Neuromorphic Computing is Turing-Complete

Prasanna Date
Thomas Potok
datepa@ornl.gov
potokte@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Catherine Schuman
University of Tennessee, Knoxville
Knoxville, Tennessee, USA
cschuman@utk.gov

Bill Kay
Pacific Northwest National Laboratory
Richland, Washington, USA
william.kay@pnnl.gov

ABSTRACT

Neuromorphic computing is a non-von Neumann computing paradigm that performs computation by emulating the human brain. Neuromorphic systems are extremely energy-efficient and known to consume thousands of times less power than CPUs and GPUs. They have the potential to drive critical use cases such as autonomous vehicles, edge computing and internet of things in the future. For this reason, they are sought to be an indispensable part of the future computing landscape. Neuromorphic systems are mainly used for spike-based machine learning applications, although there are some non-machine learning applications in graph theory, differential equations, and spike-based simulations. These applications suggest that neuromorphic computing might be capable of general-purpose computing. However, general-purpose computability of neuromorphic computing has not been established yet. In this work, we prove that neuromorphic computing is Turing-complete and therefore capable of general-purpose computing. Specifically, we present a model of neuromorphic computing, with just two neuron parameters (threshold and leak), and two synaptic parameters (weight and delay). We devise neuromorphic circuits for computing all the μ-recursive functions (i.e., constant, successor and projection functions) and all the μ-recursive operators (i.e., composition, primitive recursion and minimization operators). Given that the μ-recursive functions and operators are precisely the ones that can be computed using a Turing machine, this work establishes the Turing-completeness of neuromorphic computing.

CCS CONCEPTS

• **Hardware** → **Emerging technologies**; • **Computing methodologies** → **Artificial intelligence**; **Modeling and simulation**; **Machine learning**; • **Theory of computation** → **Recursive functions**; **Turing machines**.

KEYWORDS

Neuromorphic Computing, Turing-Complete, μ-Recursive Functions, Turing Machine, Computability and Complexity

ACM Reference Format:
Prasanna Date, Thomas Potok, Catherine Schuman, and Bill Kay. 2022. Neuromorphic Computing is Turing-Complete. In *International Conference on Neuromorphic Systems (ICONS 2022), July 27–29, 2022, Knoxville, TN, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3546790.3546806

ICONS 2022, July 27–29, 2022, Knoxville, TN, USA

ACM ISBN 978-1-4503-9789-6/22/07…$15.00
https://doi.org/10.1145/3546790.3546806

1 INTRODUCTION

With the impending end of Moore's law and Dennard scaling, the computing community is looking for alternatives to conventional computing approaches such as novel materials, devices, and architectures [46]. Neuromorphic computing is a compelling technology in this regard [33]. Neuromorphic computing is a non-von Neumann, brain-inspired computing paradigm that is extremely low power and can be implemented using a myriad of devices and materials, including CMOS [45]. For certain applications, neuromorphic systems are faster as well as more power efficient than CPUs and GPUs [7]. Over the years, a number of analog, digital and mixed analog-digital implementations of neuromorphic processors have been realized [5, 16, 18, 34, 41, 50]. Because of their low power nature, neuromorphic computers are expected to shine in critical use cases such as autonomous vehicles, robotics, edge computing, internet of things, and wearable technologies [21, 30, 35, 36, 39].

Neuromorphic computers were originally developed to perform spike-based neural network-style computation, and primarily used for machine learning and computational neuroscience applications [13, 43]. However, they have many characteristics that make them attractive to other application areas. For instance, neuromorphic computers are inherently massively parallel, scalable, have co-located processing and memory, and can perform event-driven, asynchronous computation. In the recent years, neuromorphic approaches have been used to address problems in graph theory, partial differential equations, constraint satisfaction optimization, and spike-based simulations [2, 3, 17, 22–24, 42, 48, 56].

The demonstration of broader applicability of neuromorphic computing begs the question: *Can neuromorphic computing be used for general-purpose computing, i.e., is it Turing-complete?* The idea that neuromorphic computing is Turing-complete is loosely held within the community. However, its Turing-completeness has never been proven. To this extent, our main contributions in this paper are:

(1) We propose a simple model of neuromorphic computing, where neurons and synapses have two parameters each.
(2) We devise neuromorphic circuits to compute all the μ-recursive functions and operators. In doing so, we show our model is Turing-complete.

2 RELATED WORK

The Turing machine, proposed by Alan Turing in 1937, is a model of computation that can express any arbitrary computation [52]. The

von Neumann architecture used in today's computers is a physical manifestation of the Turing machine [53, 55]. Contemporary to the Turing machine, Gödel and Herbrand introduced another model of computation, called the μ-recursive functions [20, 49]. It was subsequently shown that these two models were equivalent [52]. A computation model is said to be Turing-complete if it can compute the same set of functions as the Turing machine [10–12]. Since the set of μ-recursive functions is precisely that set, it suffices to show that a computation model can compute all the μ-recursive functions in order to show that it is Turing-complete.

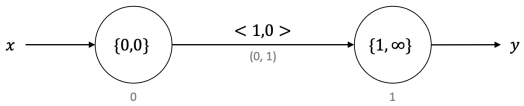
There have been some efforts to show certain models of of spiking and recurrent neural networks are Turing-complete. Maass shows that spiking neural networks with synaptic plasticity are Turing complete [32]. Siegelmann shows that analog neural networks—which are equivalent to Maass' spiking neural networks with synaptic plasticity—allow for super-Turing compute power [47]. Cabessa and Siegelmann show that recurrent neural networks have super-Turing capabilities when synaptic plasticity is assumed [9]. Cabessa shows that a rational-weighted recurrent neural network employing spike-timing-dependent-plasticity (STDP) is Turing-complete [8].

All of the above models focus on either spiking or recurrent neural networks. Some assume more complex neuron and synapse models. Most importantly, synaptic plasticity is critical to all of their arguments. Their results are restricted to neuromorphic computing models with synaptic plasticity and are not generalizable. There are Turing-completeness claims for general neuromorphic systems, as well as a notion of 'neuromorphic completeness' [25, 40, 57]. However, they are not supported by formal proofs and do not establish the Turing-completeness of neuromorphic computing.

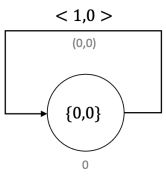
The need for a computability and complexity theory for neuromorphic computing is pressing, especially in the context of post-Moore computing. Neuromorphic computers, due to their low energy consumption, are prime candidates for co-processors and accelerators in extremely heterogeneous computing systems [2, 31]. However, because of the absence of a sound computability and complexity theory for neuromorphic computing [28], we are faced with a murky understanding of the *big-O* runtimes [27] of neuromorphic algorithms and unable to compare them to their conventional counterparts. This is not the case for other post-Moore computing paradigms such as quantum computing [15, 37, 54], which has a well-founded literature on the theory of quantum computation [6, 51]. As a result, it is possible to objectively compare quantum algorithms to their classical counterparts [4]. We have previously proposed a framework for describing the computational complexity of neuromorphic algorithms [14]. In this paper, we take the first steps to establish the computability of neuromorphic computing.

3 NEUROMORPHIC COMPUTING MODEL

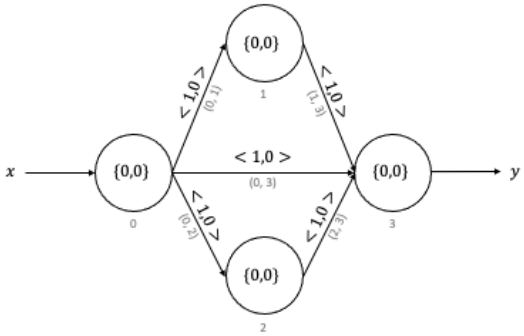
Neuromorphic computing systems implement vastly different neuron and synapse models, and the precise model details depend on the specific hardware implementation. In this work, we define an extremely stripped down neuron and synapse model that can be realized by all of the most commonly used neuron (e.g., leaky integrate-and-fire neurons, Izhikevich neurons, etc.) and synapse



(a) Symbolic notation describing a neuromorphic circuit.



(b) Self synapse `(0, 0)` going from neuron `0` to itself.



(c) Multiple synapses from neuron `0` to neuron `3` via proxy neurons.

Figure 1: Illustration of the symbolic notation, self-synapse and multiple synapses.

(e.g., synapses with and without synaptic plasticity) models in the literature by appropriately setting their parameters.

3.1 Structural Components

Our model of neuromorphic computing is comprised of two structural components: neurons and synapses. A neuron is the fundamental unit from which a neuromorphic circuit is built. A synapse constitutes a connection from a pre-synaptic (source) neuron to a post-synaptic (target) neuron. We further assume that all-to-all connectivity between neurons is possible. Neurons could also receive signals from, and send signals to external synapses, which are used for input/output (I/O) of data. Such neurons are called I/O neurons.

Figure 1a shows the symbolic notation for neurons, synapses and neuromorphic circuits throughout this paper. The circles represent neurons and arrows represent synapses. Neurons are referenced using whole numbers shown below them. Synapses are referenced using a tuple containing the references of the pre-synaptic neuron and the post-synaptic neuron. External synapses do not have explicit references. Figure 1a shows an external synapse on the left feeding an input x to the input neuron `0`. Neuron `0` is connected to neuron `1` via the synapse `(0, 1)`. Neuron `1` returns the output y

through the external synapse on the right. We allow a neuron to be connected to itself via a self-synapse as shown in Figure 1b. We do not allow a pre-synaptic and a post-synaptic neuron to be directly connected via multiple synapses in order to be consistent with our referencing scheme. Such a functionality could be achieved by routing a signal through multiple proxy neurons as shown in Figure 1c. Throughout the paper, we will represent smaller, simpler circuits using boxes, and use them in larger, more complex circuits. These boxed circuits will be referenced similar to neurons (see Figure 3 for example).

3.2 Functional Aspects and Parameters

The neurons in our model are based on the leaky integrate-and-fire neurons [1, 29]. Our neurons accumulate signals from incoming synapses in their internal state until a threshold is reached. After reaching the threshold, they spike, and send their signal via outgoing synapses. The neurons reset to an internal state of zero after spiking. Our neurons have a leak, which specifies the time it takes to push their internal state back to zero in case they do not spike. For instance, neurons with a leak of 0 (instantaneous leak) would have no recollection of their internal state. On the other hand, neurons with infinite leak would remember their internal state exactly until they spike and reset. We denote the internal state of the i^{th} neuron using v_i, which is integer-valued. The two neuron parameters—threshold and leak—are whole numbers denoted by ν_i and λ_i. We use curly parentheses to specify neuron parameters $\{\nu_i, \lambda_i\}$ in our circuit diagrams.

Synapses in our model receive signals from their pre-synaptic neuron, multiply the signal by their weight, stall for a time indicated by their delay, and deposit the signal in their post-synaptic neuron. For a synapse connecting neuron `i` to neuron `j`, its weight and delay are denoted by $\omega_{i,j}$ and $\delta_{i,j}$. The weights are integer-valued and the delays are whole numbers. In our circuit diagrams, we indicate the synaptic parameters using arrow parentheses $\langle \omega_{i,j}, \delta_{i,j} \rangle$ on top of the synapses. External synapses do not have weights or delays.

We assume it takes one unit of time for a spike to travel across any synapse in the absence of delay. This assumption is fundamental to determining the computational complexity of neuromorphic algorithms. Under this assumption, the total time taken for a spike to travel across a synapse having δ delay would be $\delta + 1$. Usually, spikes in neuromorphic systems do not have an associated value. However, it is possible to encode integers, rational numbers and real numbers as spikes using spatio-temporal encoding schemes [26, 38, 44]. For the sake of clarity, and for the ease of explaining neuromorphic circuits in Section 4, we will operate at a higher level of abstraction and let spikes have an associated integer value. This abstraction relies on the temporal delays of the synapses, and thus, synaptic delay is critical to the implementation of this approach.

A neuromorphic algorithm is defined by the configuration of a neuromorphic circuit. We assume that a neuromorphic circuit interfaces with data that is also neuromorphic, i.e. encoded as spikes. The outputs of a neuromorphic circuit are represented by the spiking behavior of the neurons. The spikes have an associated time at which they occur and a neuron on which they occur. The output of the circuit is called the spike raster, which enumerates the time and neuron for each spike that occurs over the course of the circuit's operation.

4 PROOF OF TURING COMPLETENESS

It is possible to simulate neuromorphic computations exactly on von Neumann machines using simulators such as the NEST neural simulator [19]. Since von Neumann machines are physical manifestations of the Turing machine, we know that neuromorphic computing can at most be Turing-complete. In this section, we present neuromorphic circuits for each of the three μ-recursive functions, i.e., constant function, successor function, and projection function; and each of the three μ-recursive operators, i.e., composition operator, primitive recursion operator and minimization operator. The set of μ-recursive functions and operators is known to be Turing-complete [52]. By showing that our model can compute all of the μ-recursive functions and operators, we would prove that it is Turing-complete. We let $\mathbb{N}$ be the set of natural numbers.

4.1 Constant Function (C_k)

The constant function is the first μ-recursive function. For a natural number x, it returns a constant natural number k. It is defined as:

$$C_k(x) := k \tag{1}$$

Figure 2a shows the neuromorphic circuit that computes the constant function. Neuron `1` receives the input x. Parallely, neuron `0` receives the input 1. Both neurons `1` and `0` spike as their internal states exceed their thresholds of 0. The synapse `(1, 2)` multiplies the signal x by its synaptic weight, which is 0. Parallely, synapse `(0, 2)` multiplies the signal 1 by its synaptic weight, which is k. As a result, neuron `2` receives a signal of 0 from synapse `(1, 2)` and a signal of k from synapse `(0, 2)`. It integrates the signals and spikes as its internal state k exceeds its threshold of 0. The output y received from the circuit is k as desired. The abstraction of the constant circuit is shown in Figure 2b.

4.2 Successor Function (S)

For a natural number x, the successor function returns $x + 1$. The successor of 0 is defined as 1. The successor function is defined as:

$$S(x) := x + 1 \tag{2}$$

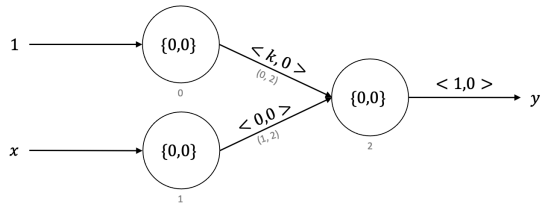
Figure 2c shows the successor function. It is similar to the constant function, with two differences. The weights of both synapses `(0, 2)` and `(1, 2)` are 1. Neuron `2` receives 1 from `(0, 2)` and x from `(1, 2)`. Its internal state becomes $x + 1$, and it spikes. The output y is $x + 1$ as desired. Figure 2d shows the abstraction of the successor function.

4.3 Projection Function (P)

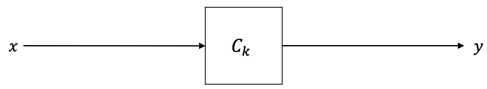
The projection function resembles getting the i^{th} element of an array. Formally, given natural numbers i and N such that $1 \leq i \leq N$, and natural numbers $x_1, \ldots, x_N$, the projection function returns x_i. It is defined as:

$$P(i, x_1, \ldots, x_N) := x_i \tag{3}$$

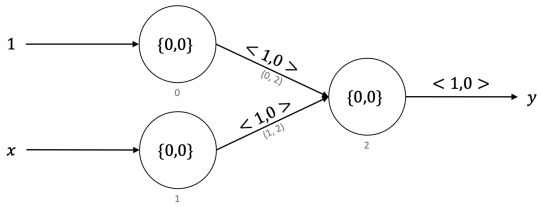
The projection function and its abstraction are shown in Figures 3 and 4. Inputs $x_1, \ldots, x_N$ are received in neurons `5N + 1` through `6N`. These neurons have a threshold of $\mathcal{M}$, which is a large positive



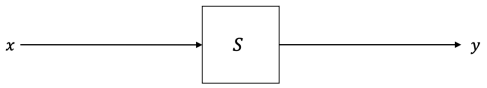
(a) Constant function circuit.



(b) Constant function abstraction.



(c) Successor function circuit.



(d) Successor function abstraction.

Figure 2: Neuromorphic circuits and abstractions for the constant and successor functions.

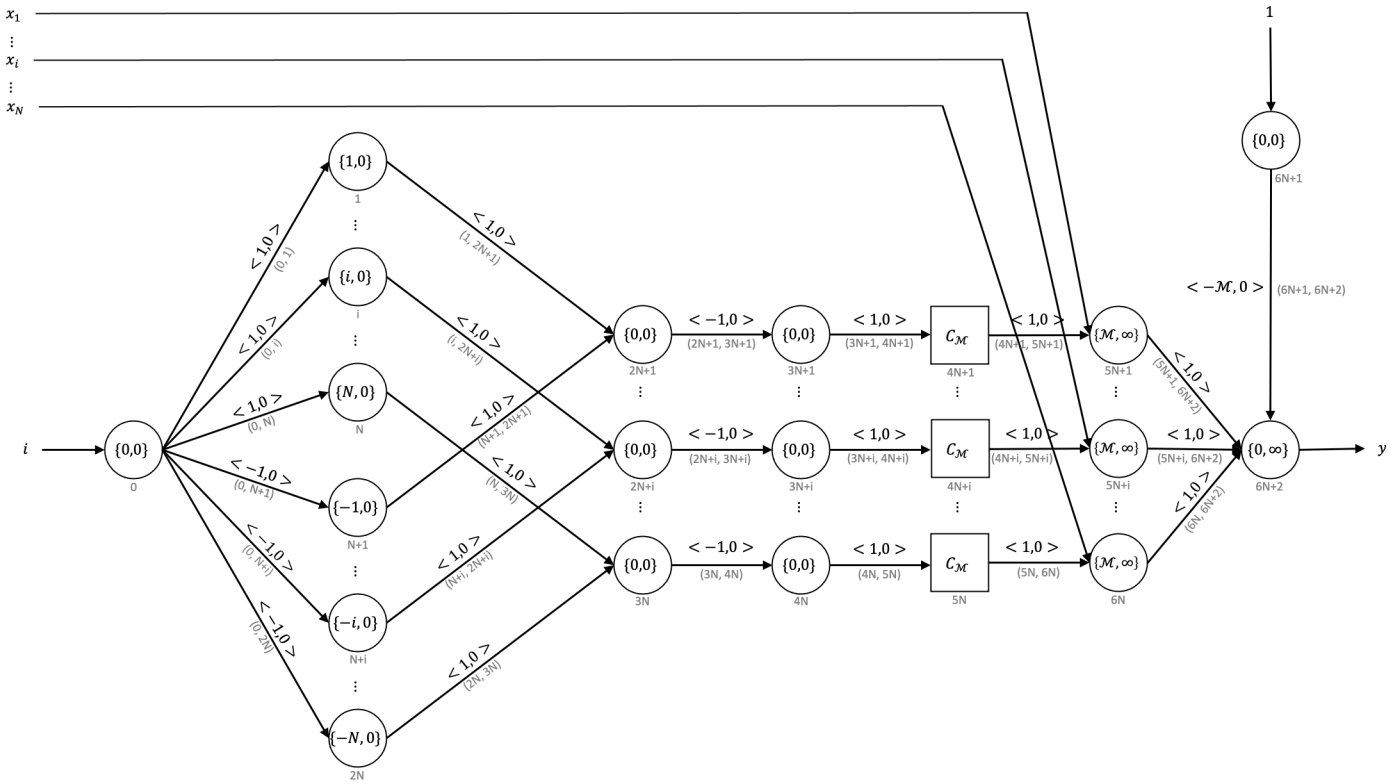


Figure 3: Neuromorphic circuit for the projection function.

number, and an infinite leak. After receiving the inputs, their internal states become $x_1, \ldots, x_N$ respectively, but they do not spike. Parallely, neuron `6N + 1` receives an input of 1 and spikes. This signal is multiplied by $-\mathcal{M}$ in the synapse `(6N + 1, 6N + 2)`. The

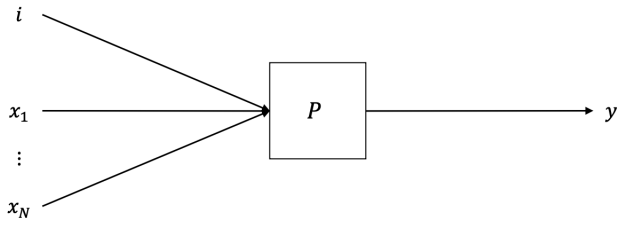


Figure 4: Abstraction for the projection function.

neuron `6N + 2` has an infinite leak and stores the value of $-\mathcal{M}$ in its internal state.

Simultaneously, neuron `0` receives i. It spikes and sends i to its outgoing synapses. Synapses `(0, 1)` through `(0, N)` multiply their signals by 1, and send them to neurons `1` through `N`. Synapses `(0, N + 1)` through `(0, 2N)` multiply their signals by -1 and send them to neurons `N + 1` through `2N`. While the neurons `1` through `N` receive a signal of i, the neurons `N + 1` through `2N` receive a signal of $-i$. The former neurons have thresholds of 1 through N, and the latter neurons have thresholds of -1 through $-N$. Thus, neuron `1` through `i` spike with a value of i and neurons `N + i` through `2N` spike with a value of $-i$.

We now look at the neurons `2N + 1` through `3N`, which have two incoming synapses each. Neuron `2N + 1` receives i from `(1, 2N + 1)`, but does not receive anything from `(N + 1, 2N + 1)`. Same is true for neurons `2N + 1` through `2N + i - 1`. All of their internal states become i. The neuron `2N + i` receives i and $-i$ from `(i, 2N + i)` and `(N + i, 2N + i)`, so that its internal state is zero. The neuron `3N` receives $-i$ from `(2N, 3N)`, but does not receive anything from `(N, 3N)`. Same is true for neurons `2N + i + 1` through `3N`. Their internal states become $-i$. Thus, neurons `2N + 1` through `2N + i - 1` have internal states of i and spike; neuron `2N + i` has an internal state of 0 and spikes; and, neurons `2N + i + 1` through `3N` have internal states of $-i$ and do not spike.

The signals from neurons `2N + 1` through `2N + i` are multiplied by -1. Neurons `3N + 1` through `3N + i - 1` receive $-i$ and do not spike. Neuron `3N + i` receives 0 and spikes. So, we have successfully isolated the signal corresponding to x_i. It is passed through the constant function, which returns $\mathcal{M}$. This value reaches the neuron `5N + i`, which has an internal state of x_i. After receiving $\mathcal{M}$ from `(4N + i, 5N + i)`, its internal state becomes $\mathcal{M} + x_i$ and it spikes. Thus, $\mathcal{M} + x_i$ reaches the neuron `6N + 2`, which has an internal state of $-\mathcal{M}$. Upon receiving the signal from `(5N + i, 6N + 2)`, its internal state becomes x_i and it spikes. The output y thus equals x_i as desired.

4.4 Composition Operator ($\circ$)

The composition operator is used for function composition in any computation model. Let M and N be natural numbers. Given the function, $h : \mathbb{N}^M \to \mathbb{N}$, and the functions, $g_1, \ldots, g_M : \mathbb{N}^N \to \mathbb{N}$, the composition operator $h \circ (g_1, \ldots, g_M)$ is defined as:

$$h \circ (g_1, \ldots, g_M)(x_1, \ldots, x_N) := h(g_1(x_1, \ldots, x_N), \ldots, g_M(x_1, \ldots, x_N)) \quad (4)$$

The composition operator and its abstraction are shown in Figures 5a and 5b. Neurons `1` through `N` receive $x_1, \ldots, x_N$. They spike and distribute the inputs to functions g_1 through g_M. Outputs from these functions are sent to the function h. Output of the function h is returned as the output of the circuit y, which equals $h \circ (g_1, \ldots, g_M)$, as desired.

4.5 Primitive Recursion Operator (ρ)

Given the functions $g : \mathbb{N}^N \to \mathbb{N}$ and $h : \mathbb{N}^{N+2} \to \mathbb{N}$, the primitive recursion operator $\rho(g, h)$ equals the function f, defined as:

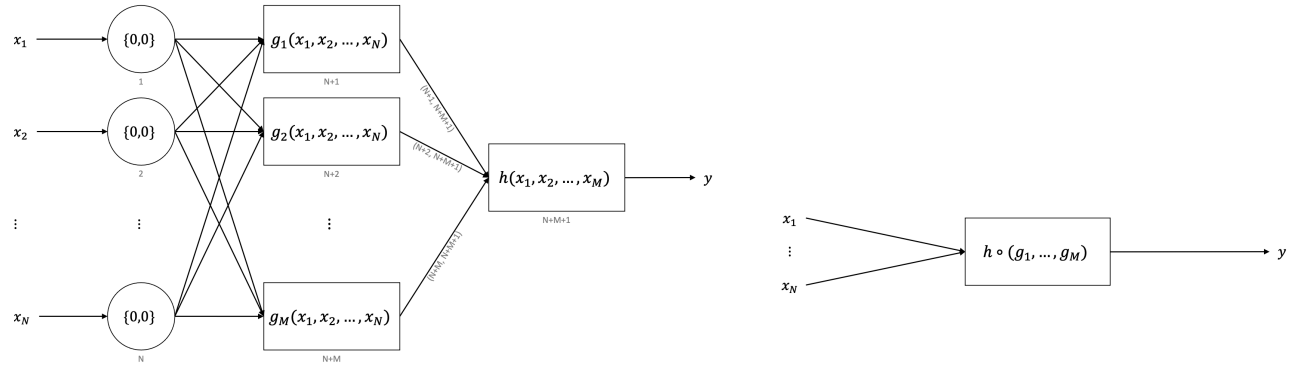
$$f(0, x_1, \ldots, x_N) := g(x_1, \ldots, x_N) \quad (5)$$

$$f(i + 1, x_1, \ldots, x_N) := h(i, f(i, x_1, \ldots, x_N), x_1, \ldots x_N) \quad (6)$$

Before describing the neuromorphic circuit for the primitive recursion operator, we describe the trigger circuit shown in Figure 6a. It stores a value v in neuron `0`, which has a threshold of $\mathcal{M}$ and infinite leak. The stored value can be erased by passing $-v$ to neuron `0`. The trigger circuit returns v stored in neuron `0` when triggered by an input of $\mathcal{M}$. When this happens, neuron `0` spikes with a value of $\mathcal{M} + v$. This value reaches neuron `2`, which has an internal state of $-\mathcal{M}$. After receiving the signal from `(0, 1)`, its internal state becomes v and it spikes. The value v is returned as the output of the trigger circuit. It is important to note that the inputs v, $-v$ and $\mathcal{M}$ are not passed to the trigger circuit simultaneously. The abstraction of the trigger circuit is shown in Figure 6b, which has three pins S, E and T, representing the store, erase and trigger functionalities.

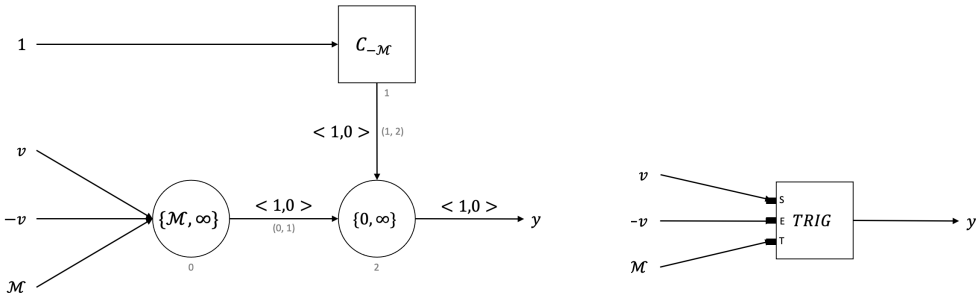
Figures 7 and 8 show the primitive recursion operator and its abstraction. The inputs i and x are passed by the user. Here, x actually represents all of $x_1, \ldots, x_N$. In an actual implementation, each of them would individually have the same connections as x, including the iterative mechanism shown by neurons `2N` and `3N`. We omit showing $x_1, \ldots, x_N$ for the sake of clarity of Figure 7. In addition to i and x, we also provide `0` and `1` to neurons `2` and `5`. The `0` sent to neuron `2` is referred to as j and is sent as an input to the function h shown in `13`. It acts as a proxy for i. The trigger circuits `1` and `9`, and the neuron `18` receive i. Trigger circuit `1` stores i for the next iteration, and `9` stores i in case we need to return $f(0, x_1, \ldots, x_N)$. Neuron `18` is used to evaluate the the expression $i - j - 1$, which when equals zero, acts as the stopping criteria. x is sent to the function g at `7`, and the neuron $2N$, from where, it is sent to the function h at `13`. j is sent to trigger circuit `3` for the next iteration. It is also sent to the function h and neuron `18`, which evaluates the stopping criteria. The input `1` to neuron `5` is sent to the trigger circuit `6` for the next iteration. It is also sent to neuron `18` for evaluating the stopping criteria.

In case i equals 0, we need to return $g(x)$ as the output of the entire circuit. This is done in the branch `9-10-11-12-25`. i is stored in the trigger circuit `9`. Parallely, x is sent to g. After $g(x)$ is evaluated, it is sent to `12`, where it waits to be returned. It is also sent to h and the constant function `8`, which in turn, triggers `9`. Upon being triggered, `9` sends i to neuron `10`, which checks if it is zero. If i equals zero, neuron `10` spikes. Else, that branch of the circuit is not activated. When neuron `10` spikes, `12` is triggered. It sends the value of $g(x)$ to neuron `25`, which is returned as desired.



(a) Composition operator circuit. **(b) Composition operator abstraction.**

Figure 5: Neuromorphic circuit and abstraction for the composition operator. All synapses are $\langle 1, 0 \rangle$ unless explicitly stated.



(a) Trigger circuit. **(b) Trigger circuit abstraction.**

Figure 6: Trigger circuit and its abstraction.

If i is greater than zero, we want to return the value of h when $i - j - 1$ equals zero. h receives x from 2N, $g(x)$ from 7 and j from 2. Parallely, $i - j - 1$ is evaluated in 18 and stored in 19. After h is evaluated, it is sent to neuron 14, which distributes it to the trigger circuits 15 and 16, the constant function 17, and h itself in preparation for the next iteration. The trigger circuit 19 is used to return the value of h if necessary, while 16 is used to erase the value in 15, if required. The constant function 17 triggers 19, which sends $i - j - 1$ to 20. Neuron 21 checks if $i - j - 1$ is zero. Parallely, $i-j-1$ is also passed to 22, which returns $\mathcal{M}$. Neuron 23 distributes $\mathcal{M}$ to 1, 3N, 3, 6 and 16. While the first four are used to initiate the next iteration, 16 is used to erase the value stored in 15. Note that the branch 23-3-4 increments j for the next iteration. In case $i-j-1$ equals zero, neuron 21 spikes. Trigger circuit 15 is eventually triggered in this branch and the value of h stored in it is returned as output of the entire circuit via neuron 25. In this case, although the branch 20-22-23-16 is active, it does not erase the value in 15 in time. The signal in the branch 20-21-24 reaches 15 first and returns the output. This is a consequence of our assumption that each synapse takes one unit of time to propagate a signal from its pre-synaptic to post-synaptic component.

Thus, when i equals zero, the recursion operator returns $g(x)$ as required. If i is greater than zero, we keep incrementing j and keep evaluating h until $i - j - 1$ equals zero. At this point, the value of h stored in 15 is returned as desired.

4.6 Minimization Operator

Given a function that only returns whole numbers for any natural number, we want to find the smallest natural number for which it returns zero. Formally, given the function $f : \mathbb{N}^{N+1} \to \mathbb{N}$, the function $\mu(f)$ is defined as:

$$\mu(f)(x_1, \ldots, x_N) := z \tag{7}$$

such that:

$$f(i, x_1, \ldots, x_N) > 0 \qquad \forall i = 1, 2, \ldots, z - 1 \tag{8}$$

$$f(z, x_1, \ldots, x_N) = 0 \tag{9}$$

Figures 9 and 10 show the minimization operator and its abstraction. Neuron 0 receives 1 and spikes. Parallely, neurons N, N + 2, ... 3N - 2 receive $x_1, x_2, \ldots, x_N$ and spike. The value from neuron 0 is stored in the trigger circuits 1, 9 and 10. 1 is used to increment the value of neuron 0 for the next iteration. 10 stores and returns the value if and when needed. 9 is used to erase the value stored 10 if required. Values from neurons N, N + 2, ... 3N - 2 are stored in the trigger circuits N + 1, N + 3, ..., 3N - 1 for the next iteration. They are also sent to the function f at 3.

f evaluates a value and sends it to neuron 4, which spikes regardless of whether the value is zero or not. If the value evaluated by f is not zero, neuron 4 sends this value to the constant circuit 7, which in turn, sends $\mathcal{M}$ to neuron 8. Neuron 8 distributes $\mathcal{M}$ to the trigger circuits 9, 1, and N + 1 through 3N - 1. Value stored

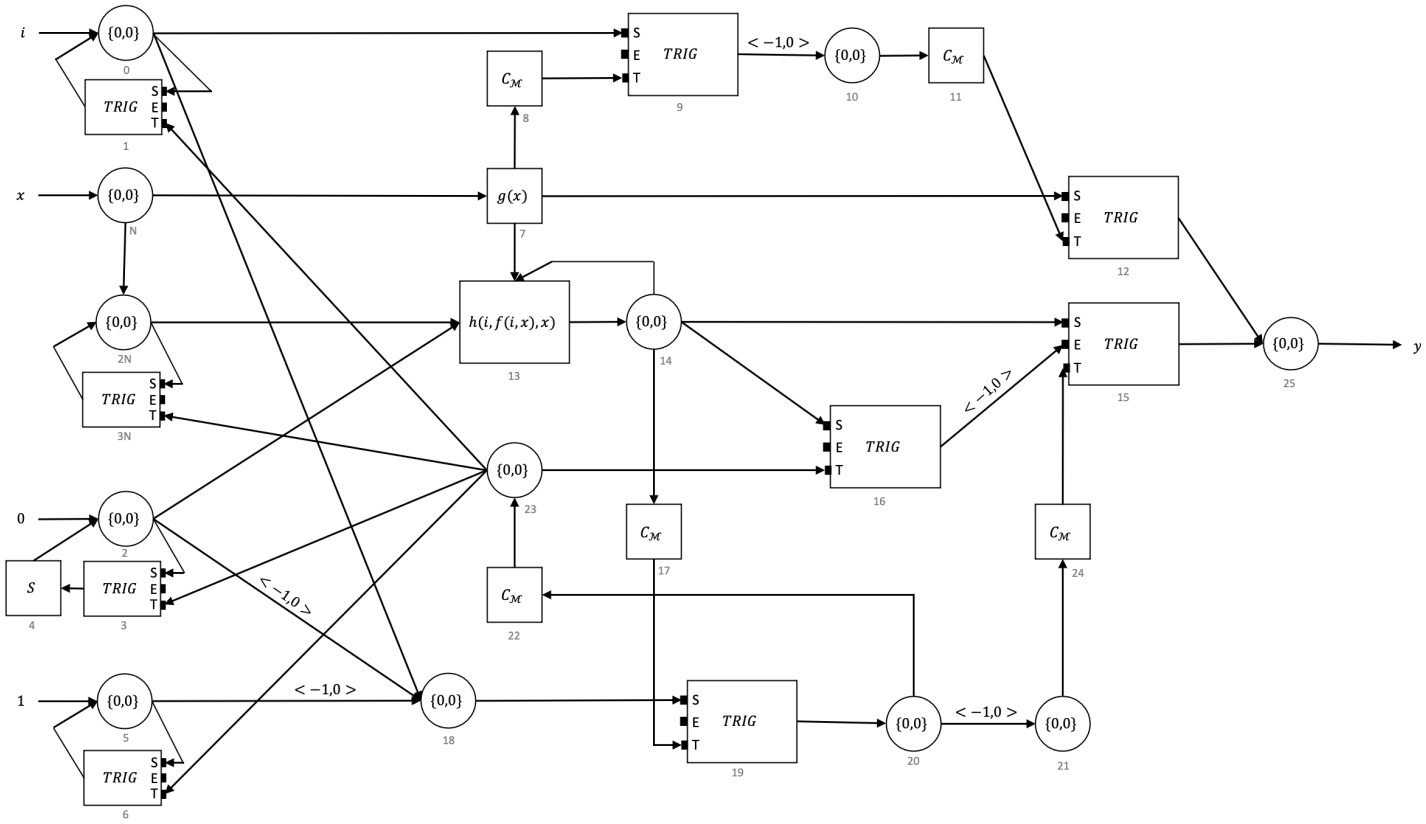


Figure 7: Neuromorphic circuit for the primitive recursion operator. All synapses are $\langle 1, 0 \rangle$ unless explicitly stated. References to all synapses can be inferred from the references of their pre-synaptic and post-synaptic circuit components. The input x in neuron `N` actually represents all of $x_1, \ldots, x_N$ such as the one shown in Figure 9. We only show it as a single input for the sake of clarity in the figure.

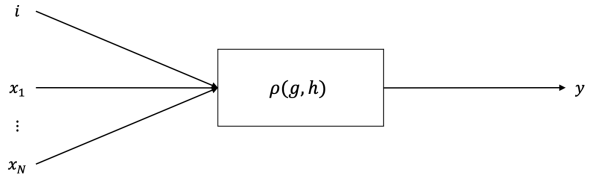


Figure 8: Abstraction for the primitive recursion operator.

in 9 is used to erase the value stored in `10`. The trigger circuits 1 and `N + 1` through `3N - 1` initiate the next iteration. The outgoing value from 1 is incremented by the successor function 2 for the next iteration. The other branch coming out of neuron 4 is not activated as neuron 5 does not spike.

If f returns zero, then neuron 5 spikes and sends this value to the constant function 6. It sends $\mathcal{M}$ and triggers `10`, which returns the value stored in it. The value returned by `10` is also the output y of the entire circuit, and equals the smallest natural number for which f returns 0. In this case, the branch 4-7-8-9-10 is also active. However, `10` still returns the correct output before its stored value is erased because the signal coming from the branch 4-5-6 reaches `10` first in initiates return of its stored value.

4.7 Computational Complexity

We compute the computational complexities of all the neuromorphic circuits presented in this paper based on the framework proposed by [14]. In this framework, the space complexity of a neuromorphic circuit is determined by counting the number of neurons and synapses. The time complexity has two parts: (1) Setup Time: The asymptotic time taken to setup the neuromorphic circuit, usually determined by counting the number of neurons and synapses; and (2) Run Time: The asymptotic time taken to execute the neuromorphic circuit, determined by counting the number of synapses in the longest path going from an input neuron to an output neuron. Since the constant function (Figure 2a) and successor function (Figure 2c) use a constant number of neurons and synapses regardless of the input, their space complexity and setup time are both $O(1)$. Furthermore, their run time is also $O(1)$ because the number of synapses in the longest path going from an input neuron to an output neuron is 1. The space complexity and setup time of the projection function in Figure 3 are $O(N)$ because we use $O(N)$ neurons and synapses in the circuit. Its run time however is $O(1)$ because there are a constant number of synapses going from the input neuron `0` to the output neuron `6N + 2`. The space and time complexities of the composition operator, primitive recursion

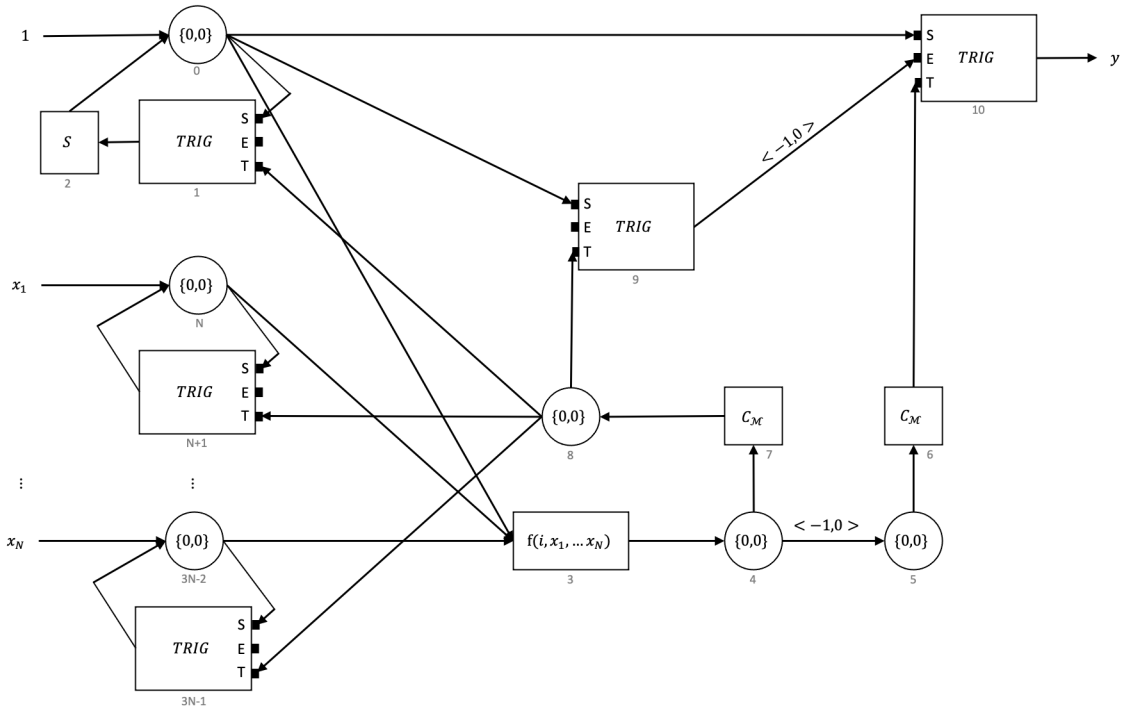


Figure 9: Neuromorphic circuit for the minimization operator. All synapses are $\langle 1, 0 \rangle$ unless explicitly stated. References to all synapses can be inferred from the references of their pre-synaptic and post-synaptic circuit components.

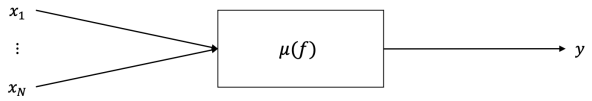


Figure 10: Abstraction for the minimization operator.

operator and minimization operator in Figures 5a, 7 and 9 respectively would depend on the complexities of the individual functions present in them, for instance, $g_1, \ldots, g_M$ in the composition operator, $g(x)$ and $h(i, f(i, x), x)$ in recursion operator, and $f(i, x)$ in the minimization operator. The computational complexity of the trigger mechanism as shown in Figure 6a are all $O(1)$.

4.8 Discussion

We show in this paper that the model of neuromorphic computing defined in Section 3 is at least as strong as the Turing machine, therefore Turing-complete. Furthermore, since our model is simple, any neuromorphic implementation that can at least realize this model would also be Turing-complete. The neuromorphic functionality provided by our model can potentially be leveraged to prove the Turing-completeness of other models of computation. We keep synaptic delay as a parameter in our model of computing, but never use it in the proof in Section 4. So, neuromorphic computing is Turing-complete even without the synaptic delay. However, we keep it in our model for the sake of convenience as it is used in many spatio-temporal encoding schemes as well as several neuromorphic algorithms. An implicit assumption that we make about all boxed functions in all our circuit diagrams is that they can be computed using our model of neuromorphic computing. Finally, through this paper, we wish to confirm the Turing-completeness of our model of neuromorphic computing. The objective of this paper is not to indicate any broader implications for neuromorphic computing or even computing in general.

It is well known that spiking neural networks (SNNs) are primarily used in machine learning applications and consume orders of magnitude less power as compared to CPUs and GPUs. We have shown in this paper that SNNs are not just restricted to machine learning applications, but are capable of all general-purpose computing tasks because they are Turing-complete. So, in theory, it should be possible to use a neuromorphic computer for anything that a CPU is used for. The main motivation for doing this would be to leverage the power efficiency of neuromorphic computing for general-purpose computing applications, i.e., to be able to do everything a CPU can do, yet consuming only a tiny fraction of the power. However, it remains to be seen what kind of reduction in power can be attained practically by using this approach. We believe this is an interesting line of research, especially when we are looking towards radically different computing paradigms such as quantum computing, to meet our computation needs beyond the end of Moore's law.

5 PRACTICAL CONSIDERATIONS

In defining our model of computation in Section 3, we assumed the existence of arbitrarily many neurons and synapses enabling all-to-all-connectivity. In a physical realization, we expect to have a finite number of neurons, each having a finite number of incoming and outgoing synapses. We observe a similar relation between the Turing machine and the von Neumann architecture. While the Turing machine assumes the existence of an infinite tape, the von Neumann architecture has finite memory in practice. We also assumed that a neuron could be connected to itself. If this functionality is not offered on hardware, it can be realized by using a proxy neuron that sends the signal back to the original neuron.

We have established that neuromorphic computing is capable of general-purpose computing by showing it can compute μ-recursive functions and operators. Systems of spiking neurons and synapses can also be used to implement Boolean logic functions such as AND, OR, and NOT. Though it is possible to implement computation in these ways on a neuromorphic system, it may not be the best way. There may be better ways to realize complex functionality with neurons and synapses than defining computation in the same way as it is defined for traditional computers. As we understand how to leverage the computational capabilities of neuromorphic computers to the fullest, we must also learn how to program them effectively and practically. As the notion of what computers look like continues to evolve with the end of Moore's law, many more ways of defining how computation is performed could emerge.

Though these proofs and circuits may be useful to implement general computation on neuromorphic computing systems, the goal of this work is not meant to provide instructions to that effect. This work is meant to demonstrate rigorously and directly that neuromorphic systems are capable of μ-recursion. We believe that proofs such as these can provide insight into how to program neuromorphic systems moving forward.

6 CONCLUSION

We propose a simple model of neuromorphic computing and prove that it is Turing-complete. With this proof, we establish that neuromorphic systems are capable of general-purpose computing, and can operate as independent processors, not just as a co-processor in a larger system that is driven by a traditional CPU. Compounding the general-purpose computability of neuromorphic computers with their extremely low power nature, there may be an opportunity to implement increasingly large amounts of computation on a neuromorphic computer—this will result in significant energy savings. With the explosion of computing devices from the cloud and high performance computing, all the way to smart phones and wearable technologies, even small power savings that can be obtained through implementation of common algorithms on more efficient hardware can have a significant impact on the carbon footprint of computing.

ACKNOWLEDGMENTS

This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan). This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This work was funded in part by the DOE Office of Science, High-energy Physics Quantised program. This work was funded in part by the DOE Office of Science, Advanced Scientific Computing Research (ASCR) program.

REFERENCES

[1] Larry F Abbott. 1999. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin* 50, 5-6 (1999), 303–304.
[2] James B Aimone, Kathleen E Hamilton, Susan Mniszewski, Leah Reeder, Catherine D Schuman, and William M Severa. 2018. Non-neural network applications for spiking neuromorphic hardware. In *Proceedings of the Third International Workshop on Post Moores Era Supercomputing*. 24–26.
[3] James B Aimone, Yang Ho, Ojas Parekh, Cynthia A Phillips, Ali Pinar, William Severa, and Yipu Wang. 2020. Provable Neuromorphic Advantages for Computing Shortest Paths. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*. 497–499.
[4] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.
[5] Felix Christian Bauer, Dylan Richard Muir, and Giacomo Indiveri. 2019. Real-time ultra-low power ECG anomaly detection using an event-driven neuromorphic processor. *IEEE transactions on biomedical circuits and systems* 13, 6 (2019), 1575–1582.
[6] Ethan Bernstein and Umesh Vazirani. 1997. Quantum complexity theory. *SIAM Journal on computing* 26, 5 (1997), 1411–1473.
[7] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. 2019. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*. 1–8.
[8] Jérémie Cabessa. 2019. Turing complete neural computation based on synaptic plasticity. *PloS one* 14, 10 (2019), e0223451.
[9] Jeremie Cabessa and Hava T Siegelmann. 2014. The super-Turing computational power of plastic recurrent neural networks. *International journal of neural systems* 24, 08 (2014), 1450029.
[10] Alonzo Church. 1936. An unsolvable problem of elementary number theory. *American journal of mathematics* 58, 2 (1936), 345–363.
[11] Alonzo Church. 1940. A formulation of the simple theory of types. *The journal of symbolic logic* 5, 2 (1940), 56–68.
[12] Alonzo Church et al. 1936. A note on the Entscheidungsproblem. *J. Symb. Log.* 1, 1 (1936), 40–41.
[13] Prasanna Date, Christopher D Carothers, James A Hendler, and Malik Magdon-Ismail. 2018. Efficient classification of supercomputer failures using neuromorphic computing. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 242–249.
[14] Prasanna Date, Bill Kay, Catherine Schuman, Robert Patton, and Thomas Potok. 2021. Computational Complexity of Neuromorphic Algorithms. In *International Conference on Neuromorphic Systems 2021*. 1–7.
[15] Prasanna Date, Robert Patton, Catherine Schuman, and Thomas Potok. 2019. Efficiently embedding QUBO problems on adiabatic quantum computers. *Quantum Information Processing* 18, 4 (2019), 1–31.
[16] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.
[17] Gabriel A Fonseca Guerra and Steve B Furber. 2017. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in neuroscience* 11 (2017), 714.
[18] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. 2014. The spinnaker project. *Proc. IEEE* 102, 5 (2014), 652–665.
[19] Marc-Oliver Gewaltig and Markus Diesmann. 2007. Nest (neural simulation tool). *Scholarpedia* 2, 4 (2007), 1430.
[20] Kurt Gödel. 1934. *On undecidable propositions of formal mathematics systems*. Institute for Advanced Study.

[21] Seonggil Ham, Minji Kang, Seonghoon Jang, Jingon Jang, Sanghyeon Choi, Tae-Wook Kim, and Gunuk Wang. 2020. One-dimensional organic artificial multisynapses enabling electronic textile neural network for wearable neuromorphic applications. *Science Advances* 6, 28 (2020), eaba1178.

[22] Kathleen Hamilton, Prasanna Date, Bill Kay, and Catherine Schuman D. 2020. Modeling epidemic spread with spike-based models. In *International Conference on Neuromorphic Systems 2020*. 1–5.

[23] Kathleen Hamilton, Tiffany Mintz, Prasanna Date, and Catherine D Schuman. 2020. Spike-based graph centrality measures. In *International Conference on Neuromorphic Systems 2020*. 1–8.

[24] Bill Kay, Prasanna Date, and Catherine Schuman. 2020. Neuromorphic graph algorithms: Extracting longest shortest paths and minimum spanning trees. In *Proceedings of the Neuro-inspired Computational Elements Workshop*. 1–6.

[25] Daniyal Kazempour. 2015. Seminar High Performance Computers-Current trends and developments Winter Term 2015/2016 Got Brain (s)? A Review and Critical Assessment on Neuromorphic Computing. (2015).

[26] Mikhail Kiselev. 2016. Rate coding vs. temporal coding-is optimum between?. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1355–1359.

[27] Donald E Knuth. 1976. Big omicron and big omega and big theta. *ACM Sigact News* 8, 2 (1976), 18–24.

[28] Johan Kwisthout and Nils Donselaar. 2020. On the computational power and complexity of Spiking Neural Networks. In *Proceedings of the Neuro-inspired Computational Elements Workshop*. 1–7.

[29] Louis Lapique. 1907. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Patholgy* 9 (1907), 620–635.

[30] Qing-Xuan Li, Tian-Yu Wang, Xiao-Lin Wang, Lin Chen, Hao Zhu, Xiao-Han Wu, Qing-Qing Sun, and David Wei Zhang. 2020. Flexible organic field-effect transistor arrays for wearable neuromorphic device applications. *Nanoscale* 12, 45 (2020), 23150–23158.

[31] Xiaoxiao Liu, Mengjie Mao, Hai Li, Yiran Chen, Hao Jiang, J Joshua Yang, Qing Wu, and Mark Barnell. 2014. A heterogeneous computing system with memristor-based neuromorphic accelerators. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–6.

[32] Wolfgang Maass. 1996. Lower bounds for the computational power of networks of spiking neurons. *Neural computation* 8, 1 (1996), 1–40.

[33] Carver Mead. 1990. Neuromorphic electronic systems. *Proc. IEEE* 78, 10 (1990), 1629–1636.

[34] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.

[35] Moritz B Milde, Hermann Blum, Alexander Dietmüller, Dora Sumislawska, Jörg Conradt, Giacomo Indiveri, and Yulia Sandamirskaya. 2017. Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Frontiers in neurorobotics* 11 (2017), 28.

[36] J Parker Mitchell, Grant Bruer, Mark E Dean, James S Plank, Garrett S Rose, and Catherine D Schuman. 2017. NeoN: Neuromorphic control for autonomous robotic navigation. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 136–142.

[37] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[38] Zihan Pan, Jibin Wu, Malu Zhang, Haizhou Li, and Yansong Chua. 2019. Neural population coding for effective temporal classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[39] Kicheol Park, Yena Lee, Jiman Hong, Jae-Hoon An, and Bongjae Kim. 2020. Selecting a Proper Neuromorphic Platform for the Intelligent IoT. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. 255–260.

[40] Oliver Rhodes. 2020. Brain-inspired computing boosted by new concept of completeness.

[41] Johannes Schemmel, Daniel Briiderle, Andreas Griibl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. 2010. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 1947–1950.

[42] Catherine D Schuman, Kathleen Hamilton, Tiffany Mintz, Md Musabbir Adnan, Bon Woong Ku, Sung-Kyu Lim, and Garrett S Rose. 2019. Shortest path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*. 1–6.

[43] Catherine D Schuman, J Parker Mitchell, Robert M Patton, Thomas E Potok, and James S Plank. 2020. Evolutionary optimization for neuromorphic systems. In *Proceedings of the Neuro-inspired Computational Elements Workshop*. 1–9.

[44] Catherine D Schuman, James S Plank, Grant Bruer, and Jeremy Anantharaj. 2019. Non-traditional input encoding schemes for spiking neuromorphic systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.

[45] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. 2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963* (2017).

[46] John M Shalf and Robert Leland. 2015. Computing beyond moore's law. *Computer* 48, 12 (2015), 14–23.

[47] Hava T Siegelmann. 2003. Neural and super-Turing computing. *Minds and Machines* 13, 1 (2003), 103–114.

[48] J Darby Smith, William Severa, Aaron J Hill, Leah Reeder, Brian Franke, Richard B Lehoucq, Ojas D Parekh, and James B Aimone. 2020. Solving a steady-state PDE using spiking networks and neuromorphic hardware. In *International Conference on Neuromorphic Systems 2020*. 1–8.

[49] Peter Smith. 2013. *An introduction to Gödel's theorems*. Cambridge University Press.

[50] Paul M Solomon. 2019. Analog neuromorphic computing using programmable resistor arrays. *Solid-State Electronics* 155 (2019), 82–92.

[51] Andrew Steane. 1998. Quantum computing. *Reports on Progress in Physics* 61, 2 (1998), 117.

[52] Alan M Turing. 1937. Computability and λ-definability. *The Journal of Symbolic Logic* 2, 4 (1937), 153–163.

[53] Alan M Turing. 2009. Computing machinery and intelligence. In *Parsing the turing test*. Springer, 23–65.

[54] Jeffrey S Vetter, Erik P DeBenedictis, and Thomas M Conte. 2017. Architectures for the post-Moore era. *IEEE Annals of the History of Computing* 37, 04 (2017), 6–8.

[55] John Von Neumann. 1993. First Draft of a Report on the EDVAC. *IEEE Annals of the History of Computing* 15, 4 (1993), 27–75.

[56] Chris Yakopcic, Nayim Rahman, Tanvir Atahary, Tarek M Taha, and Scott Douglass. 2020. Solving constraint satisfaction problems using the loihi spiking neuromorphic processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1079–1084.

[57] Youhui Zhang, Peng Qu, Yu Ji, Weihao Zhang, Guangrong Gao, Guanrui Wang, Sen Song, Guoqi Li, Wenguang Chen, Weimin Zheng, et al. 2020. A system hierarchy for brain-inspired computing. *Nature* 586, 7829 (2020), 378–384.