Element (Time t) (Name x) (Refractory 7) (Threshold 8) (Last 0))
  (Connection (Time t) (From x) (To y) (Amp 5) (Width 3) (Delay 4))
)
```

creates element `E` and a connection from `E` to `L` at time $t = 0$.

The `Meta` command `(Meta (Name E) (Window 1 5) (C (Args a b)) )` causes command `C` to execute with arguments `a` and `b` each time that element `E` fires during the time window $[1, 5]$. The keyword `dT` denotes an infinitesimal amount of time that helps coordinate almost simultaneous events. `dT` > 0 and `dT` is less than every positive rational number. The keyword `clock` evaluates to an integer, which is the current time of the AEM clock.

3 AEM Interpretations of Boolean Functions

In this section, the same boolean function is computed by two or more distinct active element firing patterns, which can be executed at two distinct times or by two different circuits in the AEM. The use of level sets helps design distinct AEM firing patterns that can compute the same boolean function. These firing patterns can be generated using quantum randomness.

The following procedure uses a finite active element program and a quantum system to either fire input element `I` or not fire `I` at time $t = n$ where n is a natural number $\{0, 1, 2, 3, \dots\}$. This random sequence of 0's and 1's can be generated by quantum systems discussed in [3], [42] or [43].

Procedure 1. *Randomness generates an AEM, representing a real number in* $[0, 1]$

A random sequence of bits creates active elements named $0, 1, 2, \dots$ that store the binary representation $b_0 b_1 b_2 \dots$ of real number $x \in [0, 1]$. If input element `I` fires at time $t = n$, then $b_n = 1$; thus, create active element `n` so that after $t = n$, element `n` fires every unit of time indefinitely. If input element `I` doesn't fire at time $t = n$, then $b_n = 0$ and active element `n` is created so that it never fires. The following program `real` exhibits this behavior.

```
(Program real  (Args t)
  (Connection (Time t) (From I) (To t) (Amp 2) (Width 1) (Delay 1))
  (Connection (Time t+1+dT) (From I) (To t) (Amp 0))
  (Connection (Time t) (From t) (To t) (Amp 2) (Width 1) (Delay 1))   )
(Element (Time clock) (Name clock) (Threshold 1) (Refractory 1) (Last -1))
(Meta (Name I) (real (Args clock)))
```

Suppose the sequence of random bits begins with 1, 0, 1, ... Thus, input element `I` fires at times 0, 2, ... At time 0, the following commands are executed.

```
(Element (Time 0) (Name 0) (Threshold 1) (Refractory 1) (Last -1))
(real (Args 0))
```

The execution of `(real (Args 0))` causes three connection commands to execute.

```
(Connection (Time 0) (From I) (To 0) (Amp 2) (Width 1) (Delay 1))
(Connection (Time 1+dT) (From I) (To 0) (Amp 0))
(Connection (Time 0) (From 0) (To 0) (Amp 2) (Width 1) (Delay 1))
```

At time 0, input element `I` sends a pulse with amplitude 2 to element `0`. Thus, element `0` fires at time 1. At time `1+dT`, a moment after time 1, the connection from input element `I` to element `0` is removed. At time 0, a connection from element `0` to itself with amplitude 2 is created. Element `0` continues to fire indefinitely, indicating that $b_0 = 1$.

`(Element (Time 1) (Name 1) (Threshold 1) (Refractory 1) (Last -1))` is created at time 1. Since element `1` has no connections into it and threshold 1, it never fires. Thus $b_1 = 0$.

3.1 Active Element Firing Patterns

This subsection explains how a firing pattern can be used to compute a boolean function $f : \{0, 1\}^2 \to \{0, 1\}$. These methods can be extended to $f : \{0, 1\}^n \to \{0, 1\}$. In the next section, the level set methods described here are combined with procedure 1 so that a quantum random firing pattern computes the boolean functions representing the execution of universal Turing machine program $\overline{\eta}$.

Active elements X_0, X_1, X_2 and X_3 are designed so that each X_i either fires or doesn't fire during window $\mathcal{W} = [a, b]$. After one of the 16 firing patterns is randomly generated, level sets $f^{-1}\{0\}$ and $f^{-1}\{1\}$ are used to compute $f : \{0, 1\}^2 \to \{0, 1\}$. Element P represents the output of f. The goal is for P to fire during window $\mathcal{W}$ if and only if P receives a unique firing pattern from elements X_0, X_1, X_2 and X_3. This goal motivates the following definition.

Definition 2. *Number of Firings during a Window*

Let X denote the set of active elements $\{X_0, X_1, \ldots, X_{n-1}\}$ that determine a firing pattern during window of time $\mathcal{W}$. Then $|(X_k, \mathcal{W})|$ = the number of times that element X_k fired during $\mathcal{W}$. Thus, define the number of firings during window $\mathcal{W}$ as $|(X, \mathcal{W})| = \sum_{k=0}^{n-1} |(X_k, \mathcal{W})|$.

Observe that $|(X, \mathcal{W})| = 0$ for firing pattern 0000 and $|(X, \mathcal{W})| = 2$ for firing pattern 0101. To isolate a firing pattern so that element P only fires if this unique firing pattern occurs, set the threshold of element $P = 2|(X, \mathcal{W})| - 1$. The element command for `P` is

`(Element (Time a-dT) (Name P) (Threshold 2| (X, W) | − 1 ) (Refractory b-a) (Last 2a-b))`

Each connection from X_k to P is based on whether X_k is supposed to fire or not fire during $\mathcal{W}$. If X_k is supposed to fire during $\mathcal{W}$, the following connection is established.

`(Connection (Time a-dT) (From X_k) (To P) (Amp 2) (Width b-a) (Delay 1))`

If X_k is not supposed to fire during $\mathcal{W}$, then the following connection is established. `(Connection (Time a-dT) (From X_k) (To P) (Amp -2) (Width b-a) (Delay 1))` The firing pattern is already known because it is determined based on a random source of bits received by input elements, as explained in procedure 1.

Example 1. *Computing $\oplus$ with Firing Pattern* 0010

Firing pattern 0010 *computes exclusive-or $A \oplus B = (A \vee B) \wedge (\neg A \vee \neg B)$.*

An AEM program is designed such that $A \oplus B = 1$ if and only if the firing pattern for X_0, X_1, X_2, X_3 is 0010. If $A \oplus B = 1$ then P fires. If $A \oplus B = 0$ then P doesn't fire. Choose window $\mathcal{W} = [2, 3]$. The following commands connect elements A and B to elements X_0, X_1, X_2, X_3.

```
(Connection (Time 0) (From A) (To X_0) (Amp 2) (Width 2) (Delay 2))
(Connection (Time 0) (From B) (To X_0) (Amp 2) (Width 2) (Delay 2))
(Element (Time 0) (Name X_0) (Threshold 3) (Refractory 1) (Last 1))

(Connection (Time 0) (From A) (To X_1) (Amp -2) (Width 2) (Delay 2))
(Connection (Time 0) (From B) (To X_1) (Amp -2) (Width 2) (Delay 2))
(Element (Time 0) (Name X_1) (Threshold -1) (Refractory 1) (Last 1))

(Connection (Time 0) (From A) (To X_2) (Amp 2) (Width 2) (Delay 2))
(Connection (Time 0) (From B) (To X_2) (Amp 2) (Width 2) (Delay 2))
(Element (Time 0) (Name X_2) (Threshold 1) (Refractory 1) (Last 1))

(Connection (Time 0) (From A) (To X_3) (Amp 2) (Width 2) (Delay 2))
(Connection (Time 0) (From B) (To X_3) (Amp 2) (Width 2) (Delay 2))
(Element (Time 0) (Name X_3) (Threshold 3) (Refractory 1) (Last 1))
```

There are four cases for $A \oplus B$ shown below.

1. $1 \oplus 0$. Element A fires at time $t = 0$ and element B doesn't fire at $t = 0$.
2. $0 \oplus 1$. Elements A doesn't fire at $t = 0$ and element B fires at time $t = 0$.
3. $1 \oplus 1$. Element A fires at time $t = 0$ and element B fires at $t = 0$.
4. $0 \oplus 0$. Elements A and B both don't fire at $t = 0$.

To isolate firing pattern 0010, set the threshold of `P` to $2|(X, \mathcal{W})| - 1 = 1$. The element command for `P` is `(Element (Time 2-dT) (Name P) (Threshold 1) (Refractory 1) (Last 1))`. To make `P` fire if and only if firing pattern 0010 occurs during $\mathcal{W}$, use the commands

```
(Connection (Time 2-dT) (From X_0) (To P) (Amp -2) (Width 1) (Delay 1))
(Connection (Time 2-dT) (From X_1) (To P) (Amp -2) (Width 1) (Delay 1))
(Connection (Time 2-dT) (From X_2) (To P) (Amp  2) (Width 1) (Delay 1))
(Connection (Time 2-dT) (From X_3) (To P) (Amp -2) (Width 1) (Delay 1))
```

For cases 1 and 2, ($1 \oplus 0$ and $0 \oplus 1$) only X_2 fires. A moment before X_2 fires at $t = 2$ (i.e., $-dT$), the amplitude from X_2 to P is set to 2. At time $t = 2$, a pulse with amplitude 2 is sent from X_2 to P, so P fires at time $t = 3$ since its threshold = 1. In other words, $1 \oplus 0 = 1$ or $0 \oplus 1 = 1$ has been computed. For case 3, $(1 \oplus 1)$, X_0, X_2 and X_3 fire. Thus, two pulses each with amplitude $= -2$ are sent from X_0 and X_3 to P. And one pulse with amplitude 2 is sent from X_2 to P. Thus, P doesn't fire. In other words, $1 \oplus 1 = 0$ has been computed. For case 4, $(0 \oplus 0)$, X_1 fires. One pulse with amplitude $= -2$ is sent to X_2. Thus, P doesn't fire. In other words, $0 \oplus 0 = 0$ has been computed.

As shown in table 1, any of the sixteen boolean functions can be mapped to one of the sixteen firing patterns by an appropriate AEM program using level sets to separate elements of the domain $\{(0,0),(1,0),(0,1),(1,1)\}$. Each active element X_k in the firing pattern separates these members based on the (amplitude from A to X_k, amplitude from B to X_k, threshold of X_k, element X_k) quadruplet. For example, the quadruplet $(0, 2, 1, X_1)$ separates $\{(1,1),(0,1)\}$ from $\{(1,0),(0,0)\}$ with respect to X_1. Recall that $A = 1$ means A fires and $B = 1$ means B fires. Then X_1 will fire with inputs $\{(1,1),(0,1)\}$ and X_1 will not fire with inputs $\{(1,0),(0,0)\}$. The separation rule is expressed as $(0, 2, 1, X_1) \leftrightarrow \frac{\{(1,1),(0,1)\}}{\{(1,0),(0,0)\}}$ The separation rule $(0, -2, -1, X_2) \leftrightarrow \frac{\{(1,0),(0,0)\}}{\{(1,1),(0,1)\}}$ indicates that X_2 has threshold -1 and amplitudes 0 and -2 from A and B respectively. Further, X_2 will fire with inputs $\{(1,0),(0,0)\}$ and will not fire with inputs $\{(1,1),(0,1)\}$.

Table 1: Functions $f_k : \{0,1\} \times \{0,1\} \to \{0,1\}$

Boolean Function	AEM Separation Rule(s)
$f_1(\mathbf{A},\mathbf{B}) = 1$	$(0, 0, -1, X_k) \leftrightarrow \frac{\{(0,0),(1,0),(0,1),(1,1)\}}{\emptyset}$
$f_2(\mathbf{A},\mathbf{B}) = 0$	$(0, 0, 1, X_k) \leftrightarrow \frac{\emptyset}{\{(0,0),(1,0),(0,1),(1,1)\}}$
$f_3(\mathbf{A},\mathbf{B}) = \mathbf{A}$	$(2, 0, 1, X_k) \leftrightarrow \frac{\{(1,1),(1,0)\}}{\{(0,1),(0,0)\}}$
$f_4(\mathbf{A},\mathbf{B}) = \mathbf{B}$	$(0, 2, 1, X_k) \leftrightarrow \frac{\{(1,1),(0,1)\}}{\{(1,0),(0,0)\}}$
$f_5(\mathbf{A},\mathbf{B}) = \neg\mathbf{A}$	$(-2, 0, -1, X_k) \leftrightarrow \frac{\{(0,1),(0,0)\}}{\{(1,1),(1,0)\}}$
$f_6(\mathbf{A},\mathbf{B}) = \neg\mathbf{B}$	$(0, -2, -1, X_k) \leftrightarrow \frac{\{(1,0),(0,0)\}}{\{(1,1),(0,1)\}}$
$f_7(\mathbf{A},\mathbf{B}) = \mathbf{A} \wedge \mathbf{B}$	$(2, 2, 3, X_k) \leftrightarrow \frac{\{(1,1)\}}{\{(1,0),(0,1),(0,0)\}}$
$f_8(\mathbf{A},\mathbf{B}) = \mathbf{A} \vee \mathbf{B}$	$(2, 2, 1, X_k) \leftrightarrow \frac{\{(1,0),(0,1),(1,1)\}}{\{(0,0)\}}$
$f_9(\mathbf{A},\mathbf{B}) = \mathbf{A} \to \mathbf{B}$	$(-4, 2, -3, X_k) \leftrightarrow \frac{\{(0,0),(0,1),(1,1)\}}{\{(1,0)\}}$
$f_{10}(\mathbf{A},\mathbf{B}) = \mathbf{A} \leftarrow \mathbf{B}$	$(2, -4, -3, X_k) \leftrightarrow \frac{\{(0,0),(1,0),(1,1)\}}{\{(0,1)\}}$
$f_{11}(\mathbf{A},\mathbf{B}) = \mathbf{A} \leftrightarrow \mathbf{B}$	$(2, -4, -3, X_k)$ and $(-4, 2, -3, X_j)$ with $j \neq k$

Boolean Function	AEM Separation Rule(s)
$f_{12}(\mathbf{A},\mathbf{B}) = \neg(\mathbf{A} \vee \mathbf{B})$	$(-2,-2,-1,X_k) \leftrightarrow \frac{\{(0,0)\}}{\{(1,0),(0,1),(1,1)\}}$
$f_{13}(\mathbf{A},\mathbf{B}) = \neg(\mathbf{A} \wedge \mathbf{B})$	$(-2,-2,-3,X_k) \leftrightarrow \frac{\{(1,0),(0,1),(0,0)\}}{\{(1,1)\}}$
$f_{14}(\mathbf{A},\mathbf{B}) = \mathbf{A} \oplus \mathbf{B}$	$(2,2,1,X_k)$ and $(-2,-2,-3,X_j)$ with $j \neq k$
$f_{15}(\mathbf{A},\mathbf{B}) = \mathbf{A} < \mathbf{B}$	$(-2,4,3,X_k) \leftrightarrow \frac{\{(0,1)\}}{\{(0,0),(1,0),(1,1)\}}$
$f_{16}(\mathbf{A},\mathbf{B}) = \mathbf{A} > \mathbf{B}$	$(4,-2,3,X_k) \leftrightarrow \frac{\{(1,0)\}}{\{(0,0),(0,1),(1,1)\}}$

For each X_j, use one of the separation rules to map the level set ${f_k}^{-1}\{1\}$ or alternatively map the level set ${f_k}^{-1}\{0\}$ to one of the sixteen firing patterns represented by X_0, X_1, X_2, X_3.

4 Random Firing Interpretations Execute a UTM

A universal Turing Machine (UTM) is a Turing machine that can mimic the computation of any Turing Machine by reading the other Turing Machine's description and input from the UTM's tape. Table 5 in the appendix shows Minsky's universal Turing machine described in [33]. A boolean representation of Minsky's UTM is shown in table 2.

The elements of $\{0,1\}^2$ are denoted as $\mathcal{A} = \{00, 01, 10, 11\}$. The tape symbols in Minsky's UTM alphabet correspond to elements in $\mathcal{A}$ as follows: $0 \leftrightarrow 00$, $1 \leftrightarrow 01$, $y \leftrightarrow 10$ and $A \leftrightarrow 11$. The states of Minsky's UTM correspond to the elements of Q as follows: $q_1 \leftrightarrow 001$, $q_2 \leftrightarrow 010$, $q_3 \leftrightarrow 011$, $q_4 \leftrightarrow 100$, $q_5 \leftrightarrow 101$, $q_6 \leftrightarrow 110$, $q_7 \leftrightarrow 111$ and the halting state $\mathcal{H} \leftrightarrow 000$. In regard to tape head moves, $L \leftrightarrow 0$ and $R \leftrightarrow 1$ in $\{0,1\}$. Symbol h indicates that the tape head does not move, which occurs when the UTM has halted.

An active element machine is designed to compute the universal Turing Machine program $\overline{\eta}$ shown in table 2. Following the methods in 3.1, multiple AEM firing interpretations are created that compute $\overline{\eta}$. The three boolean variables U, W, X are concatenated to represent the current state of the UTM. The two boolean variables Y, Z represent the current tape symbol. Observe that $\overline{\eta} = (\overline{\eta}_0\overline{\eta}_1\overline{\eta}_2,\ \overline{\eta}_3\overline{\eta}_4,\ \overline{\eta}_5)$. The level sets of $\overline{\eta}_3$ are shown below.

${\overline{\eta}_3}^{-1}(UWX, YZ)\{1\} = \{(111,00), (110,00), (110,01), (110,10), (101,00), (101,01), (101,10), (100,00), (100,10)$
$(011,01), (011,10), (010,11), (010,01), (010,00)\}$

${\overline{\eta}_3}^{-1}(UWX, YZ)\{0\} = \{(111,01), (111,10), (111,11), (110,11), (101,11), (100,01), (100,11), (011,00), (011,11)$
$(010,10), (001,00), (001,01), (001,10), (001,11), (000,01), (000,10), (000,11), (000,00)\}$

Table 2: Boolean Universal Turing Machine program $\overline{\eta} = (\overline{\eta}_0\overline{\eta}_1\overline{\eta}_2,\ \overline{\eta}_3\overline{\eta}_4,\ \overline{\eta}_5)$

	10	00	01	11
001	(001, 00, 0)	(001, 00, 0)	(010, 01, 0)	(001, 01, 0)
010	(001, 00, 0)	(010, 10, 1)	(010, 11, 1)	(110, 10, 1)
011	(011, 10, 0)	(000, 00, h)	(011, 11, 0)	(100, 01, 0)
100	(100, 10, 0)	(101, 10, 1)	(111, 01, 0)	(100, 01, 0)
101	(101, 10, 1)	(011, 10, 0)	(101, 11, 1)	(101, 01, 1)
110	(110, 10, 1)	(011, 11, 0)	(110, 11, 1)	(110, 01, 1)
111	(111, 00, 1)	(110, 10, 1)	(111, 01, 1)	(010, 00, 1)

State set $Q = \{001, 010, 011, 100, 101, 110, 111\}$. Alphabet $\mathcal{A} = \{10, 00, 01, 11\}$.

The level sets of $\overline{\eta}_k : \{0,1\}^3 \times \{0,1\}^2 \to \{0,1\}$ where $k \in \{0,1,2,4,5\}$ are shown below.

$\overline{\eta}_0{}^{-1}(UWX,YZ)\{1\} = \{(111,10),(111,01),(111,00),(110,11),(110,10),(110,01),(101,11),(101,10),(101,01)$
$(100,11),(100,10),(100,01),(100,00),(011,11)\ (010,11)\}$

$\overline{\eta}_0{}^{-1}(UWX,YZ)\{0\} = \{(111,11),(110,00),(101,00),(011,10),(011,01),(011,00),(010,10),(010,01),(010,00)$
$(001,11),(001,10),(001,01),(001,00),(000,11),(000,10),(000,01),(000,00)\}$

$\overline{\eta}_1{}^{-1}(UWX,YZ)\{1\} = \{(111,11),(111,10),(111,01),(111,00),(110,11),(110,10),(110,01),(110,00),(101,00)$
$(100,01),(011,10),(011,01),(010,11),(010,01),(010,00),(001,01)\}$

$\overline{\eta}_1{}^{-1}(UWX,YZ)\{0\} = \{(101,11),(101,10),(101,01),(100,11),(100,10),(100,00),(011,11),(011,00),(010,10)$
$(001,11),(001,10),(001,00),(000,11),(000,10),(000,01),(000,00)\}$

$\overline{\eta}_2{}^{-1}(UWX,YZ)\{1\} = \{(111,10),(111,01),(110,00),(101,11),(101,10),(101,01),(101,00),(100,01),(100,00)$
$(011,10),(011,01),(010,10),(001,11),(001,10),(001,00)\}$

$\overline{\eta}_2{}^{-1}(UWX,YZ)\{0\} = \{(111,11),(111,00),(110,11),(110,10),(110,01),(100,11),(100,10),(011,11),(011,00)$
$(010,11),(010,01),(010,00),(001,01),(000,11),(000,10),(000,01),(000,00)\}$

$\overline{\eta}_4{}^{-1}(UWX,YZ)\{1\} = \{(111,01),(110,11),(110,01),(110,00),(101,11),(101,01),(100,11),(100,01),(011,11)$
$(011,01),(010,01),(001,11),(001,01)\}$

$\overline{\eta}_4{}^{-1}(UWX,YZ)\{0\} = \{(111,11),(111,10),(111,00),(110,10),(101,10),(101,00),(100,10),(100,00),(011,10)$
$(011,00),(010,11),(010,10),(010,00),(001,10),(001,00),(000,11),(000,10),(000,01),(000,00)\}$

$\overline{\eta}_5{}^{-1}(UWX,YZ)\{1\} = \{(111,11),(111,10),(111,01),(111,00),(110,11),(110,10),(110,01),(101,11),(101,10)$
$(101,01),(100,00),(010,11),(010,01),(010,00)\}$

$\overline{\eta}_5{}^{-1}(UWX,YZ)\{0\} = \{(110,00),(101,00),(100,11),(100,10),(100,01),(011,11),(011,10),(011,01),(010,10)$
$(001,11),(001,10),(001,01),(001,00),(000,11),(000,10),(000,01),(000,00)\}$

The level set $\overline{\eta}_5{}^{-1}(UWX,YZ)\{h\} = \{(011,00)\}$ is the special case when the UTM halts (i.e., $\overline{\eta}(011,00) = (000,00,h)$). When the UTM halts, the AEM reaches a halting firing pattern $\mathcal{H}$.

The next example copies one element's firing state to another element's firing state. This program helps assign the value of a random bit to an active element and perform other functions in the UTM. When the following `copy` program is called, active element `b` fires if `a` fired during the window of time [`s`, `t`). Further, a connection is set up from `b` to `b` so that `b` will keep firing indefinitely. This enables `b` to *store* active element `a`'s firing state.

Example 2. *Copy Program*
An AEM program copies active element `a`'s firing state to element `b`.

```
(Program copy (Args s t b a)
  (Element (Time s-1) (Name b) (Threshold 1) (Refractory 1) (Last s-1))
  (Connection (Time s-1) (From a) (To b) (Amp 0) (Width 0) (Delay 1))
  (Connection (Time s)  (From a) (To b) (Amp 2) (Width 1) (Delay 1))
  (Connection (Time s)  (From b) (To b) (Amp 2) (Width 1) (Delay 1))
  (Connection (Time t)  (From a) (To b) (Amp 0) (Width 0) (Delay 1))   )
```

Procedure 2. *Computing Turing Program $\overline{\eta}$ with Random Firing Patterns*
Procedure 2 describes the computation of $\overline{\eta}$ with random AEM firing patterns.

Consider function $\overline{\eta}_3 : \{0,1\}^5 \to \{0,1\}$. The following scheme is used to represent boolean values 1 and 0 with the firing of active elements. If active element U fires during window $\mathcal{W}$, then this corresponds to input $U = 1$ in $\overline{\eta}_3$; if active element U doesn't fire during window $\mathcal{W}$, then this corresponds to input $U = 0$ in $\overline{\eta}_3$. When U fires, W doesn't fire, X fires, Y doesn't fire and Z doesn't fire, this corresponds to computing $\overline{\eta}_3(101,00)$. The value $1 = \overline{\eta}_3(101,00)$ is the underlined bit in $(011,\underline{1}0,0)$, which is located in row 101, column 00 of table 2.

Procedure 1 and the separation rules in table 3 are synthesized so that $\overline{\eta}_3$ is computed using a dynamic interpretation determined by quantum random bits. This creates random active element firing patterns that compute $\overline{\eta}_3$.

The firing activity of element P_3 represents the output value of $\overline{\eta}_3(UWX, YZ)$. Fourteen random bits are created from quantum randomness (See [42], [43]). These random bits create a corresponding random firing pattern of active elements $R_0, R_1, \ldots R_{13}$. Meta commands dynamically build active elements and connections based on the separation rules in table 3 and the firing activity of elements $R_0, R_1, \ldots R_{13}$. These dynamically created elements and connections determine the firing activity of element P_3 based on the firing activity of elements U, W, X, Y and Z. This procedure is described below.

Table 3: AEM Separation Rules for Level Set ${\overline{\eta}_3}^{-1}\{1\}$

Firing Pattern	Element	(U, D_i)	(W, D_i)	(X, D_i)	(Y, D_i)	(Z, D_i)	θ_{D_i}
111 00	D_0	2	2	2	−2	−2	5
110 00	D_1	2	2	−2	−2	−2	3
110 01	D_2	2	2	−2	−2	2	5
110 10	D_3	2	2	−2	2	−2	5
101 00	D_4	2	−2	2	−2	−2	3
101 01	D_5	2	−2	2	−2	2	5
101 10	D_6	2	−2	2	2	−2	5
100 00	D_7	2	−2	−2	−2	−2	1
100 10	D_8	2	−2	−2	2	−2	3
011 01	D_9	−2	2	2	−2	2	5
011 10	D_{10}	−2	2	2	2	−2	5
010 00	D_{11}	−2	2	−2	−2	−2	1
010 01	D_{12}	−2	2	−2	−2	2	3
010 11	D_{13}	−2	2	−2	2	2	5

Step 2.1 A quantum source creates fourteen random bits a_0, a_1, ... and a_{13}. These bit values are stored in elements R_0, $R_1, \ldots, R_{13}$. If $a_k = 1$, then R_k fires; if $a_k = 0$, R_k doesn't fire.

Step 2.2 Set up dynamical connections from active elements U, X, W, Y, Z to elements $D_0, D_1, \ldots, D_{13}$, which depend on meta commands that use the firing pattern from elements $R_0, R_1, \ldots R_{13}$.

For D_0, look at the first row and first column of table 3. The firing pattern 111 00 means that D_0 dynamically separates this pattern with respect to the other firing patterns of UWX YZ. The separation is dynamic based on whether element R_0 fires or doesn't fire. If R_0 fires, then D_0 fires when the firing pattern for UWX YZ is 111 00; for all other firing patterns for UWX YZ, then D_0 doesn't fire. If R_0 doesn't fire, then D_0 doesn't fire when the firing pattern for UWX YZ is 111 00; for all other firing patterns for UWX YZ, then D_0 fires. In a similar way, D_1 dynamically separates the firing pattern 110 00 for UWX YZ based on whether element R_1 fires. Observe that every firing pattern in the first column is in the level set of ${\overline{\eta}_3}^{-1}\{1\}$.

The firing pattern 111 00 corresponds to the $\overline{\eta}$ instruction $(111, 00)$, whose output $(110, 10, 1)$ is shown in the last row and second column of table 2. The amplitudes from U, W, X, Y, Z to D_0 are labeled by the headers (U, D_i), (W, D_i), (X, D_i), (Y, D_i) and (Z, D_i), respectively. D_0's threshold 5 is in the first row under the header θ_{D_i}.

```
(Program  set_dynamic_C  (Args  s  t  f  xk  a  w  tau  rk)
 (Connection (Time s-dT) (From f) (To xk) (Amp -a) (Width w) (Delay tau))
 (Meta (Name rk) (Window s t)
       (Connection (Time t) (From f) (To xk) (Amp a) (Width w) (Delay tau)))   )
```

```
(Program  set_dynamic_E  (Args  s  t  xk  h  r  L  rk)
 (Element (Time s-2dT) (Name xk) (Threshold -h) (Refractory r) (Last L))
 (Meta (Name rk) (Window s t)
       (Element (Time t) (Name xk) (Threshold h) (Refractory r) (Last L)))   )

(set_dynamic_E (Args s t D0  5 1 s-2 R0))    (set_dynamic_C (Args s t U D0  2 1 1 R0))
(set_dynamic_C (Args s t W D0  2 1 1 R0))    (set_dynamic_C (Args s t X D0  2 1 1 R0))
(set_dynamic_C (Args s t Y D0 -2 1 1 R0))    (set_dynamic_C (Args s t Z D0 -2 1 1 R0))
```

At time `s-dT`, `set_dynamic_C` initializes the amplitudes of the connections to $A_{UD_0} = -2$, $A_{WD_0} = -2$, $A_{XD_0} = -2$, $A_{YD_0} = 2$, $A_{ZD_0} = 2$. If R_0 fires, then the meta command in `set_dynamic_C`, causes the connection command to execute at time `t`, which flips the sign of each of the amplitudes to $A_{UD_0} = 2$, $A_{WD_0} = 2$, $A_{XD_0} = 2$, $A_{YD_0} = -2$, $A_{ZD_0} = -2$. Similarly, `set_dynamic_E` initializes the threshold of D_0 to $\theta_{D_0} = -5$. If R_0 fires, then the meta command causes the element command to execute at time `t`, which flips the sign of the threshold of D_0. In this case, $\theta_{D_0} = 5$.

Similarly, for elements $D_1, \ldots, D_{13}$, `set_dynamic_E` and `set_dynamic_C` dynamically set the element parameters and the connections from U, X, W, Y, Z to $D_1, \ldots, D_{13}$ based on the rest of the quantum random firing pattern $R_1, \ldots R_{13}$ and the appropriate parameter values shown in table 3.

Step 2.3 Set up connections to active elements $G_0, G_1, G_2, \ldots G_{14}$ which represent the number of elements in $\{R_0, R_1, R_2, \ldots R_{13}\}$ that are firing. If 0 are firing, then only G_0 is firing. Otherwise, if $k > 0$ elements in $\{R_0, R_1, R_2, \ldots R_{13}\}$ are firing, then only $G_1, G_2, G_3 \ldots G_k$ are firing.

```
(Program firing_count (Args G a b h)
  (Element (Time a-2dT) (Name G) (Threshold h) (Refractory b-a) (Last 2a-b))
  (Connection (Time a-dT) (From R0) (To G) (Amp 2) (Width b-a) (Delay 1))
  (Connection (Time a-dT) (From R1) (To G) (Amp 2) (Width b-a) (Delay 1))
   . . .
  (Connection (Time a-dT) (From R13) (To G) (Amp 2) (Width b-a) (Delay 1))
 )

(firing_count (Args G0 a b -1))   (firing_count (Args G1 a b 1))  . . .
(firing_count (Args G13 a b 25))  (firing_count (Args G14 a b 27))
```

Step 2.4 Element P_3 represents the output of $\overline{\eta}_3$. Initialize P_3's threshold based on meta commands that use the firing activity from elements $G_0, G_1, G_2, \ldots G_{13}$. Since `t + dT` $<$ `t + 2dT` $< \cdots <$ `t + 15dT`, the meta commands set the threshold for P_3 to $-2(14 - k) + 1$ where k is the number of firings. For example, if nine of the randomly chosen bits are high, then G_9 will fire, so the threshold of P_3 is set to -9. If five of the random bits are high, then the threshold of P_3 is set to -17. Each element of the level set ${\overline{\eta}_3}^{-1}\{0\}$ creates a firing pattern of $D_0, \ldots D_{13}$ equal to the complement of the random firing pattern $R_0 R_1 \ldots R_{13}$ (i.e., D_k fires if and only if R_k does not fire).

```
(Program  set_P_threshold (Args  G  P  s  t  a b theta  kdT)
 (Meta (Name G) (Window s t)
      (Element (Time t+kdT) (Name P) (Threshold theta) (Refractory b-a) (Last t-b+a))))

(set_P_threshold  (Args G0 P3 s t a b -27  dT))
(set_P_threshold  (Args G1 P3 s t a b -25 2dT))
```

```
 . . .
(set_P_threshold (Args G13 P3 s t a b -1 14dT))
(set_P_threshold (Args G14 P3 s t a b  1 15dT))
```

Step 2.5 Set up dynamical connections from $D_0, D_1, D_2, \ldots, D_{13}$ to P_3 based on the random bits stored by $R_0, R_1, \ldots R_{13}$. These connections are based on meta commands that use the firing pattern from elements $R_0, R_1, \ldots R_{13}$.

```
(Program  set_from_Xk_to_Pj  (Args s t Xk  Pj a w tau Rk)
 (Connection (Time s-dT) (From Xk) (To Pj) (Amp -a) (Width w) (Delay tau))
 (Meta (Name Rk) (Window s t)
       (Connection (Time t) (From Xk) (To Pj) (Amp a) (Width w) (Delay tau)))  )

(set_from_Xk_to_Pj  (Args s t  D0   P3 2 b-a 1 R0))     . . .
(set_from_Xk_to_Pj  (Args s t  D13  P3 2 b-a 1 R13))
```

Similar procedures use random firing patterns on elements $\{A_0, \ldots, A_{14}\}$, $\{B_0, \ldots, B_{15}\}$, $\{C_0, \ldots, C_{14}\}$, $\{E_0, \ldots, E_{12}\}$, and $\{F_0, \ldots, F_{13}\}$ to compute $\overline{\eta}_0$, $\overline{\eta}_1$, $\overline{\eta}_2$, $\overline{\eta}_4$, and $\overline{\eta}_5$, respectively. The outputs of $\overline{\eta}_0$, $\overline{\eta}_1$, $\overline{\eta}_2$, $\overline{\eta}_4$, and $\overline{\eta}_5$ are represented by active elements P_0, P_1, P_2, P_4 and P_5, respectively. In subsection 7.5 of the appendix, the level set rules for $\overline{\eta}_0$, $\overline{\eta}_1$, $\overline{\eta}_2$, $\overline{\eta}_4$, and $\overline{\eta}_5$ are shown, respectively in tables 6, 7, 8, 9 and 10.

The firing activity of element P_k represents a single bit that helps determine the next state or next tape symbol during a UTM computational step. If an eavesdropper is able to listen to the firing activity of P_0, P_1, P_2, P_3, P_4 and P_5, which collectively represent the computation of $\overline{\eta}(UXW, YZ)$, then this leaking of information could be used to reconstruct some or all of the UTM tape contents. This weakness can be rectified as follows. For each UTM computational step, the AEM uses six additional quantum random bits $b_0, b_1, b_2, b_3, b_4, b_5$. For P_3, if random bit $b_3 = 1$, then the dynamical connections from $D_0, D_1, \ldots D_{13}$ to P_3 are chosen as described above. However, if random bit $b_3 = 0$, then the amplitudes of the dynamical connections from $D_0, D_1, \ldots D_{13}$ to P_3 and the threshold of P_3 are multiplied by -1. This causes P_3 to fire when $\overline{\eta}_3(UXW, YZ) = 0$ and P_3 doesn't fire when $\overline{\eta}_3(UXW, YZ) = 1$.

This cloaking of P_3's firing activity can be coordinated with a meta command based on the value of b_3 so that P_3's firing is appropriately interpreted to dynamically change the active elements and connections that update the UTM tape contents and state after each computational step. This cloaking procedure can also be used for element P_k and random bit b_k where $k \in \{0, 1, 2, 4, 5\}$. Furthermore, the same methods can be used to cloak the active element firing patterns that represent the UTM's state and tape contents. In particular, these methods may help cloak the firing pattern representing the halting state $\mathcal{H}$.

Besides representing and computing the program $\overline{\eta}$ with quantum random firing patterns, there are other useful functions computed by active elements executing the UTM. Assume that these connections and the active element firing activity are kept perfectly secret as they represent the state and the tape contents of the UTM tape contents. Alternatively, the active elements representing the UTM tape contents and state may be cloaked similar to the description for cloaking elements $P_0, P_1, \ldots, P_5$.

- Three active elements $(q\ 0)$, $(q\ 1)$ and $(q\ 2)$ store the current state of the UTM.
- There are a collection of elements to represent the tape head location k where k is an integer.

- A marker active element $\mathcal{L}$ locates the leftmost tape square and a separate marker active element $\mathcal{R}$ locates the rightmost tape square. Any tape symbols outside these markers are assumed to be blank (i.e., 0). If the tape head moves beyond the leftmost tape square, then $\mathcal{L}$'s connection is removed and updated one tape square to the left and the machine is reading a 0. If the tape head moves beyond the rightmost tape square, then $\mathcal{R}$'s connection is removed and updated one tape square to the right and the machine is reading a 0.

- There are a collection of elements that represent the tape contents of the UTM. For each tape square k inside the marker elements, there are two elements named $(S\ k)$ and $(T\ k)$ whose firing pattern determines the alphabet symbol at tape square k. For example, if elements $(S\ 5)$ and $(T\ 5)$ are not firing, then tape square 5 contains alphabet symbol 0. If element $(S\ -7)$ is firing and element $(T\ -7)$ is not firing, then tape square -7 contains alphabet symbol y. If element $(S\ -4)$ is not firing and element $(T\ -4)$ is firing, then tape square -4 contains alphabet symbol 1. If elements $(S\ 13)$ and $(T\ 13)$ are both firing, then tape square 13 contains alphabet symbol A.

- Representing alphabet symbol 0 with two active elements that are not firing is convenient because if the tape head moves beyond the initial tape contents of the UTM, then the meta command can add two elements that are not firing to represent the contents of the new square.

The copy program helps construct useful functionality in the UTM. The following program helps copy a new alphabet symbol to the tape.

```
(Program copy_symbol (Args s t b0 a0 b1 a1)
  (copy (Args s t b0 a0))  (copy (Args s t b1 a1)) )
```

The following program enables a new state to be copied.

```
(Program copy_state (Args s t b0 a0 b1 a1 b2 a2)
  (copy (Args s t b0 a0))   (copy (Args s t b1 a1))   (copy (Args s t b2 a2))   )
```

The sequence of steps by which the UTM is executed with an AEM are described below.

1. Tape contents are initialized and the marker elements $\mathcal{L}$ and $\mathcal{R}$ are initialized.

2. The tape head is initialized to tape square $k = 0$ and the current machine state is initialized to q_2. In other words, $(q\ 0)$ is not firing $(q\ 1)$ is firing and $(q\ 2)$ is not firing.

3. $(S\ k)$ and $(T\ k)$ are copied to a_{in} and the current state $(q\ 0)$, $(q\ 1)$, $(q\ 2)$ is copied to q_{in}. and m represents the tape head move.

4. If $q_{out} = \mathcal{H}$, then the UTM halts: the AEM reaches a firing pattern that stores the current tape contents indefinitely and keeps the tape head fixed at tape square k where the UTM halted.

5. Otherwise, the firing pattern of the three elements representing q_{out} are copied to $(q\ 0)$, $(q\ 1)$, $(q\ 2)$. a_{out} is copied to the current tape square represented by $(S\ k)$, $(T\ k)$.

6. If $m = L$, then first determine if the tape head has moved to the left of the tape square marked by $\mathcal{L}$. If so, then have $\mathcal{L}$ remove its current marker and mark tape square $k - 1$. In either case, go back to step 3 where $(S\ k-1)$ and $(T\ k-1)$ are copied to a_{in}.

7. If $m = R$, then first determine if the tape head has moved to the right of the tape square marked by $\mathcal{R}$. If so, then have $\mathcal{R}$ remove its current marker and mark tape square $k+1$. In either case, go back to step 3 where $(S\ k+1)$ and $(T\ k+1)$ are copied to a_{in}.

First, a simple lemma is proved. This lemma will help show that for a non-halting UTM computation the firing activity of the elements computing $\overline{\eta}$ (i.e.,$\{A_0, \ldots, A_{14}, P_0\}$, $\{B_0, \ldots, B_{15}, P_1\}$, $\{C_0, \ldots, C_{14}, P_2\}$, $\{D_0, \ldots, D_{13}, P_3\}$, $\{E_0, \ldots, E_{12}, P_4\}$ and $\{F_0, \ldots, F_{13}, P_5\}$) is incomputable.

Lemma 4.1. *Let $\gamma : \mathbb{N} \to \{0,1\}$ be an incomputable function. For each k such that $1 \leq k \leq m$ assume the boolean function $B_k : \{0,1\}^n \to \{0,1\}^n$ is invertible. Let $h : \mathbb{N} \to \{1, \ldots, m\}$ be a computable function. Define function $g : \mathbb{N} \to \{0,1\}$ where $(g(1), g(2), \ldots, g(n)) = B_{h(1)}(\gamma(1), \gamma(2), \ldots, \gamma(n))$ and where $(g(n+1),\ g(n+2), \ldots, g(2n)) = B_{h(2)}(\gamma(n+1), \gamma(n+2), \ldots, \gamma(2n))$, and for the jth n-tuple in $\{0,1\}^n$ $(g(jn+1), g(jn+2), \ldots, g((j+1)n)) = B_{h(j)}(\gamma(jn+1), \gamma(jn+2), \ldots, \gamma((j+1)n))$, ...*

Then g is an incomputable function.

Proof. By reductio ad absurdum, suppose g is a computable function. Then $g(1), g(2), \ldots, g(m)$ are computable. Since $h(1)$ is computable and each B_k is invertible and boolean, $B^{-1}_{h(1)}$ is computable. Then $B^{-1}_{h(1)}(g(1), g(2), \ldots, g(n))$ is computable which equals $(\gamma(1), \ldots, \gamma(n))$. This completes the base case that γ is computable. Repeating this computation $j-1$ times, by induction, $\gamma(1), \ldots, \gamma((j-1)n)$ are computable. For the inductive step, $g(jn+1), g(jn+2)$, $\ldots$, $g((j+1)n)$ is computable. Since $h(j)$ is computable, then $B^{-1}_{h(j)}$ is computable. Thus, $B^{-1}_{h(j)}(g(jn+1),\ g(jn+2),\ \ldots, g((j+1)n))$ is computable which equals $(\gamma(jn+1), \gamma(jn+2)$, $\ldots, \gamma((j+1)n))$. The induction principle implies that $\gamma(i)$ is computable for every natural number i which is a contradiction. □

Definition 3. *Representing the firing activity of the active elements computing $\overline{\eta}$*

Consider the jth computational step of the UTM. For k such that $1 \leq k \leq 15$, let $f_{k,j} = 1$ if element A_{k-1} fires during this jth step and $f_{k,j} = 0$, otherwise. Let $f_{16,j}$ represent the firing activity of P_0. For k such that $17 \leq k \leq 32$, let $f_{k,j} = 1$ if element B_{k-17} fires during this jth step and $f_{k,j} = 0$, otherwise. Let $f_{33,j}$ represent the firing activity of P_1. For k such that $34 \leq k \leq 48$, let $f_{k,j} = 1$ if element C_{k-34} fires during this jth step and $f_{k,j} = 0$, otherwise. Let $f_{49,j}$ represent the firing activity of P_2. For k such that $50 \leq k \leq 63$, let $f_{k,j} = 1$ if element D_{k-50} fires during this jth step and $f_{k,j} = 0$, otherwise. Let $f_{64,j}$ represent the firing activity of P_3. For k such that $65 \leq k \leq 77$, let $f_{k,j} = 1$ if element E_{k-65} fires during this jth step and $f_{k,j} = 0$, otherwise. Let $f_{78,j}$ represent the firing activity of P_4. For k such that $79 \leq k \leq 92$, let $f_{k,j} = 1$ if element F_{k-79} fires during this jth step and $f_{k,j} = 0$, otherwise. Let $f_{93,j}$ represent the firing activity of P_5.

Definition 4. *Unbounded number of computable UTM steps*

If the UTM execution does not halt, then $f_{k,j}$ is defined for every natural number j. If the UTM execution halts on a finite or infinite number of tape inputs, then the definition of $f_{1,j}, \ldots, f_{93,j}$ can be extended as follows. Suppose the first tape input halts after m_1 steps. Before the second UTM execution starts, it is assumed that the UTM's tape contents are initialized using a computable method. For the first computational step, $j = m_1 + 1$. Inductively, if the kth UTM execution halts after m_k steps, then it is assumed that the UTM's tape contents

are initialized using a computable method. On the first step of the $k+1$th UTM execution $j = m_1 + m_2 + \ldots m_k + 1$.

For an unbounded number of computable UTM steps, define the function $\mathfrak{g} : \mathbb{N} \to \{0,1\}$ where $\mathfrak{g}(93(j-1)+r+1) = f_{(r+1),j}$ and $0 \leq r < 93$. In other words, $\mathfrak{g}(1) = \mathfrak{g}(93(1-1)+1) = f_{1,1}$ $\mathfrak{g}(2) = f_{2,1} \ldots \quad \mathfrak{g}(93) = f_{93,1} \quad \mathfrak{g}(94) = f_{1,2}$ and so on.

Theorem 4.2. *If an unbounded number of computable UTM steps are executed by the AEM according to Procedure 2 and the two quantum randomness axioms hold, then* $\mathfrak{g}$ *is an incomputable function.*

Proof. It suffices to show that the sequence $(f_{1,1}, f_{2,1}, \ldots f_{93,1},\ f_{1,2}, f_{2,2}, \ldots, f_{93,2}, \ldots, f_{1,j}, \ldots f_{93,j}, \ldots)$ in Ω is Turing incomputable. The proof covers the computation of $\overline{\eta}_3$. Similar steps, described below, can be verified for $\overline{\eta}_0, \overline{\eta}_1, \overline{\eta}_2, \overline{\eta}_4$ and $\overline{\eta}_5$. Lemma 4.1 is applied and the proof is completed.

The function γ in 4.1 is defined based on the firing activity of $R_0, R_1, \ldots, R_{13}$ discussed in steps 2.1 and 2.2 and quantum random bit b_3. From [4], the two quantum randomness axioms imply that γ is Turing incomputable. (Alternatively, it can be shown that the two quantum randomness axioms induce a Lebesgue probability measure on Ω. The computable sequences in Ω have Lebesgue measure zero because they are countable. Thus, any infinite quantum random sequence is incomputable with probability = 1).

From definition 3, $\mathfrak{g}$ represents the firing activity of elements $A_0, \ldots, A_{14}$, $B_0, \ldots, B_{15}$, $C_0, \ldots, C_{14}$, $D_0, \ldots, D_{13}$, $E_0, \ldots, E_{12}$, $F_0, \ldots, F_{13}$ and of output elements $P_0, P_1, P_2, P_3, P_4, P_5$. $\mathfrak{g}$ corresponds to function g in lemma 4.1. The elements $D_0, \ldots, D_{13}$ are discussed in steps 2.1 and 2.2. For each computational step of the UTM, table 4 shows how the AEM maps the firing activity of $R_0, R_1, \ldots, R_{13}, b_3$ to the firing activity of $D_0, D_1, \ldots, D_{13}, P_3$. This is verified by checking the program definitions of `set_dynamic_C` and `set_dynamic_E` in step 2.2. For example, if the current UTM instruction is $(111, 00)$ – i.e., in state 111 and reading tape symbol 00 – then this corresponds to boolean function $B_{(111,00)}$ shown in the first row of table 4. $B_{(111,00)}$ defines the mapping of $R_0, R_1, \ldots, R_{13}, b_3$ to the firing activity of $D_0, D_1, \ldots, D_{13}, P_3$. Each boolean function B_k in table 4 is invertible because $B_k \circ B_k$ equals the identity.

The final verification is that $h : \mathbb{N} \to \{1, 2, \ldots, 15\}$ is computable. First, identify 1 with instruction $(111, 00)$, 2 with instruction $(110, 00)$, ..., 14 with instruction $(010, 11)$ and 15 with ${\overline{\eta}_3}^{-1}\{0\}$. Let (q, T_k) be the current state and tape symbol of the UTM at the jth computational step. Define $h(j) = (q, T_k)$ if $\overline{\eta}_3(q, T_k) = 1$. Otherwise, define $h(j) = {\overline{\eta}_3}^{-1}\{0\}$. Since h can be computed based on the computation of the UTM, h is computable. In the other case, where the UTM execution halts one or more times, definition 4 assumes that the initialization of the UTM tape contents is computable, so h is computable. □

Theorem 4.3. *If an adversary can only eavesdrop on the firing activity of* $(f_{1,1}, f_{2,1}, \ldots f_{93,1}, \ldots, f_{1,j}, \ldots f_{93,j}, \ldots)$ *then the AEM execution, described in Procedure 2, of the UTM is perfectly secret. In other words,* $P(q = q_k) = P(q = q_k \mid f_{1,j} = b_1 \ \ldots \ f_{m,j} = b_m)$ *and* $P(T_k = a_k) = P(T_k = a_k \mid f_{1,j} = b_1 \ \ldots \ f_{m,j} = b_m)$ *for each* $b_i \in \{0,1\}$*, where* q *is the current state of the UTM and* T_k *is the contents of the* k*th tape square.*

Proof. The proof is a straightforward consequence of the two quantum randomness axioms, the boolean functions in table 4 characterizing the firing activity of $f_{1,1} \ldots, f_{1,j}, \ldots f_{93,j}, \ldots$ and the definition of conditional probability.

Let event $\mathcal{Q}$ correspond to $q = q_k$. Let event $\mathcal{B}$ correspond to $f_{1,j} = b_1$. Then event $\overline{\mathcal{B}}$ corresponds to $f_{1,j} = 1 - b_1$. The two quantum randomness axioms imply $P(R_k = 0)$

Table 4: Boolean functions $B_k : \{0,1\}^{15} \to \{0,1\}^{15}$ for $\overline{\eta}_3$

Boolean Function	Definition
$B_{(111,00)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_0 = x_0$. When $k \neq 0$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(110,00)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_1 = x_1$. When $k \neq 1$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(110,01)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_2 = x_2$. When $k \neq 2$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(110,10)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_3 = x_3$. When $k \neq 3$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(101,00)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_4 = x_4$. When $k \neq 4$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(101,01)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_5 = x_5$. When $k \neq 5$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(101,10)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_6 = x_6$. When $k \neq 6$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(100,00)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_7 = x_7$. When $k \neq 7$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(100,10)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_8 = x_8$. When $k \neq 8$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(011,01)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_9 = x_9$. When $k \neq 9$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(011,10)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_{10} = x_{10}$. When $k \neq 10$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(010,00)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_{11} = x_{11}$. When $k \neq 11$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(010,01)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_{12} = x_{12}$. When $k \neq 12$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{(010,11)}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_{13} = x_{13}$. When $k \neq 13$, $y_k = 1 - x_k$. $p_3 = b_3$.
$B_{\overline{\eta}_3{}^{-1}\{0\}}(x_0, x_1, \ldots, x_{13}, b_3) = (y_0, y_1, \ldots, y_{13}, p_3)$	$y_k = 1 - x_k$ for each k. $p_3 = 1 - b_3$.
$x_k = 1$ if element R_k fires.	If $b_3 = 1, P_3$ fires iff $(UWX, YZ) \in \overline{\eta}_3{}^{-1}\{1\}$.
$x_k = 0$ if element R_k doesn't fire.	If $b_3 = 0, P_3$ fires iff $(UWX, YZ) \in \overline{\eta}_3{}^{-1}\{0\}$.

$= P(R_k = 1) = \frac{1}{2}$; from table 4, this implies $P(\mathcal{B}) = P(\overline{\mathcal{B}}) = \frac{1}{2}$. Now $P(\mathcal{B})\ P(\mathcal{Q} \mid \mathcal{B}) = P(\mathcal{Q} \cap \mathcal{B})$. Also, $P(\mathcal{Q}) = P(\mathcal{Q} \cap \mathcal{B}) + P(\mathcal{Q} \cap \overline{\mathcal{B}})$. Then $P(\mathcal{Q}) = \frac{1}{2}P(\mathcal{Q} \mid \mathcal{B}) + \frac{1}{2}P(\mathcal{Q} \mid \overline{\mathcal{B}})$ because $P(\overline{\mathcal{B}})P(\mathcal{Q} \mid \overline{\mathcal{B}}) = P(\mathcal{Q} \cap \overline{\mathcal{B}})$. Lastly, $P(\mathcal{Q} \mid \mathcal{B}) = P(\mathcal{Q} \mid \overline{\mathcal{B}})$ because $P(R_k = 0) = P(R_k = 1) = P(\mathcal{B}) = P(\overline{\mathcal{B}}) = \frac{1}{2}$ and from the properties of the boolean functions in table 4. Thus, $P(\mathcal{Q}) = P(\mathcal{Q} \mid \mathcal{B})$ which means that $P(q = q_k) = P(q = q_k \mid f_{1,j} = b_1)$. Similar steps can be repeated to show that $P(q = q_k) = P(q = q_k \mid f_{1,j} = b_1 \ \ldots \ f_{m,j} = b_m)$ and $P(T_k = a_k) = P(T_k = a_k \mid f_{1,j} = b_1 \ \ldots \ f_{m,j} = b_m)$. □

It is worth mentioning that if the system reveals information about the dynamic connections from R_k to D_k or the quantum random bits generated or firing activity of elements R_k or those elements representing the UTM tape or state, then the perfect secrecy doesn't hold.

Corollary 1. *Consider an unbounded number of computable steps generated by the AEM in procedure 2, where the sequence of UTM instructions is $I_1, I_2, \ldots, I_k, \ldots$, and $I_k \in Q \times \mathcal{A}$ as defined in table 2. Define function $f : \mathbb{N} \to Q \times \mathcal{A}$ as $f(k) = I_k$. If an adversary can only eavesdrop on $\mathfrak{g}$ (the firing activity computing $\overline{\eta}$), then there does not exist a Turing machine that can map $\mathfrak{g}$ back to f.*

Proof. This corollary follows from definition 1 and the work in lemma 4.1 and theorems 4.2 and 4.3. □

5 Summary

By using a finite AEM program and the meta command, and executing procedure 2, any Turing machine program – with a computable unbounded execution – can be executed with AEM firing patterns that are Turing incomputable. For an unbounded number of computable UTM execution steps, when appropriate parameters of the AEM execution are not revealed to an adversary, then there does not exist a Turing machine that can map $\mathfrak{g}$ (i.e., the active element

firing patterns computing $\overline{\eta}$) back to the sequence of universal Turing machine instructions executed by the AEM.

6 Acknowledgements

I would like to thank Wolfgang Halang, Michael Jones, Don Knuth, David Lewis, A. Mayer, Lutz Mueller, Don Saari and Mario Stipčević for their helpful advice.

References

[1] P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22:563–591, 1980.

[2] P. Benioff. Quantum mechanical Hamiltonian models of Turing machines that dissipate no energy. *Physics Review Letter*, 48:1581–1585, 1980.

[3] Cristian S. Calude, Michael J. Dinneen, Monica Dumitrescu, and Karl Svozil. Experimental Evidence of Quantum Randomness Incomputability. *Physics Review A*, 82(022102):1–8, 2010.

[4] Cristian S. Calude and Karl Svozil. Quantum randomness and value indefiniteness. *Advanced Science Letters*, 1(2):165–168, 2008.

[5] Alonzo Church. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58:345–363, 1936.

[6] Fred Cohen. Computer Viruses Theory and Experiments. *Computers and Security*, 6(1):22–35, February 1987.

[7] S. Barry Cooper. The Incomputable Alan Turing. In *Turing 2004: A celebration of his life and achievements*. Electronic Workshops in Computing, June 2004.

[8] S. Barry Cooper and Piergiorgio Odifreddi. *Incomputability in Nature*. Plenum Publishers, 2003.

[9] Martin Davis. *Computability and Unsolvability*. Dover Publications, 1982.

[10] Martin Davis. *The Myth of Hypercomputation*. Springer-Verlag, 2004.

[11] Martin Davis. Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178(1):4–7, July 2006.

[12] K. de Leeuw, E.F. Moore, C.E. Shannon, and N. Shapiro. *Computability of Probabilistic Machines*. Princeton University Press, 1956.

[13] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of London Mathematical Society. Series A*, 400(1818):97–117, 1985.

[14] Gábor Etesi and István Németi. Non-Turing computations via Malament-Hogarth spacetimes. *International Journal of Theoretical Physics*, 41(2):341–370, 2002.

[15] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[16] Richard Feynman. Quantum mechanical computers. *Foundations of Physics*, 16:507–531, 1986.

[17] Eric Filiol. *Malicious Cryptology and Mathematics*. Intech, 2012.

[18] Michael Stephen Fiske. The Active Element Machine. In *Proceedings of Computational Intelligence. Autonomous Systems: Developments and Trends*, volume 391, pages 69–96. Springer-Verlag, 2011.

[19] L.K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physics Review Letters*, 79:325–328, 1997.

[20] Wolfgang Halang and Boudewijn Hoogeboom. *The concept of time in the specification of real-time systems*. Kluwer Academic Publishers, 1992.

[21] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction To The Theory of Neural Computation*. Addison-Wesley, Redwood City, California, 1991.

[22] Mark Hogarth. Does general relativity allow an observer to view an eternity in a finite time? *Foundations of Physics Letters*, 5(2):173 – 181, 1992.

[23] Mark Hogarth. Non-Turing Computers and Non-Turing Computability. In *Proceedings of the Biennial Meeting of the Philosophy of Science Association*, volume 1, pages 126 – 138. University of Chicago Press, 1994.

[24] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554 – 2558, 1982.

[25] John J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:33 – 36, 1995.

[26] John J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141 – 152, 1985.

[27] Tien Kieu. Quantum Algorithm for Hilbert's Tenth Problem. `http://arxiv.org/abs/quant-ph/0110136`, 2001.

[28] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology - Crypto 99 Proceedings. Lecture Notes in Computer Science*, volume 1666. Springer-Verlag, 1999.

[29] Harry R. Lewis and Christos H. Papadimitriou. *Elements Of The Theory Of Computation*. Prentice-Hall, 1981.

[30] Yuri Manin. *A Course in Mathematical Logic*. Springer-Verlag, 1977.

[31] Yuri Manin. *Computable and Uncomputable (in Russian)*. Sovetskoye Radio, Moscow, 1980.

[32] Warren S. McCulloch and Walter Pitts. A logical calculus immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 – 133, 1943.

[33] Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall (1st edition), Englewood Cliffs, New Jersey, 1967.

[34] Marvin Minsky and Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, Massachusetts, 1969.

[35] Wilfrid Rall. *The Theoretical Foundation of Dendritic Function. Selected Papers of Wilfrid Rall with Commentaries. Edited by Idan Segev, John Rinzel, and Gordon Shepherd*. MIT Press, Cambridge, Massachusetts, 1995.

[36] Abraham Robinson. *Non-standard Analysis*. Princeton University Press (Revised Edition), Princeton, New Jersey, 1996.

[37] Frank Rosenblatt. Two theorems of statistical separability in the perceptron. In *Proceedings of a Symposium on the Mechanization of Thought Processes*, pages 421 – 456, London, 1959. Her Majesty's Stationary Office.

[38] Claude Shannon. Communication Theory of Secrecy Systems. `http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf`, 1949.

[39] Peter W. Shor. Algorithms for quantum computation: discrete log and factoring. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 2 – 22, 1994.

[40] Hava Siegelmann. Computation Beyond the Turing Limit. *Science*, 268(5210):545 – 548, April 1995.

[41] Robert Soare. Computability and Recursion. *Bulletin of Symbolic Logic*, 2:284 – 321, 1996.

[42] André Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. Optical quantum random number generator. *Journal of Modern Optics*, 47(4):595 – 598, 2000.

[43] Mario Stipčević and B. Medved Rogina. Quantum random number generator based on photonic emission in semiconductors. *Review of Scientific Instruments*, 78:1 – 7, 2007.

[44] H. E. Sturgis and J. C. Shepherdson. Computability of Recursive Functions. *Journal Assoc.*

Computing Machines, 10:217–255, 1963.

[45] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society. Series 2*, 42(3 and 4):230–265, 1936.

7 Appendix

7.1 Turing Machine

Define a Turing Machine, where the program definition η is explicitly represented as a function instead of quintuples ([9], [45]).

Definition 5. *Turing Machine*

A Turing machine is a triple $(Q, \mathcal{A}, \eta)$ where

- Q is a finite set of states that does not contain a unique halting state, represented as $\mathcal{H}$.
- When machine execution begins, the machine is in an initial state s and $s \in Q$.
- $\mathcal{A}$ is a finite set of alphabet symbols that are read from and written to the tape.
- L and R represent advancing the tape head to the left or right square, respectively.
- η is a function where $\eta : Q \times \mathcal{A} \to Q \times \mathcal{A} \times \{L, R\} \cup \{\mathcal{H}\} \times \mathcal{A} \times \{h\}$. η acts as the *program* for the Turing machine. For each q in Q and α in $\mathcal{A}$, $\eta(q, \alpha) = (r, \beta, x)$ describes how machine $(Q, \mathcal{A}, \eta)$ executes one computational step. When in state q and scanning alphabet symbol α on the tape:
  - Machine $(Q, \mathcal{A}, \eta)$ changes to state r.
  - Machine $(Q, \mathcal{A}, \eta)$ rewrites alphabet symbol α as symbol β on the tape.
  - If $x = L$, then machine $(Q, \mathcal{A}, \eta)$ moves its tape head one square to the left on the tape and is subsequently scanning the symbol in this square.
  - If $x = R$, then machine $(Q, \mathcal{A}, \eta)$ moves its tape head one square to the right on the tape and is subsequently scanning the symbol in this square.
  - If $x = h$, machine $(Q, \mathcal{A}, \eta)$ enters the halting state $\mathcal{H}$ and the machine stops executing.

Definition 6. *Turing Machine Tape*
The Turing machine tape T is represented as a function $T : \mathbb{Z} \to \mathcal{A}$ where $\mathbb{Z}$ is the integers.

The tape T is M-bounded if there exists a bound $M > 0$ such that $T(k) = T(j)$ whenever $|k|, |j| \geq M$. The Turing machine definitions in [9] and [45] assume the initial tape, before program execution begins, is M-bounded and the tape contains only blank symbols, denoted here as #, outside the bound. The symbol on the kth square of the tape is $T(k)$.

Definition 7. *Configuration with Tape Head Location*

Let $(Q, \mathcal{A}, \eta)$ be a Turing machine with tape T. A configuration is an element of the set $C = (Q \cup \{\mathcal{H}\}) \times \mathbb{Z} \times \{T : T \text{ is tape with range } \mathcal{A}\}$. If (q, k, T) is a configuration, then k is called the tape head location.

Consider the configuration $(p, 2, \ldots \#\#\alpha\underline{\beta}\#\# \ldots)$. The 1st coordinate indicates that the Turing machine is in state p. The 2nd coordinate indicates that its tape head is currently scanning tape square 2, denoted as $T(2)$. The 3rd coordinate indicates that tape square 1 contains symbol α, tape square 2 contains symbol β, and all other tape squares contain the # symbol. The underlining of β indicates that the tape head is currently scanning tape square 2.

Definition 8. *Turing Machine Computational Step*

Given Turing machine $(Q, \mathcal{A}, \eta)$ in current configuration (q, k, T) such that $T(k) = \alpha$. After the execution of one computational step, the new configuration is determined by one and only one of the three cases.

1. $(r, k-1, S)$ if $\eta(q, \alpha) = (r, \beta, L)$ for non-halting state r.
2. $(r, k+1, S)$ if $\eta(q, \alpha) = (r, \beta, R)$ for non-halting state r.
3. $(\mathcal{H}, k, T)$ if $\eta(q, \alpha) = (\mathcal{H}, \alpha, h)$ for halting state $\mathcal{H}$.

In cases (1) and (2) the new tape $S(j) = T(j)$ whenever $j \neq k$ and $S(k) = \beta$. In case (3) the machine execution halts. Sometimes (q, α) is called a Turing machine instruction.

If the machine is currently in configuration (q_0, k_0, T_0) and over the next n steps the sequence of machine configurations (points) is (q_0, k_0, T_0), (q_1, k_1, T_1), (q_2, k_2, T_2), $\ldots$, (q_n, k_n, T_n), then this execution sequence is sometimes called the next n computational steps.

Table 5: Minsky Universal Turing Machine with Program η from [33]

	y	0	1	A
q_1	$(q_1, 0, L)$	$(q_1, 0, L)$	$(q_2, 1, L)$	$(q_1, 1, L)$
q_2	$(q_1, 0, L)$	(q_2, y, R)	(q_2, A, R)	(q_6, y, R)
q_3	(q_3, y, L)	$(\mathcal{H}, 0, h)$	(q_3, A, L)	$(q_4, 1, L)$
q_4	(q_4, y, L)	(q_5, y, R)	$(q_7, 1, L)$	$(q_4, 1, L)$
q_5	(q_5, y, R)	(q_3, y, L)	(q_5, A, R)	$(q_5, 1, R)$
q_6	(q_6, y, R)	(q_3, A, L)	(q_6, A, R)	$(q_6, 1, R)$
q_7	$(q_7, 0, R)$	(q_6, y, R)	$(q_7, 1, R)$	$(q_2, 0, R)$

State set $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$. Alphabet $\mathcal{A} = \{y, 0, 1, A\}$. Halt state $\mathcal{H}$.

7.2 Active Element Machine Architecture

Define the extended integers as $\overline{\mathbb{Z}} = \{m + kdT : m, k \in \mathbb{Z}$ and dT is a fixed infinitesimal$\}$. For more on infinitesimals, see keyword 1 and reference [36].

Definition 9. *Machine Architecture*

Γ, Ω, and Δ are index sets that index the input, computational, and output active elements, respectively. Depending on the machine architecture, the intersections $\Gamma \cap \Omega$ and $\Omega \cap \Delta$ can be empty or non-empty. A machine architecture, denoted as $\mathcal{M}(\mathcal{I}, \mathcal{E}, \mathcal{O})$, consists of a collection of input active elements, denoted as $\mathcal{I} = \{E_i : i \in \Gamma\}$; a collection of computational active elements $\mathcal{E} = \{E_i : i \in \Omega\}$; and a collection of output active elements $\mathcal{O} = \{E_i : i \in \Delta\}$.

Each computational and output active element, E_i, has the following components and properties:

- A threshold θ_i
- A refractory period r_i where $r_i > 0$.
- A collection of pulse amplitudes $\{A_{ki} : k \in \Gamma \cup \Omega\}$.
- A collection of transmission times $\{\tau_{ki} : k \in \Gamma \cup \Omega\}$, where $\tau_{ki} > 0$ for all $k \in \Gamma \cup \Omega$.
- A function of time, $\Psi_i(t)$, representing the time active element E_i last fired. $\Psi_i(t) = \sup\{s : s < t \text{ and } g_i(s) = 1\}$, where $g_i(s)$ is the output function of active element E_i and is defined below. The sup is the least upper bound and is always defined here, whence Ψ_i is well-defined.
- A binary output function, $g_i(t)$, representing whether active element E_i fires at time t. The value of $g_i(t) = 1$ if $\sum A_{ki}(t) > \theta_i$ where the sum ranges over all $k \in \Gamma \cup \Omega$ and $t \geq \Psi_i(t) + r_i$. In all other cases, $g_i(t) = 0$. For example, $g_i(t) = 0$, if $t < \Psi_i(t) + r_i$.
- A set of firing times of active element E_k within active element E_i's integrating window, $W_{ki}(t) = \{s : \text{active element } E_k \text{ fired at time } s \text{ and } 0 \leq t - s - \tau_{ki} < \omega_{ki}\}$. Let $|W_{ki}(t)|$ denote the number of elements in the set $W_{ki}(t)$. If $W_{ki}(t) = \emptyset$, then $|W_{ki}(t)| = 0$.
- A collection of input functions, $\{\phi_{ki} : k \in \Gamma \cup \Omega\}$, each a function of time, and each representing pulses coming from computational active elements, and input active elements. The value of the input function is computed as $\phi_{ki}(t) = |W_{ki}(t)|A_{ki}(t)$.
- The refractory periods, transmission times and pulse widths are positive integers; and pulse amplitudes and thresholds are integers. These parameters are a function of time (i.e., $\theta_i(t), r_i(t), A_{ki}(t), \omega_{ki}(t), \tau_{ki}(t)$). Time t is an element of the extended integers $\overline{\mathbb{Z}}$.

Input active elements that are not computational have the same characteristics as computational elements, except they have no inputs ϕ_{ki} coming from elements in this machine. Input elements are assumed to be externally firable. An external source such as the environment or an output element from another distinct machine $\mathcal{M}(\mathcal{I}', \mathcal{E}', \mathcal{O}')$ can cause an input element to fire. An input element can fire at any time after its refractory period has expired. An element can be an input and computational element. Similarly, an element can be an output and computational element. Alternatively, when an output element, E_i, is not a computational element, where $i \in \Delta - \Omega$, then E_i does not send pulses to elements in this machine.

If $g_i(s) = 1$, this means active element E_i *fired* at time s. The *refractory period*, r_i, is the amount of time that must elapse after active element E_i just fired before E_i can fire again. The *transmission time*, τ_{ki}, is the amount of time it takes for active element E_i to find out that active element E_k has fired. The *pulse amplitude*, A_{ki}, represents the strength of the pulse that active element E_k transmits to active element E_i after active element E_k has fired. After this pulse reaches E_i, the *pulse width* ω_{ki} represents how long the pulse lasts as input to active element E_i. If $A_{ki} = 0$, then there is no connection from active element E_k to active element E_i.

7.3 Active Element Machine Programming Language

This subsection describes a programming language for the active element machine. There are five types of commands `Element`, `Connection`, `Fire`, `Program` and `Meta`.

Syntax 1. *AEM Program*
In Backus-Naur form, an AEM program is defined as follows.

```
<AEM_program> ::= <cmd_sequence>
<cmd_sequence> ::= "" | <AEM_cmd><cmd_sequence> | <program_def><cmd_sequence>
<AEM_cmd> ::= <element_cmd> | <fire_cmd> | <meta_cmd> | <cnct_cmd> | <program_cmd>
```

Syntax 2. *AEM Symbols and Extended Integer Expressions*

```
<ename> ::= <int> | <symbol>
<symbol> ::= <char_symbol><str_tail> | (<ename> . . . <ename>)
<str_tail> ::= "" | <char_symbol><str_tail> | 0<str_tail> | <pos_int><str_tail>
<char_symbol> ::= <letter> | <special_char>
<letter> ::= <lower_case> | <upper_case>
<lower_case> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<upper_case> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
<special_char> ::=  _
```

The following rules represent the extended integers, addition and subtraction.

```
<int> ::= <pos_int> | <neg_int> | 0
<neg_int> ::=  - <pos_int>
<pos_int> ::=  <non_zero><digits>

<digits> ::= <numeral> | <numeral><digits>
<non_zero> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<numeral> ::= "" | <non_zero> | 0

<aint> ::= <aint><math_op><d> | <d><math_op><aint> | <d>
<math_op> ::= + | -
<d> ::= <int> | <symbol_string> | <infinitesimal>
<infinitesimal> ::= dT
```

Command 1. *Element*

An `Element` *command specifies the time when an active element is created or its parameter values are updated. This command has the following Backus-Naur syntax.*

```
<element_cmd> ::= (Element (Time <aint>) (Name <ename>) (Threshold <int>)
                           (Refractory <pos_int>) (Last <int>))
```

The keyword `Time` tags the time value s (extended integer) at which the element is created or updated. If the name symbol value is `E`, the keyword `Name` tags the name `E` of the active element. The keyword `Threshold` tags the threshold $\theta_E(s)$ assigned to `E`. `Refractory` tags the refractory value $r_E(s)$. The keyword `Last` tags the last time fired value $\Psi_E(s)$.

Command 2. *Connection*

A `Connection` *command creates or updates a connection from one active element to another active element. This command has the following Backus-Naur syntax.*

```
<cnct_cmd> ::= (Connection (Time <aint>) (From <ename>) (To <ename>)
                           [(Amp <int>) (Width <pos_int>) (Delay <pos_int>)])
```

The keyword `Time` tags the time value s at which the connection is created or updated. The keyword `From` tags the name `F` of the active element that sends a pulse with these updated values. The keyword `To` tags the name `T` of the active element that receives a pulse with these updated values. The keyword `Amp` tags the pulse amplitude value $A_{FT}(s)$ that is assigned to

this connection. The keyword `Width` tags the pulse width value $\omega_{FT}(s)$. The keyword `Delay` tags the transmission time $\tau_{FT}(s)$.

When the AEM clock reaches time `s`, `F` and `T` are name values that must be the name of an element that already has been created or updated before or at time s. Not all of the connection parameters need to be specified in a connection command. If the connection does not exist beforehand and the Width and Delay values are not specified appropriately, then the amplitude is set to zero and this zero connection has no effect on the AEM computation. The connection exists indefinitely with the same parameter values until a new connection is executed at a later time between `From` element `F` and `To` element `T`.

Command 3. *`Fire`*

The `Fire` command has the following Backus-Naur syntax.

```
<fire_cmd> ::= (Fire (Time <aint>) (Name <ename>) )
```

The `Fire` command fires the active element indicated by the `Name` tag at the time indicated by the `Time` tag. This command can be used to fire input active elements.

Command 4. *`Program`*

The `Program` combines a sequence of commands into a single command. It has the following definition syntax.

```
<program_def> ::= (Program <pname> [(Cmds <cmds>)]  [(Args <args>)] <cmd_sequence>)
<pname> ::= <ename>
<cmds> ::= <cmd_name> | <cmd_name><cmds>
<cmd_name> ::=  Element | Connection | Fire | Meta | <pname>
<args> ::= <symbol> | <symbol><args>
```

The `Program` command has the following execution syntax.

```
<program_cmd> ::= (<pname> [(Cmds <cmds>)] [(Args <args_cmd>)] )
<args_cmd> ::= <ename> | <ename><args_cmd>
```

Keyword 1. *`dT`*

The keyword `dT` represents a positive infinitesimal amount of time.

If `m` and `n` are integers and `0` $\leq$ `m < n`, then `mdT < ndT`. Furthermore, `dT > 0` and `dT` is less than every positive rational number. Similarly, `-dT < 0` and `-dT` is greater than every negative rational number. The infinitesimal `dT` helps coordinate almost simultaneous events that are non-commutative or indeterminate. For example, element `A` could be about to receive a pulse from element `B` at the same time that a connection between them is removed.

Keyword 2. *`clock`*

The keyword `clock` evaluates to an integer, which is the current active element machine time. `clock` is an instance of `<ename>`.

If the current AEM time is 5, then command
`(Element (Time clock) (Name clock) (Threshold 1) (Refractory 1) (Last -1))` executes as
`(Element (Time 5) (Name 5) (Threshold 1) (Refractory 1) (Last -1))`

After `(Element (Time clock) (Name clock) (Threshold 1) (Refractory 1) (Last -1))` is created, then at each time step this command is executed with the current time of the AEM. If this command is in the original AEM program before the clock starts at 0, then the following sequence of elements named 0, 1, 2, ... are created.

```
(Element (Time 0) (Name 0) (Threshold 1) (Refractory 1) (Last -1))
(Element (Time 1) (Name 1) (Threshold 1) (Refractory 1) (Last -1))
(Element (Time 2) (Name 2) (Threshold 1) (Refractory 1) (Last -1))  . . .
```

Command 5. *Meta*

The `Meta` *command causes a command to execute when an element fires within a window of time. This command has the following execution syntax.*

```
<meta_cmd> ::= (Meta (Name <ename>) [<win_time>] <AEM_cmd>)
<win_time> ::= (Window <aint> <aint>)
```

To understand the behavior of the meta command, consider the execution of

```
(Meta (Name E) (Window l  w) (C (Args t a))
```

where `E` is the name of the active element. The keyword `Window` tags an interval, which is called a *window of time*. `l` is an integer, which locates one of the boundary points of the window of time. Usually, `w` is a positive integer, so the window of time is `[l, l+w]`. If `w` is a negative integer, then the window of time is `[l+w, l]`.

The command `C` executes each time that `E` fires during the window of time, which is either `[l, l+w]` or `[l+w, l]`, depending on the sign of `w`. If the window of time is omitted, then command `C` executes at any time that element `E` fires.

In regard to the meta command, the following assumption is analogous to the Turing machine tape being unbounded as Turing program execution proceeds. (See Definitions 5 and 6.) During execution of a finite active element program, an active element can fire and due to one or more meta commands, new elements and connections can be added to the machine. As a consequence, at any time the active element machine only has a finite number of computing elements and connections but the number of elements and connections can be unbounded as a function of time as the active element program executes.

7.4 Active Element Machine Computation

In section 3, the firing patterns of active elements represent the computation of a boolean function. Firing representations, machine computation and interpretation are defined below.

Definition 10. *Firing Representation*

Consider active element E_i's firing times in the interval of time $W = [t_1, t_2]$. Let s_1 be the earliest firing time of E_i lying in W, and s_n the latest firing time lying in W. Then E_i's firing sequence $F(E_i, W) = [s_1, \ldots, s_n] = \{s \in W : g_i(s) = 1\}$ is called a firing sequence of the active element E_i over the window of time W. From active elements $\{E_1, E_2, \ldots, E_n\}$, create the tuple $(F(E_1, W), F(E_2, W), \ldots, F(E_n, W))$, which is called a firing representation of the active elements $\{E_1, \ldots, E_n\}$ within the window of time W.

As E_i's refractory period is a positive integer and t_1 and t_2 are finite, observe that s_1 and s_n are well-defined and the set $F(E_i, W)$ is finite. At the machine level of interpretation, firing representations (firing patterns) express the input to, the computation of, and the output of an active element machine. At a more abstract level, firing representations can represent a sequence of symbols, a spatio-temporal pattern, or even a family of program instructions.

Definition 11. *Sequence of Firing Representations*

Let $W_1, \ldots, W_n$ be a sequence of time intervals. Let $\mathcal{F}(\mathcal{E}, W_1) = (F(E_1, W_1), F(E_2, W_1), \ldots, F(E_n, W_1))$ be a firing representation of active elements $\mathcal{E} = \{E_1, \ldots, E_n\}$ over the interval W_1. In general, let $\mathcal{F}(\mathcal{E}, W_k) = (F(E_1, W_k), F(E_2, W_k), \ldots F(E_n, W_k))$ be a firing representation over the interval of time W_k. From these, a sequence of firing representations, $[\mathcal{F}(\mathcal{E}, W_1), \mathcal{F}(\mathcal{E}, W_2), \ldots, \mathcal{F}(\mathcal{E}, W_n)]$ is created.

Definition 12. *Machine Computation*

Let $[\mathcal{F}(\mathcal{E},W_1),\mathcal{F}(\mathcal{E},W_2),\ldots,\mathcal{F}(\mathcal{E},W_n)]$ *be a sequence of firing representations. Suppose* $[\mathcal{F}(\mathcal{E},S_1),\ \mathcal{F}(\mathcal{E},S_2),\ \ldots,\mathcal{F}(\mathcal{E},S_m)]$ *is some other sequence of firing representations. Suppose machine architecture* $\mathcal{M}(\mathcal{I},\mathcal{E},\mathcal{O})$ *has input active elements* $\mathcal{I}$ *fire with the pattern* $[\mathcal{F}(\mathcal{E},S_1),\ \mathcal{F}(\mathcal{E},S_2),\ \ldots,\mathcal{F}(\mathcal{E},S_m)]$ *and consequently* $\mathcal{M}$*'s output active elements* $\mathcal{O}$ *fire according to* $[\mathcal{F}(\mathcal{E},W_1),\ \mathcal{F}(\mathcal{E},W_2),\ \ldots,\mathcal{F}(\mathcal{E},W_n)]$*. In this case, the machine* $\mathcal{M}$ *computes* $[\mathcal{F}(\mathcal{E},W_1),\mathcal{F}(\mathcal{E},W_2),\ \ldots,\mathcal{F}(\mathcal{E},W_n)]$ *from* $[\mathcal{F}(\mathcal{E},S_1),\mathcal{F}(\mathcal{E},S_2),\ \ldots,\mathcal{F}(\mathcal{E},S_m)]$.

An active element machine is an *interpretation* between two sequences of firing representations if the machine computes the output sequence from the input sequence.

7.5 Active Element Machine Level Set Rules

Table 6: AEM Separation Rules for Level Set ${\overline{\eta}_0}^{-1}\{1\}$

Firing Pattern	Element	(U,A_i)	(W,A_i)	(X,A_i)	(Y,A_i)	(Z,A_i)	θ_{A_i}
111 10	A_0	2	2	2	2	−2	7
111 01	A_1	2	2	2	−2	2	7
111 00	A_2	2	2	2	−2	−2	5
110 11	A_3	2	2	−2	2	2	7
110 10	A_4	2	2	−2	2	−2	5
110 01	A_5	2	2	−2	−2	2	5
101 11	A_6	2	−2	2	2	2	7
101 10	A_7	2	−2	2	2	−2	5
101 01	A_8	2	−2	2	−2	2	5
100 11	A_9	2	−2	−2	2	2	5
100 10	A_{10}	2	−2	−2	2	−2	3
100 01	A_{11}	2	−2	−2	−2	2	3
100 00	A_{12}	2	−2	−2	−2	−2	1
011 11	A_{13}	−2	2	2	2	2	7
010 11	A_{14}	−2	2	−2	2	2	5

Table 7: AEM Separation Rules for Level Set ${\overline{\eta}_1}^{-1}\{1\}$

Firing Pattern	Element	(U,B_i)	(W,B_i)	(X,B_i)	(Y,B_i)	(Z,B_i)	θ_{B_i}
111 11	B_0	2	2	2	2	2	9
111 10	B_1	2	2	2	2	−2	7
111 01	B_2	2	2	2	−2	2	7
111 00	B_3	2	2	2	−2	−2	5
110 11	B_4	2	2	−2	2	2	7
110 10	B_5	2	2	−2	2	−2	5
110 01	B_6	2	2	−2	−2	2	5
110 00	B_7	2	2	−2	−2	−2	3
101 00	B_8	2	−2	2	−2	−2	3
100 01	B_9	2	−2	−2	−2	2	3
011 10	B_{10}	−2	2	2	2	−2	5
011 01	B_{11}	−2	2	2	−2	2	5
010 11	B_{12}	−2	2	−2	2	2	5
010 01	B_{13}	−2	2	−2	−2	2	3
010 00	B_{14}	−2	2	−2	−2	−2	1
001 01	B_{15}	−2	−2	2	−2	2	3

Table 8: AEM Separation Rules for Level Set ${\overline{\eta}_2}^{-1}\{1\}$

Firing Pattern	Element	(U, C_i)	(W, C_i)	(X, C_i)	(Y, C_i)	(Z, C_i)	θ_{C_i}
111 10	C_0	2	2	2	2	−2	7
111 01	C_1	2	2	2	−2	2	7
110 00	C_2	2	2	−2	−2	−2	3
101 11	C_3	2	−2	2	2	2	7
101 10	C_4	2	−2	2	2	−2	5
101 01	C_5	2	−2	2	−2	2	5
101 00	C_6	2	−2	2	−2	−2	3
100 01	C_7	2	−2	−2	−2	2	3
100 00	C_8	2	−2	−2	−2	−2	1
011 10	C_9	−2	2	2	2	−2	5
011 01	C_{10}	−2	2	2	−2	2	5
010 10	C_{11}	−2	2	−2	2	−2	3
001 11	C_{12}	−2	−2	2	2	2	5
001 10	C_{13}	−2	−2	2	2	−2	3
001 00	C_{14}	−2	−2	2	−2	−2	1

Table 9: AEM Separation Rules for Level Set ${\overline{\eta}_4}^{-1}\{1\}$

Firing Pattern	Element	(U, E_i)	(W, E_i)	(X, E_i)	(Y, E_i)	(Z, E_i)	θ_{E_i}
111 01	E_0	2	2	2	−2	2	7
110 11	E_1	2	2	−2	2	2	7
110 01	E_2	2	2	−2	−2	2	5
110 00	E_3	2	2	−2	−2	−2	3
101 11	E_4	2	−2	2	2	2	7
101 01	E_5	2	−2	2	−2	2	5
100 11	E_6	2	−2	−2	2	2	5
100 01	E_7	2	−2	−2	−2	2	3
011 11	E_8	−2	2	2	2	2	7
011 01	E_9	−2	2	2	−2	2	5
010 01	E_{10}	−2	2	−2	−2	2	3
001 11	E_{11}	−2	−2	2	2	2	5
001 01	E_{12}	−2	−2	2	−2	2	3

Table 10: AEM Separation Rules for Level Set $\overline{\eta}_5^{-1}\{1\}$

Firing Pattern	Element	(U, F_i)	(W, F_i)	(X, F_i)	(Y, F_i)	(Z, F_i)	θ_{F_i}
111 11	F_0	2	2	2	2	2	9
111 10	F_1	2	2	2	2	−2	7
111 01	F_2	2	2	2	−2	2	7
111 00	F_3	2	2	2	−2	−2	5
110 11	F_4	2	2	−2	2	2	7
110 10	F_5	2	2	−2	2	−2	5
110 01	F_6	2	2	−2	−2	2	5
101 11	F_7	2	−2	2	2	2	7
101 10	F_8	2	−2	2	2	−2	5
101 01	F_9	2	−2	2	−2	2	5
100 00	F_{10}	2	−2	−2	−2	−2	1
010 11	F_{11}	−2	2	−2	2	2	5
010 01	F_{12}	−2	2	−2	−2	2	3
010 00	F_{13}	−2	2	−2	−2	−2	1