Turing Computation with Neural Networks Composed of Synfire Rings

Jérémie Cabessa[1,2]

[1]Laboratory of Mathematical Economics and Applied Microeconomics (LEMMA)
University Paris 2 – Panthéon-Assas, 75006 Paris, France
jeremie.cabessa@u-paris2.fr

[2]Institute of Computer Science of the Czech Academy of Sciences
18207 Prague 8, Czech Republic

***Abstract*—Synfire rings are fundamental neural circuits capable of conveying self-sustained activities in a robust and temporally precise manner. We propose a Turing-complete paradigm for neural computation based on synfire rings. More specifically, we provide an algorithmic procedure which, for any fixed-space Turing machine, builds a corresponding Boolean neural network composed of synfire rings capable of simulating it. As a consequence, any fixed-space Turing machine with tapes of length N can be simulated in linear time by some Boolean neural network composed of $O(N)$ rings and cells. The construction can naturally be extended to general Turing machines. Therefore, any Turing machine can be simulated in linear time by some Boolean neural network composed of infinitely many synfire rings. The linear time simulation relies on the possibility to mimic the behavior of the machines. In the long term, these results might contribute to the realization of biological neural computers.**

I. Introduction

In theoretical neural computation, the computational capabilities of diverse models of neural networks have been shown to range from the finite state automaton degree, up to the Turing or even to the super-Turing level [1–6]. In short, (finite) Boolean neural networks are computationally equivalent to finite state automata, rational-weighted neural nets are Turing complete, and real-weighted and evolving neural networks are super-Turing. But the neural networks involved in these results are generally far from the biological reality.

In machine learning, an augmented neural network model that can interact with a read and write external memory – following the behavior of a Turing machine – has been proposed [7]. In a more biological context, several bio-inspired Turing-complete neural network models have been considered [8–10].

Synfire chains are feedforward neural circuits where every layer is connected to the next by means of convergent/divergent excitatory synapses [11–15]. According to this architecture, the neurons of each layer tend to fire simultaneously, and the firing activity can propagate through the successive layers in a synchronized manner. Hence, these circuits are able to convey repeated complex spatio-temporal patterns of discharges in a robust and highly temporally precise way. *Synfire rings* are looping synfire chains [16–18]. As an additional dynamical feature, the ring shape enables the emergence of self-sustained activities, which correspond to attractor dynamics.

Synfire chains and rings have been argued to be crucially involved in the processing and coding of information in biological neural networks. Furthermore, they have been shown to spontaneously emerge in neural networks subject to diverse kinds of synaptic plasticity mechanisms, like spike-timing-dependent plasticity (STDP) for instance [16–24].

Based on these considerations, an automaton-complete paradigm for neural computation based on synfire rings has been proposed [25–27]. Here, we extend these results towards Turing computation. More specifically, we describe an algorithmic procedure which, for any fixed-space Turing machine, builds a corresponding Boolean neural network composed of synfire rings capable of simulating it. As a consequence, any fixed-space Turing machine with tapes of length N can be simulated in linear time by some Boolean neural network composed of $O(N)$ rings and cells. The construction can naturally be extended to general Turing machines. Therefore, any Turing machine can be simulated in linear time by some Boolean recurrent neural network composed of infinitely many synfire rings. The linear time simulation relies on the possibility to mimic the behavior of the machines. In the long term, these results might contribute to the realization of biological neural computers.

II. Boolean Neural Networks and Turing Machines

A. Boolean neural networks

A *Boolean recurrent neural network (BRNN)* is a network composed of binary neurons related together in a general architecture [1]. Formally, a BRNN is a tuple $\mathcal{N} = (\mathrm{U}, \mathrm{X}, \mathbf{A}, \mathbf{B}, \mathbf{c})$, where

- $\mathrm{U} = \{u_j : j = 1, \ldots, M\}$ is the set of *input cells*;
- $\mathrm{X} = \{x_j : j = 1, \ldots, N\}$ is the set of *internal cells*;
- $\mathbf{A} \in \mathbb{Q}^{N \times N}$ is the *internal weight matrix*, where $\mathbf{A}_{ij}$ is the synaptic weight from x_j to x_i;
- $\mathbf{B} \in \mathbb{Q}^{N \times M}$ is the *input weight matrix*, where $\mathbf{B}_{ij}$ is the synaptic weight from u_j to x_i;
- $\mathbf{c} \in \mathbb{Q}^N$ the *bias vector*, where $\mathbf{c}_i$ is the bias of x_i.

The *input* and the *state* of network $\mathcal{N}$ at time t are denoted by $\mathbf{u}(t) = (u_1(t), \ldots, u_M(t))^T \in \mathbb{B}^M$ and $\mathbf{x}(t) = (x_1(t), \ldots, x_N(t))^T \in \mathbb{B}^N$, respectively, for $t = 1, 2, \ldots$. The dynamics of $\mathcal{N}$ is given by the following equation

$$\mathbf{x}(t+1) = \theta\left(\mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t) + \mathbf{c}\right) \tag{1}$$

where $\mathbf{x}(0) = \mathbf{0}$ and θ is the componentwise *hard-threshold activation function* defined by $\theta(x) = 0$ if $x < 1$ and $\theta(x) = 1$ if $x \geq 1$.

B. Turing Machines

A *k-tape Turing machine (k-tape TM)* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

- Q is the set of *states*;
- $\Sigma = \{0, 1\}$ is the *input alphabet* not containing the *blank* symbol b;
- $\Gamma = \Sigma \cup \{b\}$ is the *tape alphabet*;
- $\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$ is the *transition function*;
- $q_0, q_{acc}, q_{rej} \in Q$ are the *initial*, *accepting* and *rejecting* states, respectively.

A transition $\delta(q, a_1, \ldots, a_k) = (q', a'_1, \ldots, a'_k, d_1, \ldots, d_k)$ (also denoted as $(q, a_1, \ldots, a_k) \overset{a'_1,\ldots,a'_k,d_1,\ldots,d_k}{\longrightarrow} (q', \ldots)$) means that if the machine $\mathcal{M}$ is in state q and reads symbols $a_1, \ldots, a_k$ with its k heads, then it will switch to state q', overwrite symbol a_i by a'_i, and move its heads to the directions $d_i \in \{L, R, S\}$, for all $i = 1, \ldots, k$ (L, R, S stand for 'left', 'right', 'stay', respectively).

A *configuration* of $\mathcal{M}$ is an instantaneous description of the state, positions of the heads, and contents of the tapes of the machine. It is denoted by a tuple of the form $C = (q, u_1 \# v_1, \ldots, u_k \# v_k)$, where $q \in Q$, $u_i \in \Gamma^*$, $v_i \in \Gamma^\omega$, and $\# \notin \Gamma$, for all $1 \leq i \leq k$. This notation stands for the fact that $\mathcal{M}$ is in state q, and for each $i = 1, \ldots, k$, the infinite word $u_i v_i$ is written on the i-th tape and the head is scanning the first letter of v_i.[1]

Given some input string $w = a_1 \cdots a_p \in \Sigma^*$, the machine $\mathcal{M}$ starts its computation in state q_0, with input w being written on its first tape, all other tapes being empty (blank symbols everywhere), and all heads located at the leftmost positions. This situation corresponds to the configuration $C_0^{\mathcal{M}} = (q_0, \epsilon \# w b^\omega, \epsilon \# b^\omega, \ldots, \epsilon \# b^\omega)$, where ϵ is the empty word. The *computation* of $\mathcal{M}$ over w is the sequence of configurations $\mathcal{C}^{\mathcal{M}} = (C_0^{\mathcal{M}}, C_1^{\mathcal{M}}, C_2^{\mathcal{M}}, \ldots)$ induced by the transition function δ while reading input w. The input $w \in \Sigma^*$ is said to be *accepted* (resp. *rejected*) by $\mathcal{M}$ iff the machine eventually reaches the state q_{acc} (resp. q_{rej}). Without loss of generality, we focus on Turing machines that never stay in a same state during their computations.

A Turing machine can be represented as a graph: the nodes and edges of the graph represent the states and transitions of the machine, respectively. As an example, the 2-tape Turing machine of Figure 1 decides the non regular and non context-free langage $L = \{0^n 1^n 0^n : n \geq 0\}$, i.e., the set of words that begin with a certain number of 0's, continue with the same number of 1's, and end up with the same number of 0's again.[2] The machine is designed in such a way that it never stays in a same state during its computation. The computation over input $w = 000111000$ is given by the following sequence of transitions.

$$\begin{aligned}(q_0, 0, b) &\overset{0b,SS}{\longrightarrow} (q_1, 0, b) \overset{00,RR}{\longrightarrow} (q'_1, 0, b) \overset{00,RR}{\longrightarrow} (q_1, 0, b)\\ &\overset{00,RR}{\longrightarrow} (q'_1, 1, b) \overset{1b,SL}{\longrightarrow} (q_2, 1, 0) \overset{11,RL}{\longrightarrow} (q'_2, 1, 0)\\ &\overset{11,RL}{\longrightarrow} (q_2, 1, 0) \overset{11,RL}{\longrightarrow} (q'_2, 0, 1) \overset{00,RR}{\longrightarrow} (q_3, 0, 1)\\ &\overset{00,RR}{\longrightarrow} (q'_3, 0, 1) \overset{00,RR}{\longrightarrow} (q_3, b, b) \overset{bb,SS}{\longrightarrow} (q_{acc}, b, b).\end{aligned} \tag{2}$$

In this work, a *fixed-space Turing machine* is a k-tape TM whose every tape has length $N > 0$.[3] Note that a fixed-space TM can generate at most $|Q| \cdot 3^{k \cdot N} \cdot k \cdot N = O(N \cdot 3^{k \cdot N})$ different configurations. Accordingly, the machine can be simulated by some finite state automaton of size $O(N \cdot 3^{k \cdot N})$, if each configuration of the former is represented by a computational state of the latter. Following to the non-optimal construction from Minsky [3], the automaton can then also be simulated by some Boolean neural networks containing $O(N \cdot 3^{k \cdot N})$ cells.[4] Here, we show that any fixed-space TM can be simulated in linear time by some Boolean neural networks containing $O(N)$ synfire rings and cells only.

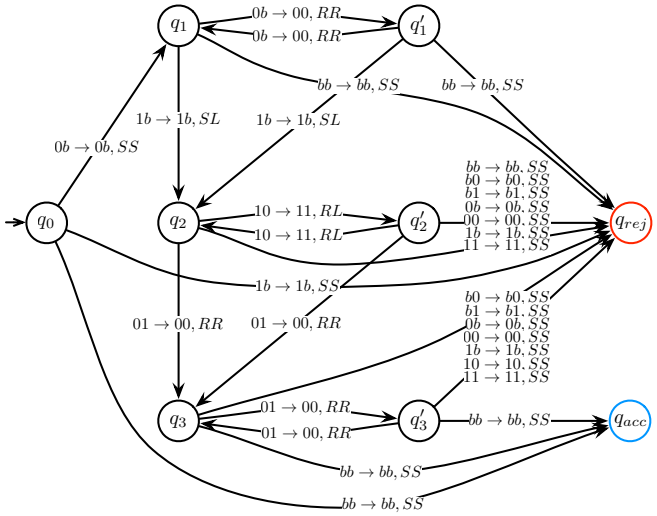

Fig. 1: Graph representation of a 2-tape Turing machine deciding the language $\{0^n 1^n 0^n : n \geq 0\}$. An edge from q to q' labelled by $a_1 a_2 \to a'_1 a'_2, d_1 d_2$ represents the transition $\delta(q, a_1, a_2) = (q', a'_1, a'_2, d_1, d_2)$. The first set of 7 labels is associated to both edges $q_2 \to q_{rej}$ and $q'_2 \to q_{rej}$ and the second set of 7 labels to the edges $q_3 \to q_{rej}$ and $q'_3 \to q_{rej}$.

[1] We recall that Γ^* and Γ^ω denote the sets of finite and infinite words over Γ, respectively.

[2] Regular and context-free languages are the languages recognized by finite state automata and pushdown automata, respectively.

[3] A fixed-space TM should not be confused with a bounded-space TM. In general, a bounded-space TM refers to a TM which, for every input of length n, is allowed to use at most $f(n)$ cells of its tape for its computation (for some function $f : \mathbb{N} \to \mathbb{N}$). In this case, there is no notion of fixed-size tape.

[4] The Boolean neural network of optimal size would contain $\Theta(\sqrt{N \cdot 3^{k \cdot N}})$ cells [28, 29].

III. Synfire Rings

Synfire chains and synfire rings are neural circuits which have been argued to be involved in the processing and coding of information in the brain. A *synfire chain* is a feedforward neural circuit where each layer is fully connected to the next one by means of excitatory connections (cf. Figure 2). A *synfire ring* is a looping synfire chain, namely, a chain whose last layer is connected back to the first one (cf. Figure 2). The synfire ring architecture exhibits the following important dynamical properties [27]:

- It allows for the emergence of a self-sustained activity.
- It is robust against synaptic failures and unreliabilities.
- It forces synchronicity among all cells of a same layer.
- It leads to the emergence of a discrete temporal structure that corresponds to the time steps at which the successive layers are activated.[5]

In this study, synfire rings will always be activated via one of their specific layers: the *activation layer*. Once activated, a ring enters into a self-sustained activity which persists as long as no inhibition is further received (cf. Figure 3). In addition, the synfire rings that we consider are coupled with an *inhibitory cell* (blue cells in Figure 4 (left)). The role of this cell is to shut down the activity of other rings by sending strong inhibition to all of their cells.

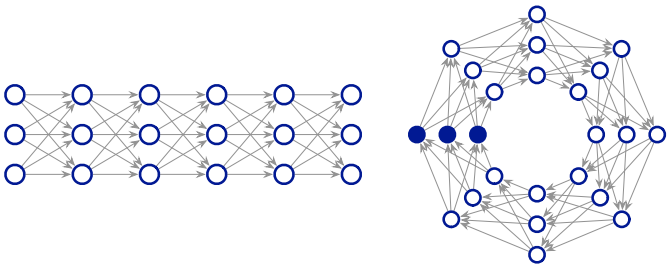

Fig. 2: A synfire chain (left) and ring (right). The filled cells form the activation layer of the ring.

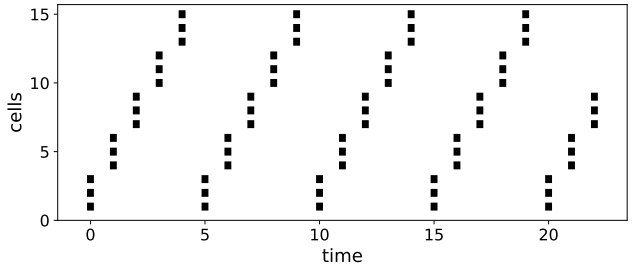

Fig. 3: Raster plot representing the self-sustained activity of a synfire ring. The ring is composed of 5 layers of 3 cells each governed by Equation (1). At consecutive time steps $t = 0, 1, 2, \ldots$, the successive layers of the synfire ring are activated (cells $1, 2, 3$, cells $4, 5, 6$, cells $7, 8, 9$, $\ldots$).

Below, four kinds of connections between cells and rings and between rings and rings are considered. These patterns are described below and illustrated in Figure 4 (left panels A, B, C, D).

(A) The *cell-to-ring one-shot excitation* (type A). When the cell c is activated, it sends excitations (orange arrows) to the activation layer and inhibitory cell of the ring R. The weights of these connections are denoted by w^{c}_{exc}.

(B) The *ring-to-ring constant excitation* (type B). As long as the ring R_1 is activated, it keeps sending excitations (orange arrows) to the activation layer and the inhibitory cell of the ring R_2. The synaptic weights are chosen such that these excitations do not suffice to activate R_2 (cf. Conditions (1)-(5) of Section IV). The weights of these connections are denoted by w^{B}_{exc}.

(C) The *ring-to-ring constant excitation/one-shot inhibition* (type C). As long as the ring R_1 is activated, it keeps sending excitations (orange arrows) to the activation layer and inhibitory cell of the ring R_2. Here again, the constant activations do not suffice to activate R_2. But if R_2 happens to be activated (by receiving other excitations), then it will send back strong inhibition (blue reverse arrows) to R_1 via its inhibitory cell, in order to shut it down. The weights of these excitatory and inhibitory connections are denoted by w^{C}_{exc} and w_{inh}.

(D) The *ring-to-ring bidirectional one-shot inhibition* (type D). When the ring R_1 (resp. R_2) is activated, it sends strong inhibition (blue arrows) to the ring R_2 (resp. R_1) via its inhibitory cell, in order to shut it down.

IV. Turing Computation

We provide an algorithmic construction which, for any fixed-space Turing machine, builds a synfire ring based Boolean neural network that simulates the behavior of the machine step by step. As a consequence, any fixed-space Turing machine with tapes of length N can be simulated in linear time by some Boolean neural network containing $O(N)$ rings and cells. Furthermore, any general Turing machine can be simulated in linear time by some Boolean recurrent neural network composed of infinitely many synfire rings.

Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ be some fixed-space k-tape TM where every tape is of length N, and let $w \in \Sigma^*$ be some input string of length p, where $p \leq N$ (the n-th letter of $wb^{N-p} \in \Gamma^N$ is denoted by $wb^{N-p}[n]$ for all $1 \leq n \leq N$). The construction of a BRNN $\mathcal{N}$ composed of $|Q| \cdot |\Gamma|^k + 8Nk = O(N)$ synfire rings that simulates the computation of $\mathcal{M}$ over w is provided in Algorithm 1 and illustrated in Figure 4 (right). The elements of this construction are described in more detail below.

Clock cells. The network $\mathcal{N}$ contains 4 *input cells* called 'start', 'tic1', 'tic2' and 'tic3' (line 1 of Algorithm 1 and purple cells of Figure 4). The computation of $\mathcal{N}$ will consist of an initial activation of 'start' followed by cyclic activations of 'tic1', 'tic2' and 'tic3'. After the initial activation of 'start', $\mathcal{N}$ will hold the encoding of the initial configuration of $\mathcal{M}$. After each activation of 'tic1', 'tic2', and 'tic3', the network $\mathcal{N}$ will simulate the fact that $\mathcal{M}$ reads the current symbols with its heads, updates its current state, and writes new symbols

[5]This property also holds when the neurons are governed by continuous time differential equations [27].

and moves its heads, respectively, according to its current transition. Hence, after each activation of 'tic3', $\mathcal{N}$ will have simulated one transition of $\mathcal{M}$.

Program rings The *program* (set of transitions) of $\mathcal{M}$ is represented by a set of $|Q|\cdot|\Gamma|^k$ *program rings* in $\mathcal{N}$ (lines 2–5 of Algorithm 1 and top set of rings in Figure 4). Each such ring encodes a specific event of the form: "$\mathcal{M}$ is in state q and is currently reading symbols $a_1, \ldots, a_k$ with its k heads". During the simulation process, such a ring will be active in $\mathcal{N}$ iff the event that it encodes is realized by $\mathcal{M}$. Program rings are connected together by means of type C connections (lines 14–15 of Algorithm 1).

Tape rings Each *tape* of lenght N of $\mathcal{M}$ is represented by 8 rows of N rings each in $\mathcal{N}$. This amounts to $8Nk$ *tape rings* (lines 6–13 of Algorithm 1 and bottom rings in Figure 4). The two last rows of rings form the *position rings*. They encode the current position of $\mathcal{M}$'s head on the tape. The n-th ring of the top (resp. bottom) row is active iff $\mathcal{M}$'s head is currently scanning the n-th square of its tape, and the last move was 'right' (resp. 'left').[6] The three middle rows form the *symbol rings*. They encode the symbols written on the tape of $\mathcal{M}$. The n-th ring of the top (resp. middle, bottom) row is active iff the n-th symbol of $\mathcal{M}$'s tape is a 1 (resp. 0, b). The three first rows form the *cache rings*. They encode the symbol currently read by the heads of $\mathcal{M}$, and will be used to simulate the state update of $\mathcal{M}$. Again, the n-th ring of the top (resp. middle, bottom) row is active iff $\mathcal{M}$'s head is currently reading the n-th symbol of its tape, and this symbol is a 1 (resp. 0, b). Note that for a k-tape $\mathcal{M}$, these 8 rows of rings are duplicated k times.

According to these considerations, in the network $\mathcal{N}$ of Figure 4, the tape rings encode the configuration of $\mathcal{M}$ where the head is scanning the 4-th cell (cf. positions rings), the string $00110bb$ is written on the tape (cf. symbol rings), and hence, the head is currently reading symbol 1 (cf. cache rings).

The connection patterns between these rings are described by their schematic representations (cf. Figure 4 (left)). Each top (resp. bottom) position ring has type C connections with its right and bottom left (resp. left and top right) neighboring rings. Each symbol ring has type D connections with others rings in the same column. Each cache ring has type D connections with all of its neighboring rings. Finally, there are vertical type B connections from position to symbol and cache rings, as well as from symbol to cache rings. The role of these connections is described below.

Initial configuration: 'start' spike. The 'start' cell sends type A connections to specific symbol and position rings (lines 22–26 of Algorithm 1). When 'start' spikes, it activates those rings that encode the initial configuration of the machine: the input is written on the first tape, all other tapes are blank, and the heads are placed to the tapes' leftmost positions.

Read symbols: 'tic1' spike. The 'tic1' cell sends type A connections to the cache rings (line 12 of Algorithm 1). When 'tic1' spikes, its activation gets combined with the type B connections coming from the position and symbol rings (line 10 of Algorithm 1). Accordingly, the cache rings encoding the symbols currently read by $\mathcal{M}$ are activated, and all previous cache rings are shut down by means of the inter-cache type D connections (line 10 of Algorithm 1).

Update state: 'tic2' spike. The 'tic2' cell sends type A connections to the program rings (line 4 of Algorithm 1). When 'tic2' spikes, its activation gets combined with the type B connections coming from the cache rings (line 17 of Algorithm 1) as well as with the type C connections coming from the currently active program ring (line 15 of Algorithm 1). Consequently, a program ring encoding the new state and symbols currently read by $\mathcal{M}$ is activated, and the previous program ring is shut down.

Write symbols and move heads: 'tic3' spike. The 'tic3' cell sends connections of type A to the symbol and position rings (line 11 of Algorithm 1). When 'tic3' spikes, its activation gets combined with the type B connections coming from the current program ring (lines 18–19 of Algorithm 1). As a consequence, the symbol and position rings encoding the new symbols written by $\mathcal{M}$ and new heads' positions of $\mathcal{M}$ are activated, and the previous symbol and position rings are shut down.

Weight conditions. In order to work properly, the processes of 'setting initial configuration', 'reading symbols', 'updating state' and 'writing symbols and moving heads' need to satisfy the following conditions (line 27 of Algorithm 1):

(1) The 'start' cell should be able to activate its targeted rings directly. Therefore,

$$w_{exc}^{\text{start}} \geq 1.$$

(2) Each cache ring receives activations from 'tic1', from one position ring and from one symbol ring. Hence, in order for the activation of this ring to be possible, the following condition must hold:

$$w_{exc}^{\text{tic1}} + 2 \cdot w_{exc}^{\text{B}} \geq 1.$$

(3) Each program ring receives activations from 'tic2', from k cache rings and from some other program rings (at most one being be active). Thus, in order for this activation to be possible, the following condition must hold:

$$w_{exc}^{\text{tic2}} + k \cdot w_{exc}^{\text{B}} + w_{exc}^{\text{C}} \geq 1.$$

(4) Each symbol ring receives activations from the 'tic3' cell, from some program ring (at most one being active), and from two position rings (at most one being active). Accordingly, the following condition must hold:

$$w_{exc}^{\text{tic3}} + w_{exc}^{\text{A}} + w_{exc}^{\text{B}} \geq 1.$$

(5) Each position ring (except to leftmost ones) receives activations from the 'tic3' cell, some program rings (but at most 1 is active), 2 other position rings (at most one being active). Therefore, the following condition must hold:

$$w_{exc}^{\text{tic3}} + w_{exc}^{\text{A}} + w_{exc}^{\text{C}} \geq 1.$$

[6] It was not possible to model the positions of the TM's head with a single row of rings.

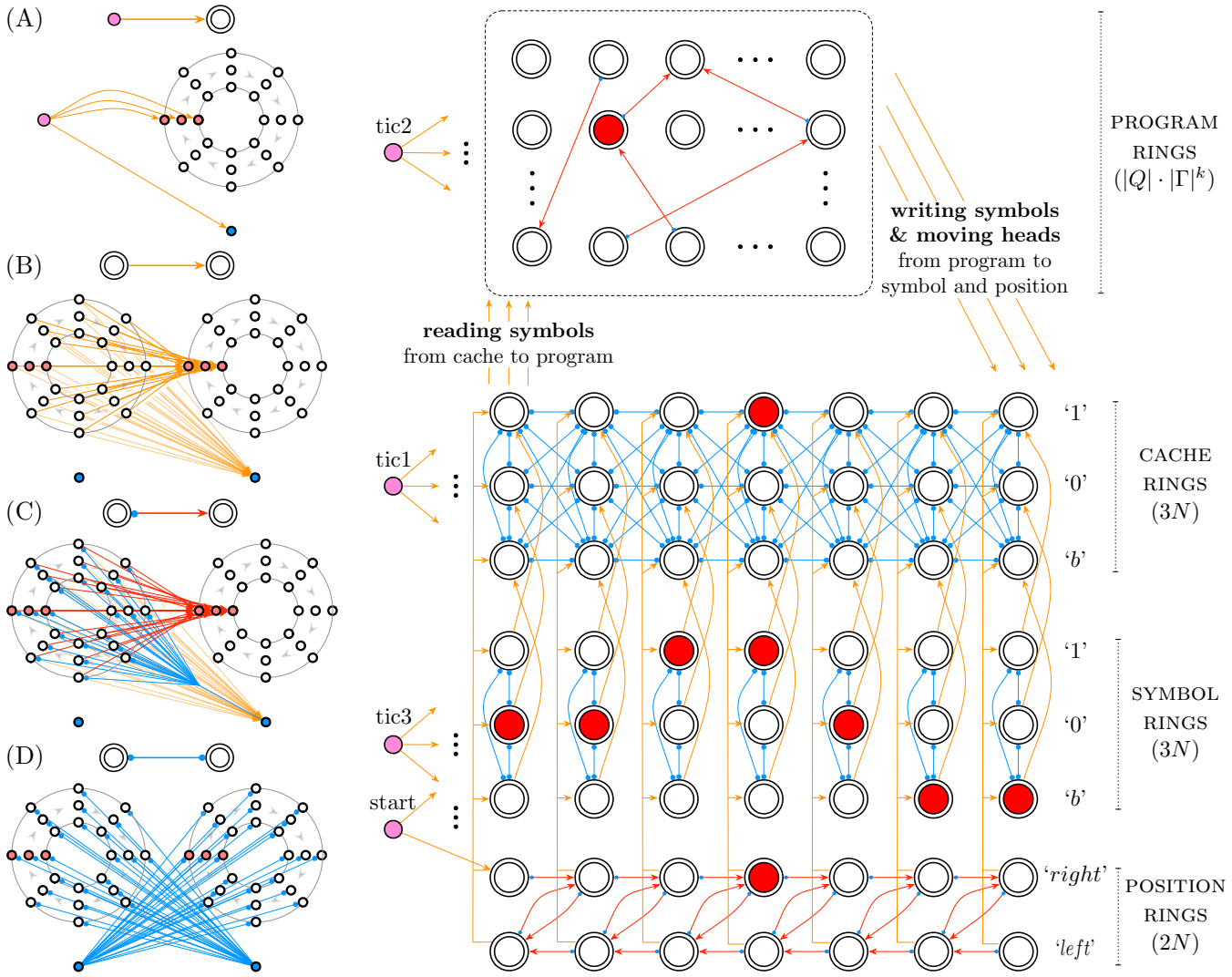

Fig. 4: Left. Connection patterns between a Boolean cell c and a synfire ring R (panel A), or between two synfire rings R_1 and R_2 (panels B, C, D). Each ring is associated with an inhibitory cell (blue cell). In each panel, a schematic representation of the connection pattern is provided on top and its detailed description given below. Red and orange arrows represent excitatory connections, while blue arrows are inhibitory ones. In panel C, for the sake of clarity, the superimposed orange and reverse blue connections are drawn so that both can be visualized clearly. **Right.** General architecture of a Boolean recurrent neural network composed of syfire rings simulating a 1-tape Turing machine. The purple nodes are the clock cells. The white and red circles represent active and quiet synfire rings, respectively. The cell-to-ring and ring-to-ring connection patterns are those described in the left panel (schematic representations).

In addition, in each of the above condition, it is further required that if at least one term of the sum is null, then the sum must be strictly less than 1. This last condition ensures that any missing activation will prevent the targeted ring from being activated.

Consider some BRNN $\mathcal{N}$ obtained by application of Algorithm 1 over some TM $\mathcal{M}$. For any state $q \in Q$, and any words $u_1, \ldots, u_k \in \Gamma^*$ and $v_1, \ldots, v_k \in \Gamma^* \cup \Gamma^\omega$, if the program, symbol, position, and cache ring activities of $\mathcal{N}$ encode the facts that $\mathcal{M}$ is in state q, that $u_i v_i$ is written on the i-th tape, and that the i-th head is scanning the first symbol of v_i, for each $i = 1, \ldots, k$, then the network $\mathcal{N}$ is said to be in the *configuration* $C = (q, u_1 \# v_1, \cdots, u_k \# v_k)$. Otherwise, the configuration of $\mathcal{N}$ is undefined.

Now, suppose that $\mathcal{N}$ is provided with the cyclic input pattern start $= 1$, tic1 $= 1$, tic2 $= 1$, tic3 $= 1$, tic1 $= 1$, tic2 $= 1$, tic3 $= 1$, $\ldots$, where the spikes of these input cells are separated by sufficiently many time steps T in order for the network to settle into stable activities (T is a constant depending on the lengths of the synfire rings). The *computation* of $\mathcal{N}$ over w is the sequence of configurations $\mathcal{C}^{\mathcal{N}} = (C_0^{\mathcal{N}}, C_1^{\mathcal{N}}, C_2^{\mathcal{N}}, \ldots)$ of $\mathcal{N}$, where $C_0^{\mathcal{N}}$ is the configuration obtained 2 time steps after the spike of 'start', and for each $t > 0$, $C_t^{\mathcal{N}}$ is the configuration obtained 2 time steps after the t-th spike of 'tic3'. We say that the network $\mathcal{N}$ *simulates* the machine $\mathcal{N}$ in linear time if for any input $w \in \Sigma^{\leq N}$, the computations $\mathcal{C}^{\mathcal{N}}$ and $\mathcal{C}^{\mathcal{M}}$ of $\mathcal{N}$ and $\mathcal{M}$ over w are the same, and each configuration $C_t^{\mathcal{N}}$ is obtained at time $\alpha \cdot t + \beta$, for some constants $\alpha, \beta > 0$ and for all $t > 0$. The following result holds:

Algorithm 1

Require: k-tape TM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ and input word $w \in \Sigma^{\leq N}$

create cells 'start', 'tic1', 'tic2', 'tic3' — *** clock cells ***
for all $(q, a_1, \ldots, a_k) \in Q \times \Gamma^k$ **do** — *** program rings ***
  create the ring $R_{(q,a_1,\ldots,a_k)}$ — program ring
  connect 'tic2' cell to program ring $R_{(q,a_1,\ldots,a_k)}$: connections of type A — tic2 to programs
end for

for all $i = 1$ to k **do** — *** k tape rings ***
  create $2N$ *position rings*: $\{R^n_{i,L} : 1 \leq n \leq N\}$, $\{R^n_{i,R} : 1 \leq n \leq N\}$ — position rings
  create $3N$ *symbol rings*: $\{R^n_{i,b} : 1 \leq n \leq N\}$, $\{R^n_{i,0} : 1 \leq n \leq N\}$, $\{R^n_{i,1} : 1 \leq n \leq N\}$ — symbol rings
  create $3N$ *cache rings*: $\{R^n_{i,Cb} : 1 \leq n \leq N\}$, $\{R^n_{i,C0} : 1 \leq n \leq N\}$, $\{R^n_{i,C1} : 1 \leq n \leq N\}$ — cache rings
  connect the position, symbol and cache rings as described in Figure 4: connections of types B, C and D — tape ring connections
  connect 'tic3' cell to each position and symbol ring: connections of type A — tic3 to positions and symbols
  connect 'tic1' cell to each cache ring: connections of type A — tic1 to caches
end for

for all transition $\delta(q, a_1, \ldots, a_k) = (q', a'_1, \ldots, a'_k, d_1, \ldots, d_k)$ **do** — *** connections ***
  connect program ring $R_{(q,a_1,\ldots,a_k)}$ to program ring $R_{(q',a'_1,\ldots,a'_k)}$: connections of type C — program to program
  for all $1 \leq n \leq N$ **do**
    connect the k cache rings $R^n_{1,Ca_1}, \ldots, R^n_{k,Ca_k}$ to program ring $R_{(q,a_1,\ldots,a_k)}$: connections of type B — caches to program
    connect program ring $R_{(q,a_1,\ldots,a_k)}$ to the k symbol rings $R^n_{1,a'_1}, \ldots, R^n_{k,a'_k}$: connections of type B — program to symbols
    connect program ring $R_{(q,a_1,\ldots,a_k)}$ to the k position rings $R^n_{1,d_1}, \ldots, R^n_{k,d_k}$: connections of type B — program to positions
  end for
end for

for all $1 \leq n \leq N$ **do** — *** initial configuration ***
  connect 'start' cell to symbol rings $R^n_{1,wb^{N-p[n]}}$: connections of type A — start to symbols: input wb^* on 1st tape rings
  connect 'start' cell to the symbol rings $R^n_{2,b}, \ldots, R^n_{k,b}$: connections of type A — start to symbols: b's on other tape rings
end for
connect 'start' cell to the position rings $R^1_{1,R}, \ldots, R^1_{k,R}$:connections of type A — start to positions: leftmost positions
set all synaptic weights in order to satisfy Conditions (1)–(5) — *** weights ***

Theorem 1.

(i) Let $\mathcal{M}$ be a fixed-space k-tape TM and $w \in \Sigma^{\leq N}$ be some input. Then, there exists some BRNN composed of $O(N)$ synfire rings and cells that simulates $\mathcal{M}$ in linear time.

(ii) Let $\mathcal{M}$ be a general k-tape TM and $w \in \Sigma^*$ be some input. Then, there exists a BRNN $\mathcal{N}$ composed of infinitely many synfire rings that simulates $\mathcal{M}$ in linear time.

Proof. (Sketch) (i) Let $\mathcal{N}$ be the BRNN composed of synfire ring provided by the application of Algorithm 1 on input $\mathcal{M}$ and w. By construction, $\mathcal{N}$ is composed of $|Q| \cdot |\Gamma|^k + 8Nk = O(N)$ synfire rings, and hence of $O(N)$ cells also. Now, let $\mathcal{C}^{\mathcal{M}} = (C^{\mathcal{M}}_t)_{t \geq 0}$ and $\mathcal{C}^{\mathcal{N}} = (C^{\mathcal{N}}_t)_{t \geq 0}$ be the finite or infinite computations of $\mathcal{M}$ and $\mathcal{N}$ over w, respectively. By induction on t, we can show that $C^{\mathcal{M}}_t = C^{\mathcal{N}}_t$, for all $t = 0, 1, 2, \ldots$. In addition, for each $t \geq 0$, the configuration $C^{\mathcal{N}}_t$ is obtained at time $3T \cdot t + 2$, where $3T$ is the constant number of time steps separating two consecutive spikes of 'tic3'.

(ii) Let $\mathcal{N}$ be the BRNN composed of synfire ring provided by a slightly modified version of Algorithm 1 where the finitely many tape rings are replaced by infinitely many ones. By the same argument as above, the finite or infinite computations of $\mathcal{M}$ and $\mathcal{N}$ are the same on every input $w \in \Sigma^*$ and $C^{\mathcal{N}}_t$ and the simulation is in linear time. □

V. Computer Simulation

To illustrate the correctness of our construction (Algorithm 1), we implemented the network composed of synfire rings that simulates the fixed-size 2-tape Turing machine of Figure 1 (with $N = 10$) over input $w = 000111000$.

The synfire rings that we consider are composed of 5 layers of 2 cells each, plus the 1 inhibitory cell, which makes 11 cells in total. The BRNN is composed of the 4 input cells, 47 program rings, and for each tape (there are 2 tapes), there are $3 \times 10 = 30$ cache rings, $3 \times 10 = 30$ symbol rings, $2 \times 10 = 20$ symbol rings. This makes a total of 4 cells and 207 rings, which amounts to 2281 cells. Moreover, according to the connection patterns described in Figure 4 and Algorithm 1, the BRNN contains 45177 synaptic connections.

The activity of the BRNN is represented in Figure 5 in the form of a raster plot. We can see that the network simulates the 2-tape TM correctly, in the sense described in Theorem 1. More specifically, the successive program rings that are activated are (top of Figure 5):

$$R_{(q_0,0,b)}, R_{(q_1,0,b)}, R_{(q'_1,0,b)}, R_{(q_1,0,b)}, R_{(q'_1,1,b)},$$
$$R_{(q_2,1,0)}, R_{(q'_2,1,0)}, R_{(q_2,1,0)}, R_{(q'_2,0,1)},$$
$$R_{(q_3,0,1)}, R_{(q'_3,0,1)}, R_{(q_3,b,b)}, R_{(q_{acc},b,b)}$$

The indices of these rings correspond precisely to the successive states and symbols read by the 2-tape TM of Figure 1 over input w, as described in Computation (2). Moreover, the activity of the position, symbol and cache rings associated to the first and second tapes of the TM encode the following facts:

- the first head performs the following successive moves: S, R, R, R, S, R, R, R, R, R, R, S.

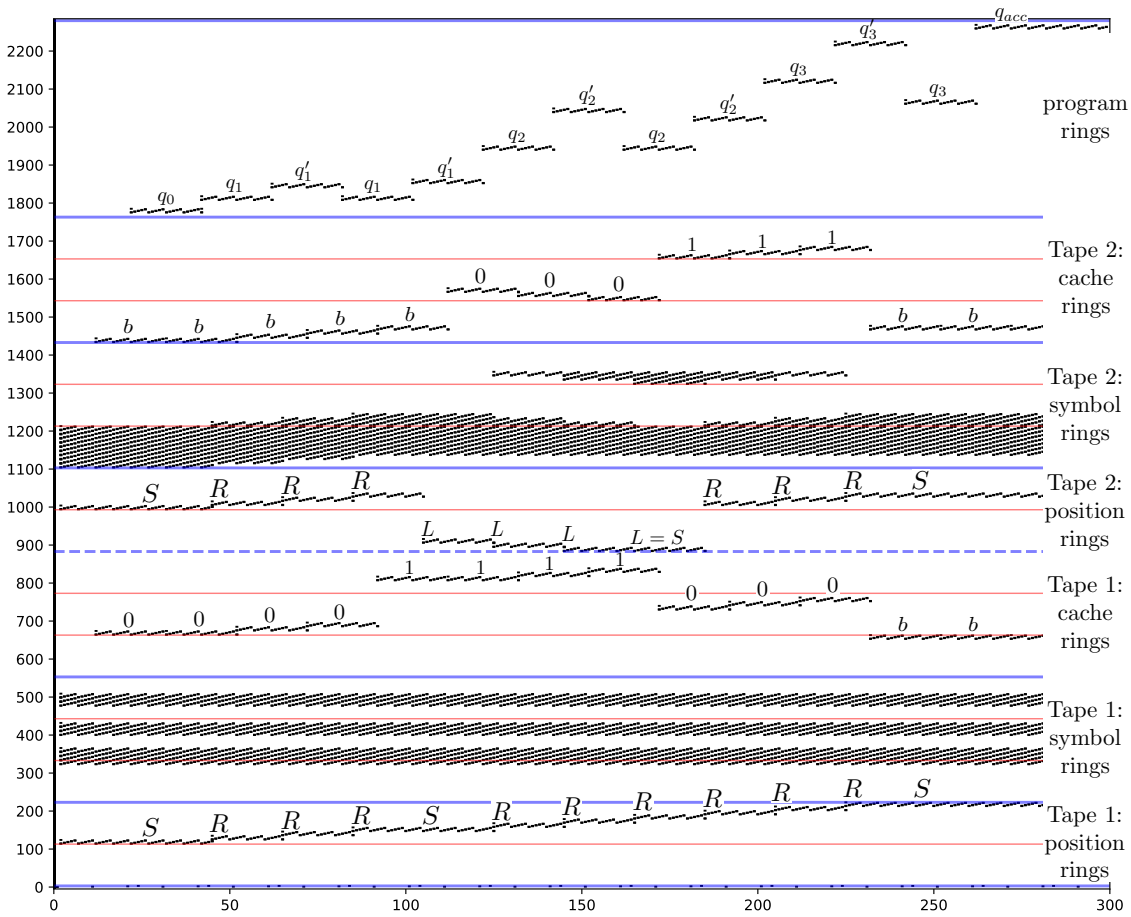

Fig. 5: Raster plot: activity of the BRNN composed of synfire rings simulating the 2-tape TM of Figure 1 over input $000111000b$. The time and the network's cells are represented on the x and y axes, respectively. Each "wave-like" horizontal trace represents the activity of a specific synfire ring (cf. Figure 3). From bottom to top, and delimited by blue lines, we have: the 4 input cells 'start', 'tic1', 'tic2' and 'tic3', the cache, symbol and positions rings associated to the first tape, the cache, symbol and positions rings associated to the second tape, and the program rings. Every time 'tic1', 'tic2' or 'tic3' spikes, the activities of the cache rings, the program rings and the symbol and position rings are modified, respectively. After each spike of 'tic3', the network enters into a new configuration. The sequence of configurations of the network form its computation. To facilitate the reading of the raster, the states, symbols and positions encoded by the activities of the program rings, cache rings and position rings are written on top of their respective traces, respectively.

- the symbols on the first tape are never rewritten, meaning that the successive contents of the tape remain as $000111000b$.
- the successive cache symbols associated to the first tape are: 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, b, b.
- the second head performs the following successive moves: S, R, R, R, L, L, L, L, R, R, R, S.
- the successive contents of the second tape are: $bbbbbbbbbb$, $bbbbbbbbbb$, $0bbbbbbbbb$, $00bbbbbbbb$, $000bbbbbbb$, $000bbbbbbb$, $001bbbbbbb$, $011bbbbbbb$, $111bbbbbbb$, $011bbbbbbb$, $001bbbbbbb$, $000bbbbbbb$, $000bbbbbbb$.
- the successive cache symbols associated to the second tape are: b, b, b, b, b, 0, 0, 0, 1, 1, 1, b, b.

These features correctly match to the successive positions, symbols read and symbols written by the TM's heads, as described in Computation (2). A movie displaying the computation of this network over input $w = 000111000$ can be downloaded here. The code to reproduce the results is available on GitHub: https://github.com/JeremCab/SynfireRings.

VI. Conclusion

We proposed a novel Turing complete paradigm for neural computation based on synfire rings. With these achievements, we do not intend to argue that brain computational processes really perform simulations of Turing machines in the way described here. Rather, our intention is to show that a neuronal paradigm for abstract computation based on sustained activities of cell assemblies – the synfire rings – is possible and potentially exploitable.

For future work, we intend to generalize the proposed results to the context of more biological neural networks, where the cells dynamics are driven by the Hodgkin-Huxley differential equations and the patterns of connection based on more realistic features.

In biology, reliable logical gates have been implemented *in vitro*, via geometrically designed neural cultures [30, 31]. The living neural networks are forced to grow on quasi-one-dimensional configurations which enable precise input-output

patterns of activity. Along these lines, similar biological implementations of our construction would lead to the realization of *biological neural computers*.

VII. Acknowledgements

This research was done with institutional support RVO: 67985807 and partially supported by the grant of the Czech Science Foundation AppNeCo No. GA22-02067S.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysic*, vol. 5, pp. 115–133, 1943.

[2] S. C. Kleene, "Representation of events in nerve nets and finite automata," in *Automata Studies*, C. Shannon and J. McCarthy, Eds. Princeton, NJ: Princeton University Press, 1956, pp. 3–41.

[3] M. L. Minsky, *Computation: finite and infinite machines*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1967.

[4] H. T. Siegelmann and E. D. Sontag, "Analog computation via neural networks," *Theor. Comput. Sci.*, vol. 131, no. 2, pp. 331–360, 1994.

[5] ——, "On the computational power of neural nets," *J. Comput. Syst. Sci.*, vol. 50, no. 1, pp. 132–150, 1995.

[6] J. Cabessa and H. T. Siegelmann, "The super-Turing computational power of plastic recurrent neural networks," *Int. J. Neural Syst.*, vol. 24, no. 8, 2014.

[7] A. Graves and al., "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[8] G. S. Carmantini, P. beim Graben, M. Desroches, and S. Rodrigues, "Turing computation with recurrent artificial neural networks," in *Proceedings of CoCo@NIPS*, T. R. Besold and al., Eds., vol. 1583. CEUR-WS.org, 2015.

[9] ——, "A modular architecture for transparent computation in recurrent neural networks," *Neural Networks*, vol. 85, pp. 85–105, 2017.

[10] J. Cabessa, "Turing complete neural computation based on synaptic plasticity," *PLOS ONE*, vol. 14, no. 10, pp. 1–34, 10 2019.

[11] M. Abeles, *Corticonics: Neuronal Circuits of the Cerebral Cortex*, 1st ed. Cambridge University Press, 1991.

[12] ——, "Time is precious," *Science*, vol. 304, no. 5670, pp. 523–524, 2004.

[13] Y. Ikegaya, G. Aaron, R. Cossart, D. Aronov, I. Lampl, D. Ferster, and R. Yuste, "Synfire chains and cortical songs: Temporal modules of cortical activity," *Science*, vol. 304, no. 5670, pp. 559–564, 2004.

[14] Z. Mainen and T. Sejnowski, "Reliability of spike timing in neocortical neurons," *Science*, vol. 268, no. 5216, pp. 1503–1506, 1995.

[15] M. Diesmann, M.-O. Gewaltig, and A. Aertsen, "Stable propagation of synchronous spiking in cortical neural networks," *Nature*, vol. 402, no. 6761, pp. 529–533, 1999.

[16] D. Horn, N. Levy, I. Meilijson, and E. Ruppin, "Distributed synchrony of spiking neurons in a hebbian cell assembly," in *Advances in Neural Information Processing Systems, NIPS 1999*, S. A. Solla and al., Eds. The MIT Press, 1999, pp. 129–135.

[17] N. Levy, D. Horn, I. Meilijson, and E. Ruppin, "Distributed synchrony in a cell assembly of spiking neurons," *Neural Networks*, vol. 14, no. 6, pp. 815–824, 2001.

[18] P. Zheng and J. Triesch, "Robust development of synfire chains from multiple plasticity mechanisms," *Front. Comput. Neurosci.*, vol. 8, no. 66, 2014.

[19] J. Hertz and A. Prügel-Bennett, "Learning synfire chains by self-organization," *Network: Computation in Neural Systems*, vol. 7, pp. 357–363, 1996.

[20] D. V. Buonomano, "A learning rule for the emergence of stable dynamics and timing in recurrent networks," *Journal of Neurophysiology*, vol. 94, no. 4, pp. 2275–2283, 2005.

[21] J. K. Jun and D. Z. Jin, "Development of neural circuitry for precise temporal sequences through spontaneous activity, axon remodeling, and synaptic plasticity," *PLOS ONE*, vol. 2, no. 8, pp. 1–17, 2007.

[22] K. Kitano, H. Câteau, and T. Fukai, "Self-organization of memory activity through spike-timing-dependent plasticity." *Neuroreport*, vol. 13, no. 6, pp. 795–798, 2002.

[23] N. Masuda and H. Kori, "Formation of feedforward networks and frequency synchrony by spike-timing-dependent plasticity," *Journal of Computational Neuroscience*, vol. 22, no. 3, pp. 327–345, 2007.

[24] R. Hosaka, O. Araki, and T. Ikeguchi, "STDP provides the substrate for igniting synfire chains by spatiotemporal input patterns," *Neural Computation*, vol. 20, no. 2, pp. 415–435, 2008.

[25] J. Cabessa and P. Masulli, "Emulation of finite state automata with networks of synfire rings," in *International Joint Conference on Neural Networks, IJCNN 2017*. IEEE, 2017, pp. 4641–4648.

[26] J. Cabessa and A. Tchaptchet, "Automata computation with hodgkin-huxley based neural networks composed of synfire rings," in *International Joint Conference on Neural Networks, IJCNN 2018*. IEEE, 2018, pp. 1–8.

[27] ——, "Automata complete computation with hodgkin-huxley neural networks composed of synfire rings," *Neural Networks*, vol. 126, pp. 312–334, 2020.

[28] B. G. Horne and D. R. Hush, "Bounds on the complexity of recurrent neural network implementations of finite state machines," *Neural Networks*, vol. 9, no. 2, pp. 243–252, 1996.

[29] P. Indyk, "Optimal simulation of automata by neural nets," in *STACS*, 1995, pp. 337–348.

[30] O. Feinerman, A. Rotem, and E. Moses, "Reliable neuronal logic devices from patterned hippocampal cultures," *Nature Physics*, vol. 4, pp. 967–973, 10 2008.

[31] F. Wolf and T. Geisel, "Neurophysics: Logic gates come to life," *Nat. Phys.*, vol. 4, no. 12, pp. 905–906, 2008.