

# AUTOMATA AND TURING COMPLETE COMPUTATION WITH SYNFiRE RING-BASED NEURAL NETWORKS

Jérémie Cabessa

Laboratoire d'économie mathématique  
Université Paris II, France

March 4, 2019, Prague, Czech Republic

# INTRODUCTION

- ▶ We recall important results about the computational capabilities of recurrent neural networks.
- ▶ We introduce a bio-inspired paradigm for neural computation based on the concept of *synfire rings*.
- ▶ We show that finite state machines can be simulated by Boolean recurrent neural networks composed of synfire rings.

# INTRODUCTION

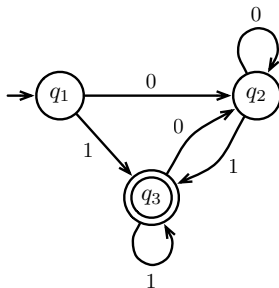
- ▶ We recall important results about the computational capabilities of recurrent neural networks.
- ▶ We introduce a bio-inspired paradigm for neural computation based on the concept of *synfire rings*.
- ▶ We show that finite state machines can be simulated by Boolean recurrent neural networks composed of synfire rings.

# INTRODUCTION

- ▶ We recall important results about the computational capabilities of recurrent neural networks.
- ▶ We introduce a bio-inspired paradigm for neural computation based on the concept of *synfire rings*.
- ▶ We show that finite state machines can be simulated by Boolean recurrent neural networks composed of synfire rings.

# FINITE STATE AUTOMATON

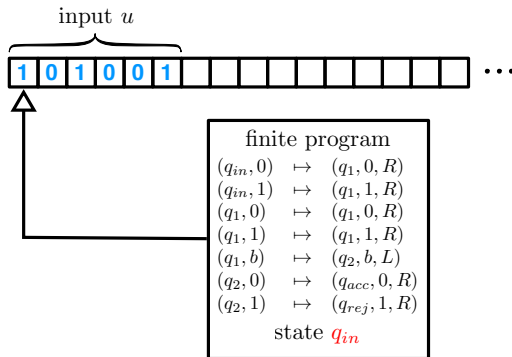
A *finite state automaton* (FSA) can be represented as a graph whose nodes and edges are the computational states and transitions between those.



- ▶ input  $u$  is *accepted* by  $\mathcal{A}$  if  $\mathcal{A}(u)$  reaches a final state
- ▶ input  $u$  is *rejected* by  $\mathcal{A}$  if  $\mathcal{A}(u)$  otherwise

# TURING MACHINE

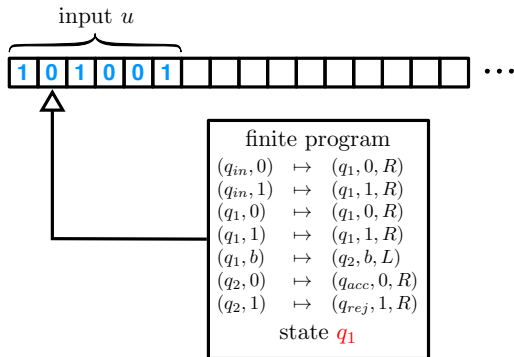
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input  $u$  is *accepted* by  $M$  if  $M(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $M$  if  $M(u)$  reaches the state  $q_{rej}$

# TURING MACHINE

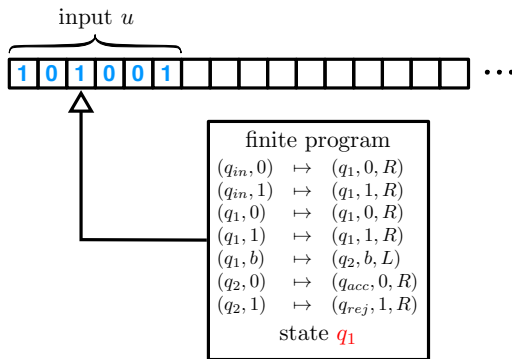
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$

# TURING MACHINE

A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.

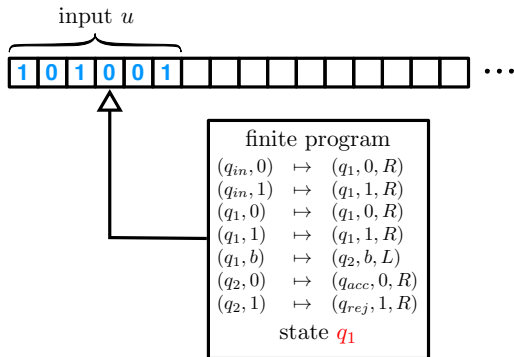


- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$



# TURING MACHINE

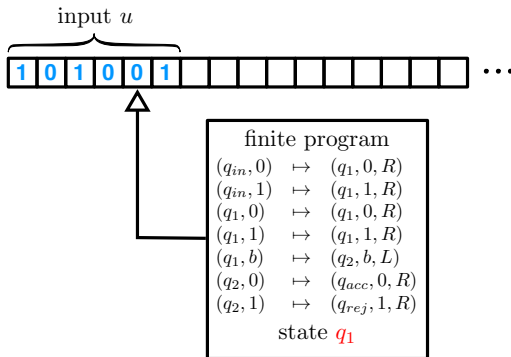
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$

# TURING MACHINE

A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.

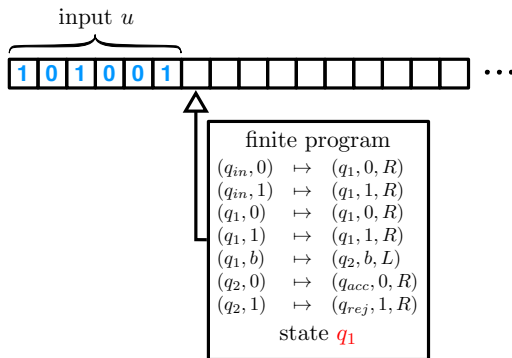


- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$



# TURING MACHINE

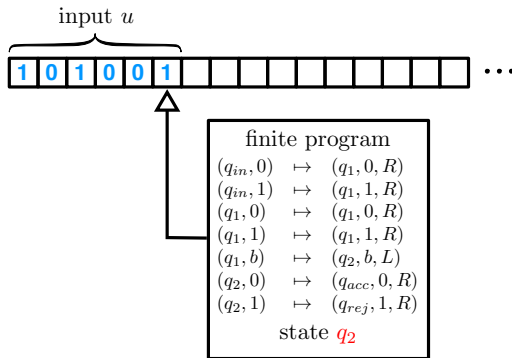
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$

# TURING MACHINE

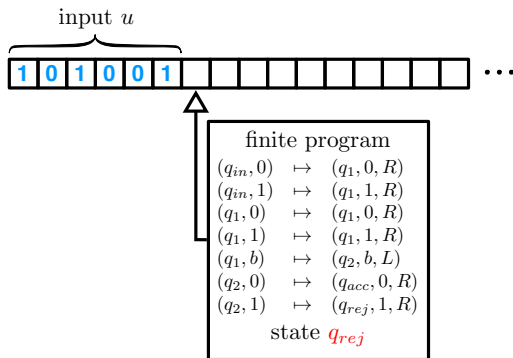
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$

# TURING MACHINE

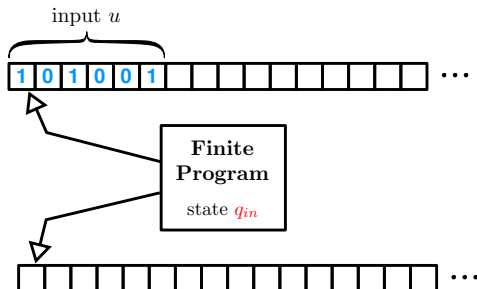
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input  $u$  is *accepted* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{acc}$
- ▶ input  $u$  is *rejected* by  $\mathcal{M}$  if  $\mathcal{M}(u)$  reaches the state  $q_{rej}$

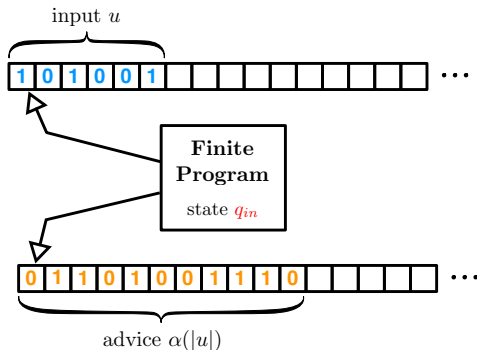
# TURING MACHINE WITH ADVICE

A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function  $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$ .



# TURING MACHINE WITH ADVICE

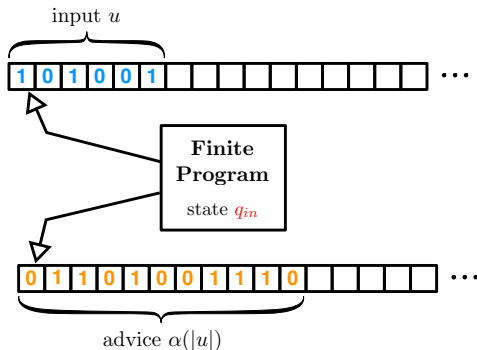
A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function  $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$ .





# TURING MACHINE WITH ADVICE

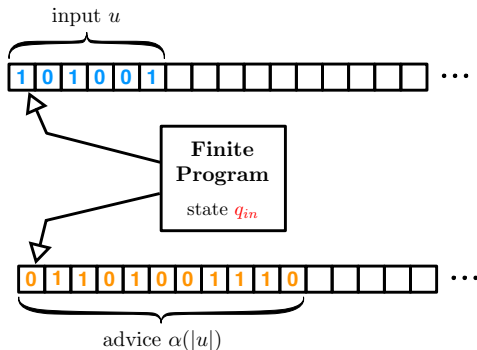
A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function  $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$ .



- ▶ The class of languages recognized in polynomial time by Turing machines with polynomial advices (TM/poly(A)) is **P/poly**.

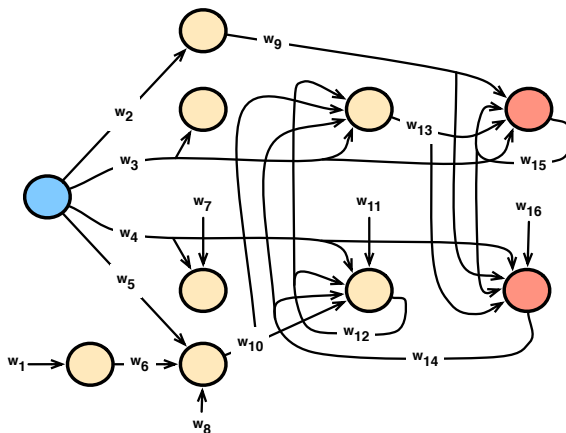
# TURING MACHINE WITH ADVICE

A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function  $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$ .

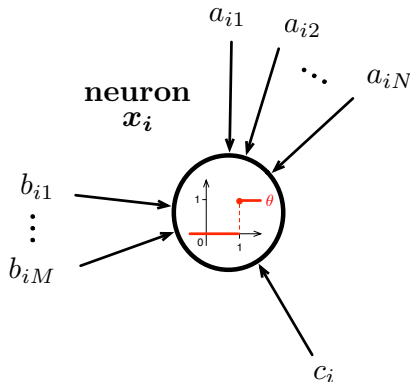


- ▶ TM/As are strictly more powerful than TMs:  $\mathbf{P/poly} \supsetneq \mathbf{P}$   
They are *super-Turing*...

# RECURRENT NEURAL NETWORK

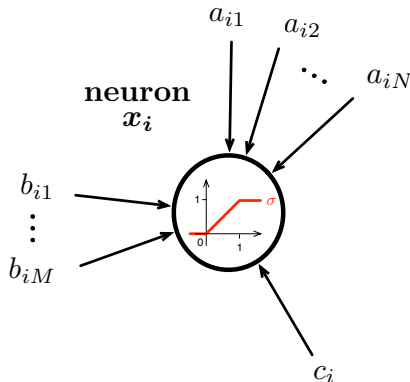


# BOOLEAN RECURRENT NEURAL NETWORKS



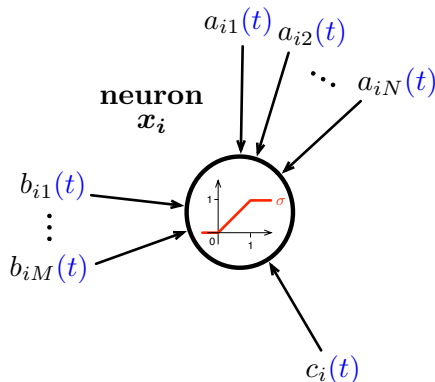
$$x_i(t+1) = \theta \left( \sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

# SIGMOID RECURRENT NEURAL NETWORKS



$$x_i(t+1) = \sigma \left( \sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

# EVOLVING RECURRENT NEURAL NETWORKS



$$x_i(t+1) = \sigma \left( \sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

# RESULTS: CLASSICAL COMPUTATION

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED EVOLVING	EVOLVING
Q	<b>FSA</b>	<b>TM</b>	<b>TM/poly(A)</b>	<b>TM/poly(A)</b>
	<b>REG</b>	<b>P</b>	<b>P/poly</b>	<b>P/poly</b>
	KI 56, Mi 67	Si & So 95	Ca & Si 11,14	Ca & Si 11,14
R	<b>FSA</b>	<b>TM/poly(A)</b>	<b>TM/poly(A)</b>	<b>TM/poly(A)</b>
	<b>REG</b>	<b>P/poly</b>	<b>P/poly</b>	<b>P/poly</b>
	KI 56, Mi 67	Si & So 94	Ca & Si 11,14	Ca & Si 11,14

# RESULTS: CLASSICAL COMPUTATION

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED EVOLVING	EVOLVING
Q	<b>FSA</b>	<b>TM</b>	TM/poly(A)	TM/poly(A)
	<b>REG</b>	<b>P</b>	P/poly	P/poly
	KI 56, Mi 67	Si & So 95	Ca & Si 11,14	Ca & Si 11,14
R	<b>FSA</b>	TM/poly(A)	TM/poly(A)	TM/poly(A)
	<b>REG</b>	P/poly	P/poly	P/poly
	KI 56, Mi 67	Si & So 94	Ca & Si 11,14	Ca & Si 11,14



# RESULTS: CLASSICAL COMPUTATION

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED EVOLVING	EVOLVING
Q	<b>FSA</b>	<b>TM</b>	TM/poly(A)	TM/poly(A)
	<b>REG</b>	<b>P</b>	P/poly	P/poly
	KI 56, Mi 67	Si & So 95	Ca & Si 11,14	Ca & Si 11,14
R	<b>FSA</b>	TM/poly(A)	TM/poly(A)	TM/poly(A)
	<b>REG</b>	<b>P/poly</b>	P/poly	P/poly
	KI 56, Mi 67	Si & So 94	Ca & Si 11,14	Ca & Si 11,14

# RESULTS: CLASSICAL COMPUTATION

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED EVOLVING	EVOLVING
Q	<b>FSA</b>	<b>TM</b>	TM/poly(A)	TM/poly(A)
	<b>REG</b>	<b>P</b>	P/poly	<b>P/poly</b>
	KI 56, Mi 67	Si & So 95	Ca & Si 11,14	Ca & Si 11,14
R	<b>FSA</b>	TM/poly(A)	TM/poly(A)	TM/poly(A)
	<b>REG</b>	<b>P/poly</b>	P/poly	<b>P/poly</b>
	KI 56, Mi 67	Si & So 94	Ca & Si 11,14	Ca & Si 11,14

# RESULTS: CLASSICAL COMPUTATION

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED	EVOLVING
Q	<b>FSA</b>	<b>TM</b>	<b>TM/poly(A)</b>	<b>TM/poly(A)</b>
	<b>REG</b>	<b>P</b>	<b>P/poly</b>	<b>P/poly</b>
	KI 56, Mi 67	Si & So 95	Ca & Si 11,14	Ca & Si 11,14
R	<b>FSA</b>	<b>TM/poly(A)</b>	<b>TM/poly(A)</b>	<b>TM/poly(A)</b>
	<b>REG</b>	<b>P/poly</b>	<b>P/poly</b>	<b>P/poly</b>
	KI 56, Mi 67	Si & So 94	Ca & Si 11,14	Ca & Si 11,14

# RESULTS: INFINITE COMPUTATION / DET.

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED EVOLVING	EVOLVING
Q	Muller FSA	Muller TM	super-Turing	super-Turing
	$\in BC(\Pi_2^0)$	$= BC(\Pi_2^0)$	$= BC(\Pi_2^0)$	$= BC(\Pi_2^0)$
	Ca & Vi 10,14	Ca & Vi 15,16	Ca & Vi 15,16	Ca & Vi 15,16
R	Muller FSA	super-Turing	super-Turing	super-Turing
	$\in BC(\Pi_2^0)$	$= BC(\Pi_2^0)$	$= BC(\Pi_2^0)$	$= BC(\Pi_2^0)$
	Ca & Vi 10,14	Ca & Vi 15,16	Ca & Vi 15,16	Ca & Vi 15,16

# RESULTS: INFINITE COMPUTATION / NONDET.

	BOOLEAN	SIGMOID		
		STATIC	BI-VALUED EVOLVING	EVOLVING
Q	Muller FSA	Muller TM	super-Turing	super-Turing
	$\in \Sigma_1^1$	$= \Sigma_1^1$	$= \Sigma_1^1$	$= \Sigma_1^1$
	Ca & Vi 10,14	Ca & Vi 15,16	Ca & Vi 15,16	Ca & Vi 15,16
R	Muller FSA	super-Turing	super-Turing	super-Turing
	$\in \Sigma_1^1$	$= \Sigma_1^1$	$= \Sigma_1^1$	$= \Sigma_1^1$
	Ca & Vi 10,14	Ca & Vi 15,16	Ca & Vi 15,16	Ca & Vi 15,16

# DRAWBACKS OF THE CONSTRUCTIONS

- ▶ Computational states of the machines are represented as Boolean states, i.e., spiking configurations of the network.
- ★ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- ▶ Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ★ Network should be robust to *architectural plasticity* and *synaptic noises*.
- ▶ We propose a novel paradigm for abstract neural computation based on *synfire rings*.

# DRAWBACKS OF THE CONSTRUCTIONS

- ▶ Computational states of the machines are represented as Boolean states, i.e., spiking configurations of the network.
- ★ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- ▶ Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ★ Network should be robust to *architectural plasticity* and *synaptic noises*.
- ▶ We propose a novel paradigm for abstract neural computation based on *synfire rings*.

# DRAWBACKS OF THE CONSTRUCTIONS

- ▶ Computational states of the machines are represented as Boolean states, i.e., spiking configurations of the network.
- ★ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- ▶ Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ★ Network should be robust to *architectural plasticity* and *synaptic noises*.
- ▶ We propose a novel paradigm for abstract neural computation based on *synfire rings*.



# DRAWBACKS OF THE CONSTRUCTIONS

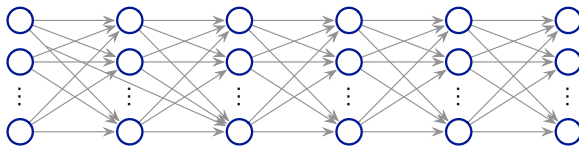
- ▶ Computational states of the machines are represented as Boolean states, i.e., spiking configurations of the network.
- ★ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- ▶ Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ★ Network should be robust to *architectural plasticity* and *synaptic noises*.
- ▶ We propose a novel paradigm for abstract neural computation based on *synfire rings*.

# DRAWBACKS OF THE CONSTRUCTIONS

- ▶ Computational states of the machines are represented as Boolean states, i.e., spiking configurations of the network.
- ★ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- ▶ Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ★ Network should be robust to *architectural plasticity* and *synaptic noises*.
- ▶ We propose a novel paradigm for abstract neural computation based on *synfire rings*.

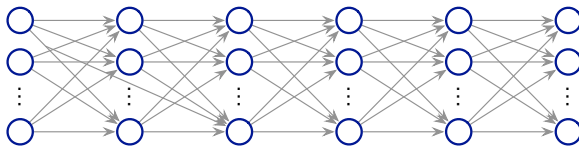
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



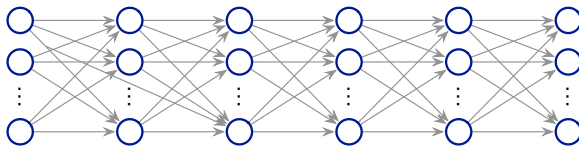
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



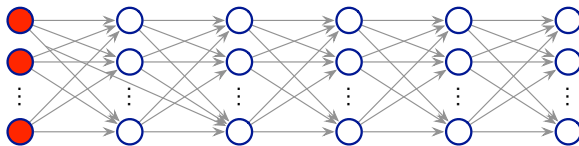
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



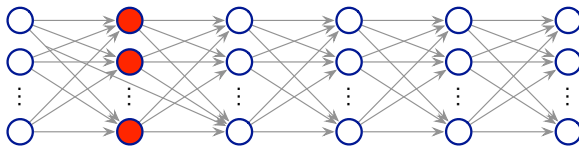
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



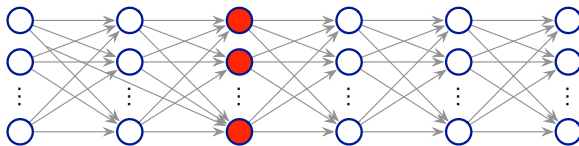
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



# SYNFIRE CHAINS

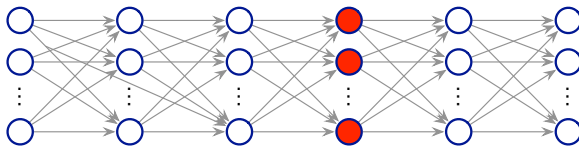
- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.





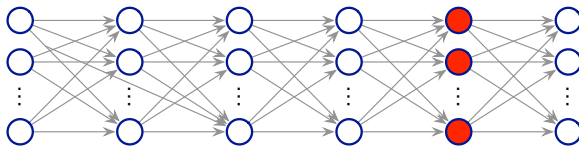
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



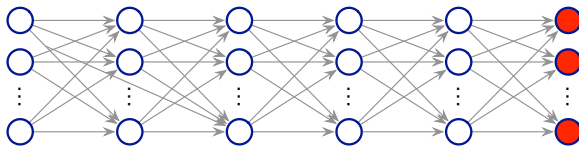
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



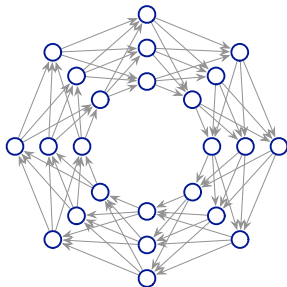
# SYNFIRE CHAINS

- ▶ *Synfire chains* have been theorized as fundamental neuronal structures (ABELES 82).
- ▶ They allow for robust and highly precise transmission of information in neural networks.



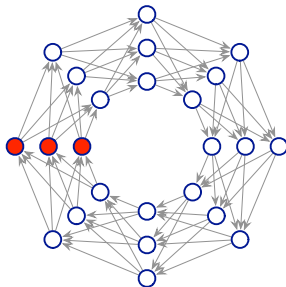
# SYNFIRE RINGS

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



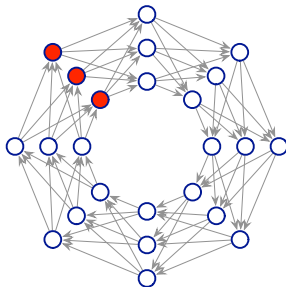
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



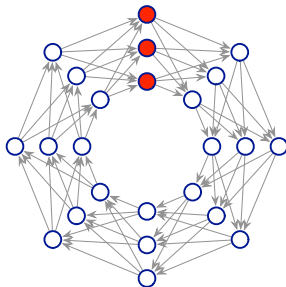
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



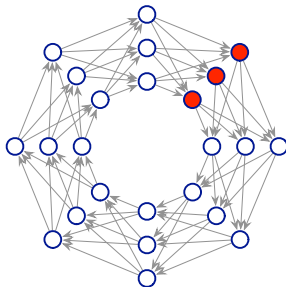
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



# SYNFIRE RINGS

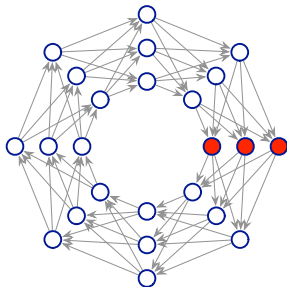
- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.





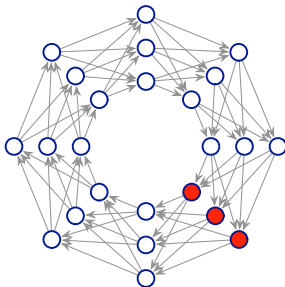
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



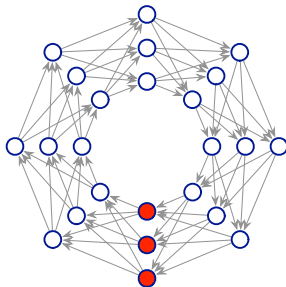
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



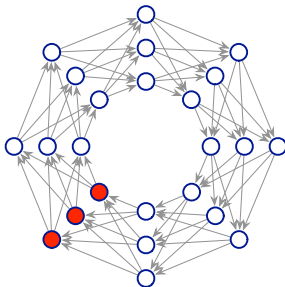
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



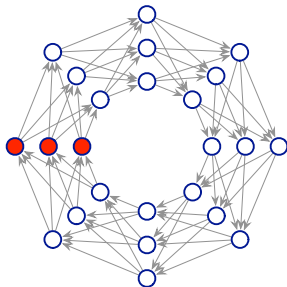
# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.

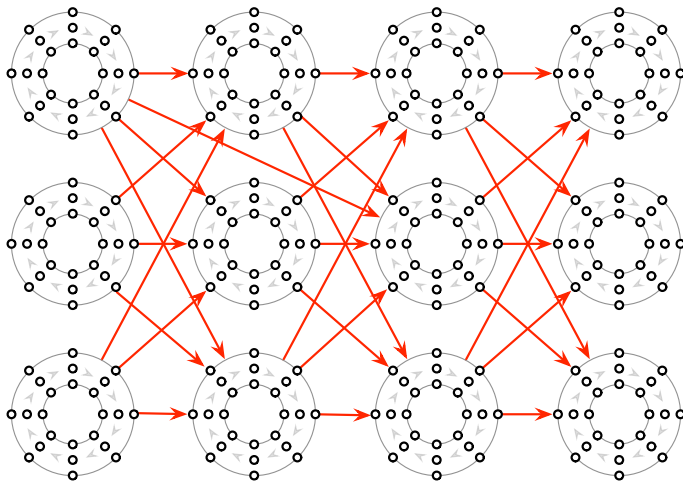


# SYNFIRE RINGS

- ▶ *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (ZHENG & TRIESCH 14).
- ▶ *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



# SYNFIRE RING ARCHITECTURE



# NEURAL COMPUTATION WITH SYNfire RINGS

- ▶ We introduce a paradigm of neural computation based on *synfire rings*.
- ▶ Computational states are represented by sustained activities within synfire rings.
- ▶ Hence, the successive computational states are encoded into cyclic attractors.
- ▶ The transitions between such attractors are perfectly controlled by the inputs.
- ▶ The global computational process is robust to various kinds of architectural plasticities and noises.

# NEURAL COMPUTATION WITH SYNfire RINGS

- ▶ We introduce a paradigm of neural computation based on *synfire rings*.
- ▶ Computational states are represented by sustained activities within synfire rings.
- ▶ Hence, the successive computational states are encoded into cyclic attractors.
- ▶ The transitions between such attractors are perfectly controlled by the inputs.
- ▶ The global computational process is robust to various kinds of architectural plasticities and noises.



# NEURAL COMPUTATION WITH SYNfire RINGS

- ▶ We introduce a paradigm of neural computation based on *synfire rings*.
- ▶ Computational states are represented by sustained activities within synfire rings.
- ▶ Hence, the successive computational states are encoded into cyclic attractors.
- ▶ The transitions between such attractors are perfectly controlled by the inputs.
- ▶ The global computational process is robust to various kinds of architectural plasticities and noises.

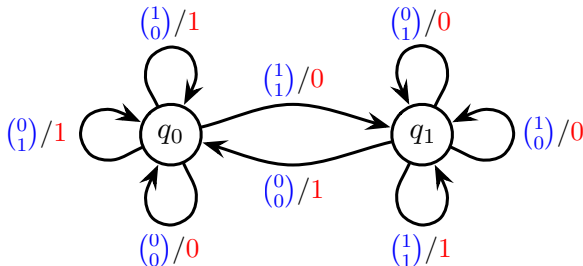
# NEURAL COMPUTATION WITH SYNfire RINGS

- ▶ We introduce a paradigm of neural computation based on *synfire rings*.
- ▶ Computational states are represented by sustained activities within synfire rings.
- ▶ Hence, the successive computational states are encoded into cyclic attractors.
- ▶ The transitions between such attractors are perfectly controlled by the inputs.
- ▶ The global computational process is robust to various kinds of architectural plasticities and noises.

# NEURAL COMPUTATION WITH SYNfire RINGS

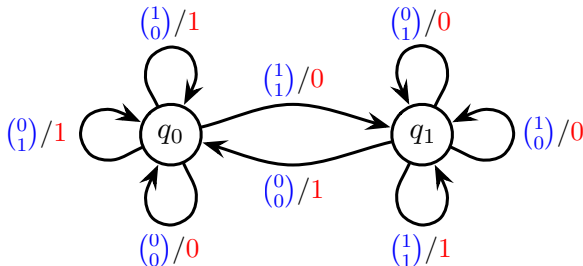
- ▶ We introduce a paradigm of neural computation based on *synfire rings*.
- ▶ Computational states are represented by sustained activities within synfire rings.
- ▶ Hence, the successive computational states are encoded into cyclic attractors.
- ▶ The transitions between such attractors are perfectly controlled by the inputs.
- ▶ The global computational process is robust to various kinds of architectural plasticities and noises.

# BINARY ADDER AUTOMATON



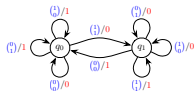
$$\begin{array}{r}
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \hline
 \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0}
 \end{array}$$

# BINARY ADDER AUTOMATON



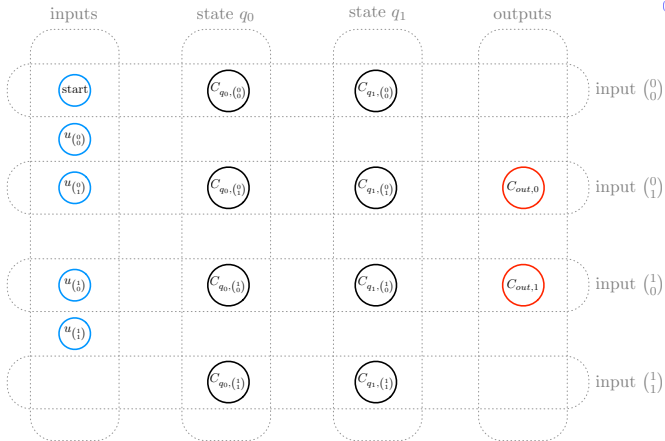
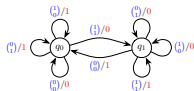
$$\begin{array}{rcccccccc}
 & & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
 \hline
 & & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

# BINARY ADDER BOOLEAN NEURAL NETWORK

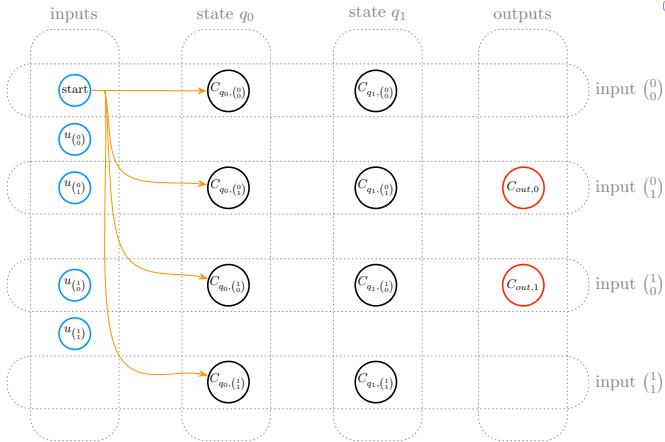
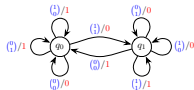


inputs	state $q_0$	state $q_1$	outputs
			input $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$
			input $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
			input $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$
			input $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$

# BINARY ADDER BOOLEAN NEURAL NETWORK

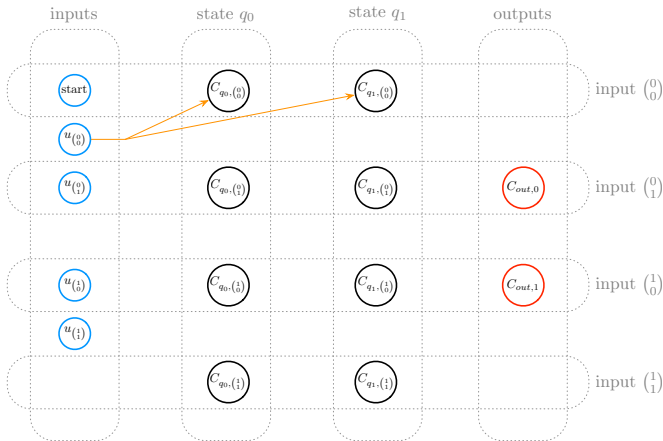
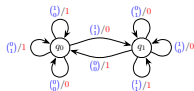


# BINARY ADDER BOOLEAN NEURAL NETWORK

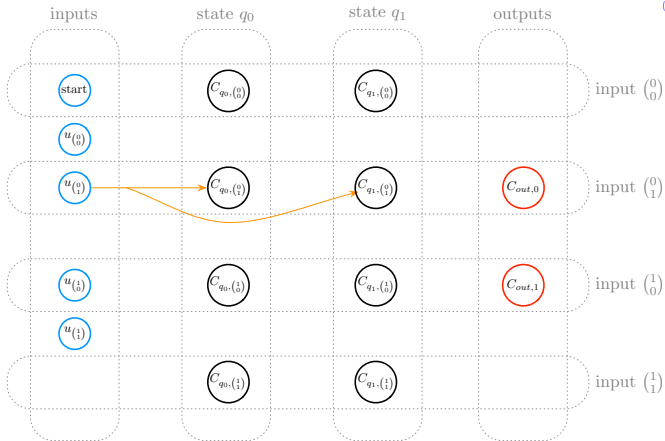
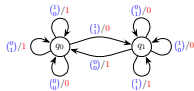




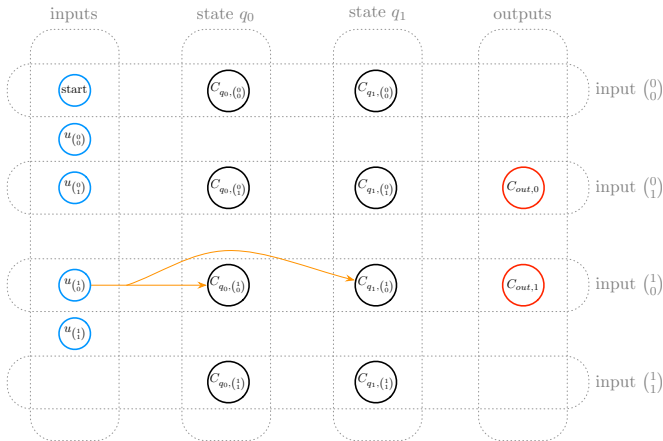
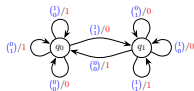
# BINARY ADDER BOOLEAN NEURAL NETWORK



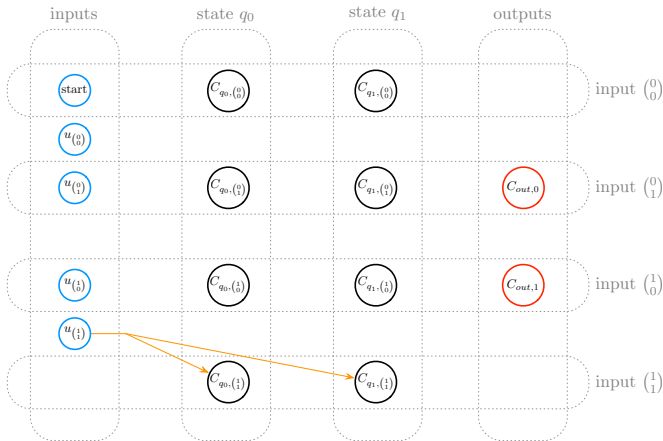
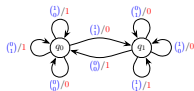
# BINARY ADDER BOOLEAN NEURAL NETWORK



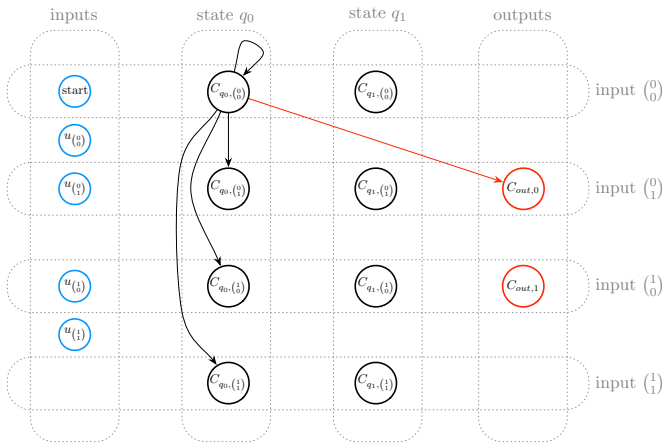
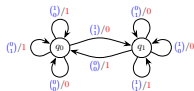
# BINARY ADDER BOOLEAN NEURAL NETWORK



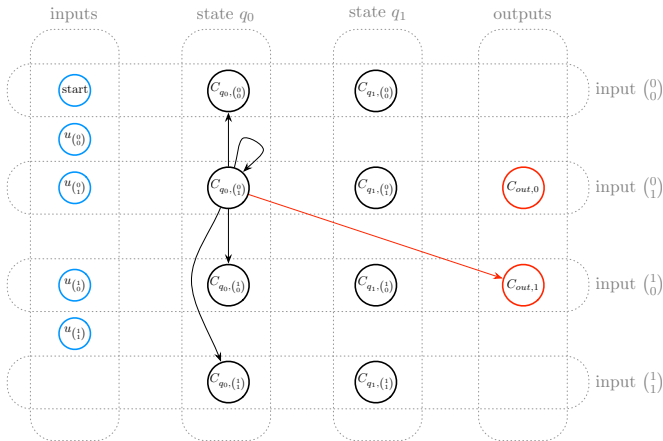
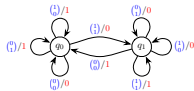
# BINARY ADDER BOOLEAN NEURAL NETWORK



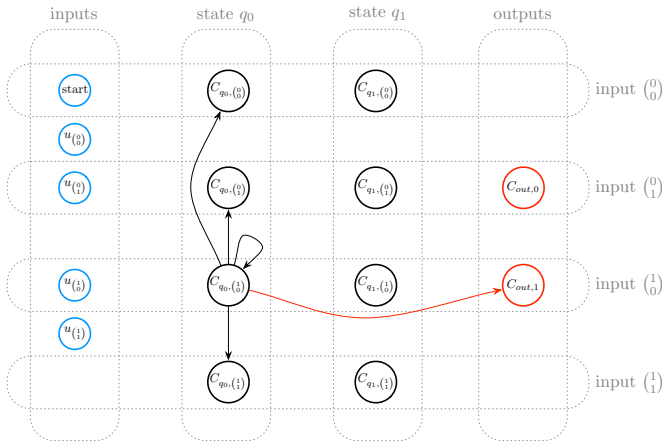
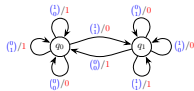
# BINARY ADDER BOOLEAN NEURAL NETWORK



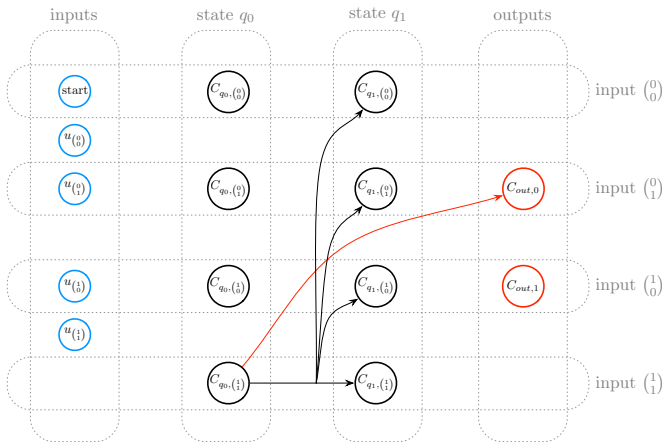
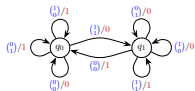
# BINARY ADDER BOOLEAN NEURAL NETWORK



# BINARY ADDER BOOLEAN NEURAL NETWORK

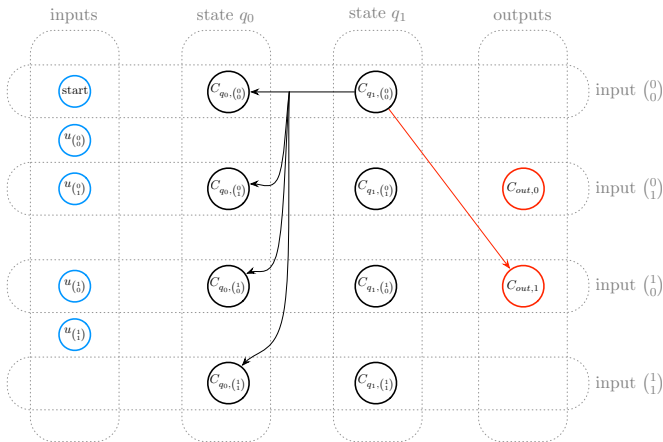
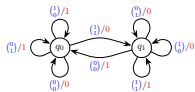


# BINARY ADDER BOOLEAN NEURAL NETWORK

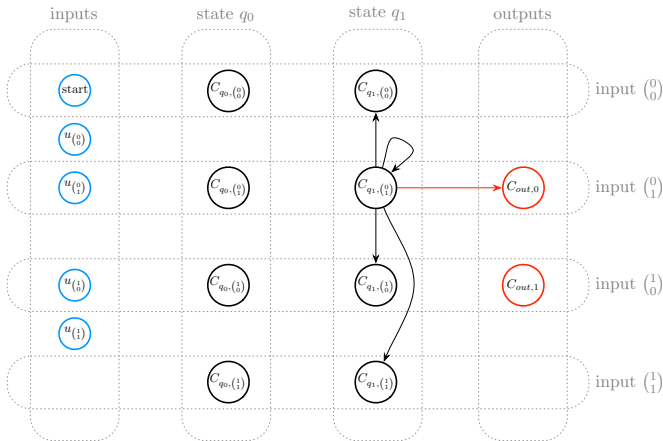
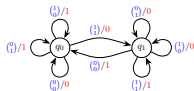




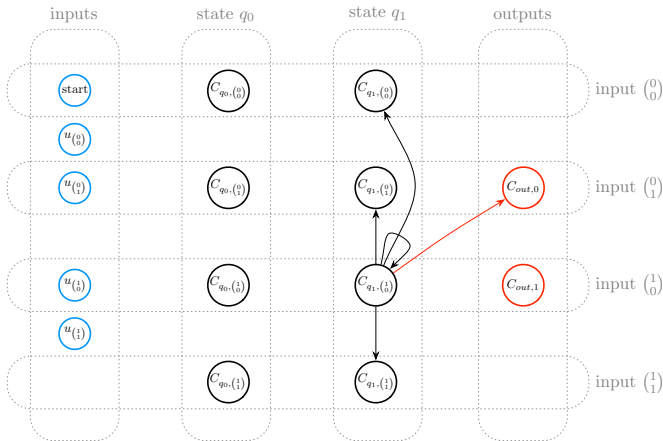
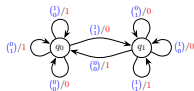
# BINARY ADDER BOOLEAN NEURAL NETWORK



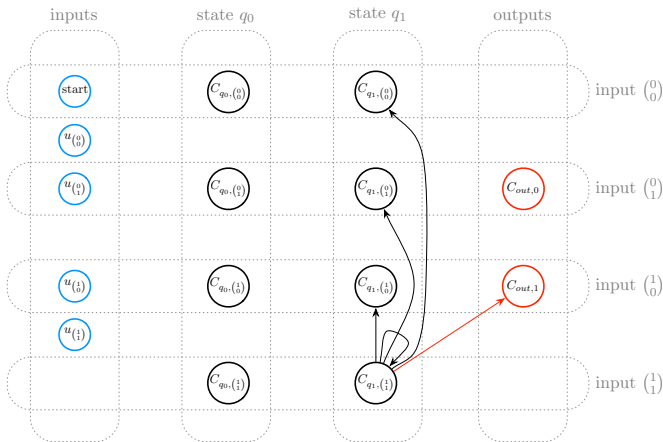
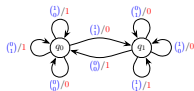
# BINARY ADDER BOOLEAN NEURAL NETWORK



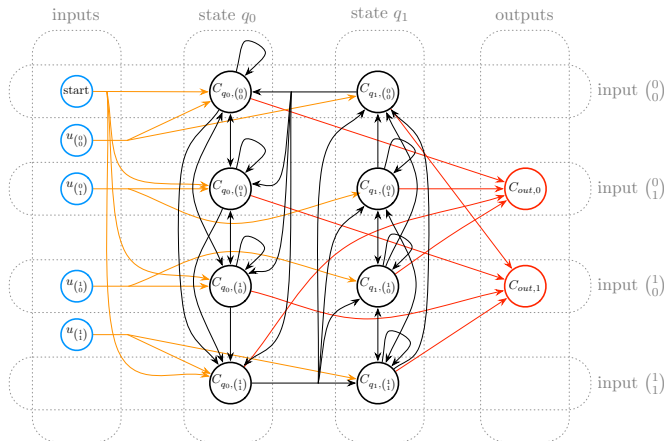
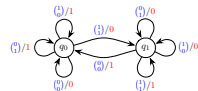
# BINARY ADDER BOOLEAN NEURAL NETWORK



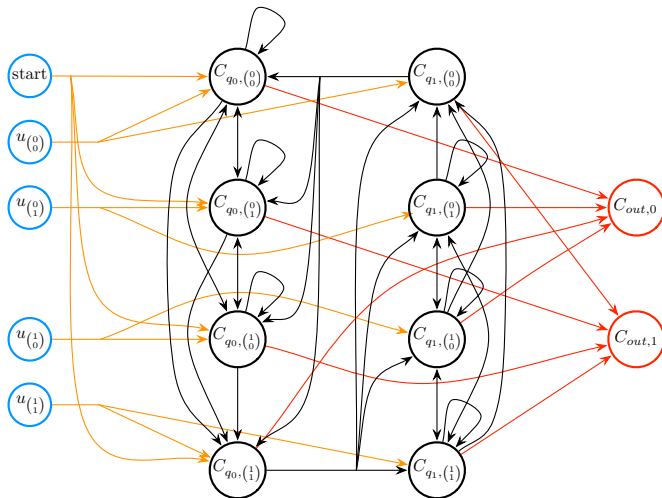
# BINARY ADDER BOOLEAN NEURAL NETWORK



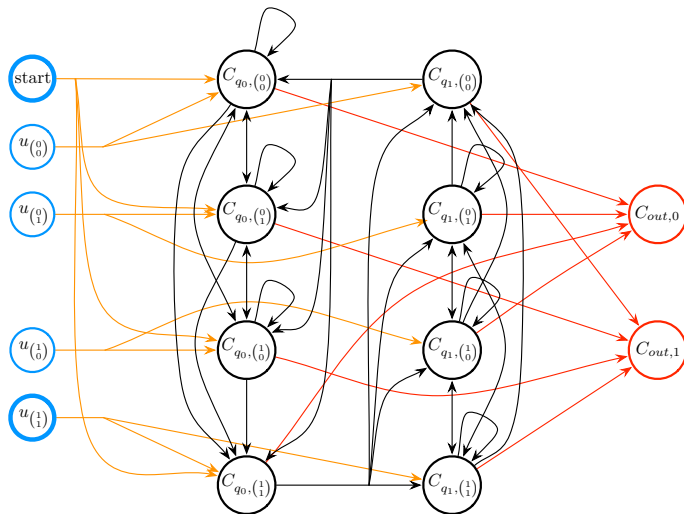
# BINARY ADDER BOOLEAN NEURAL NETWORK



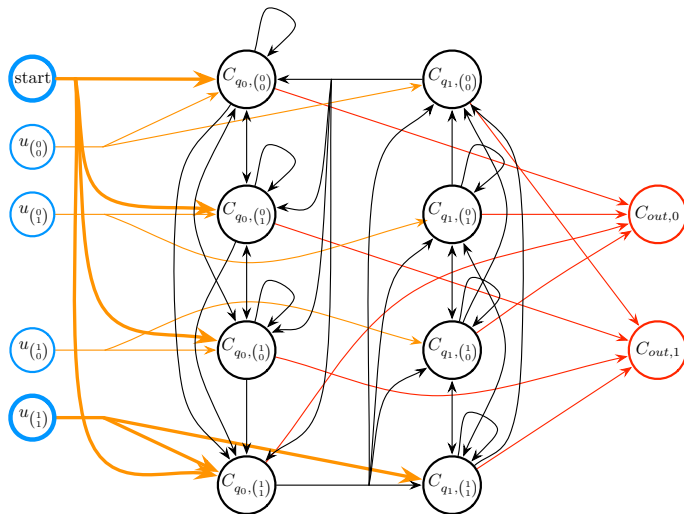
## SIMULATION



## SIMULATION

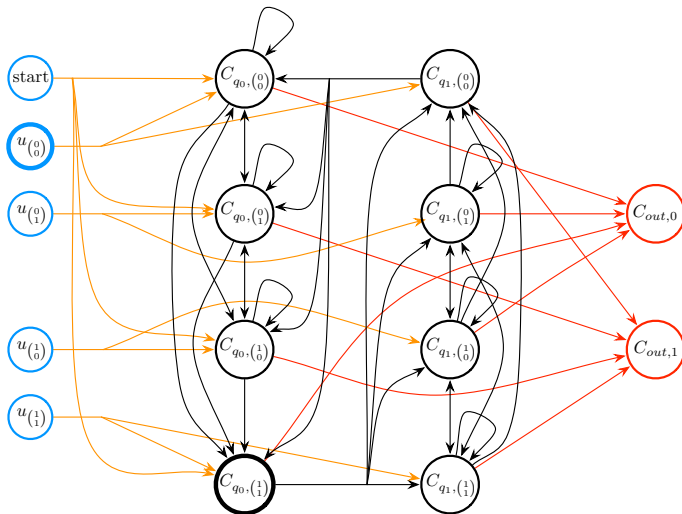


# SIMULATION

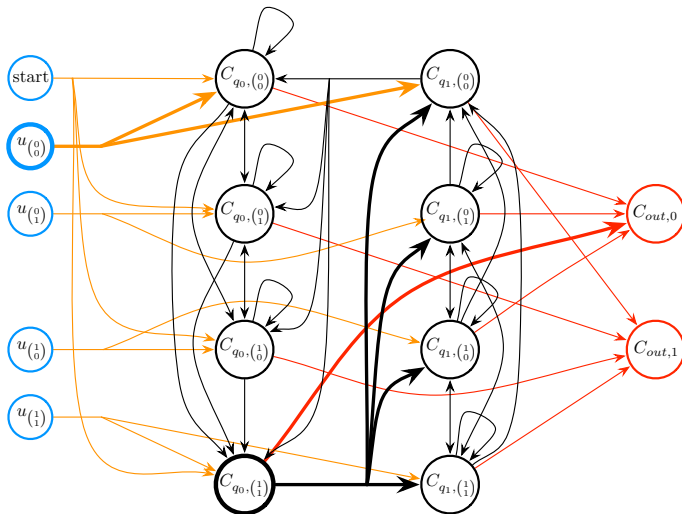




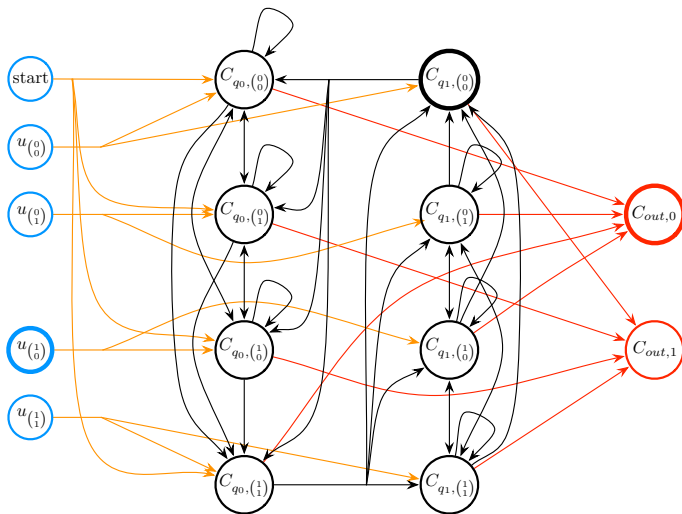
## SIMULATION



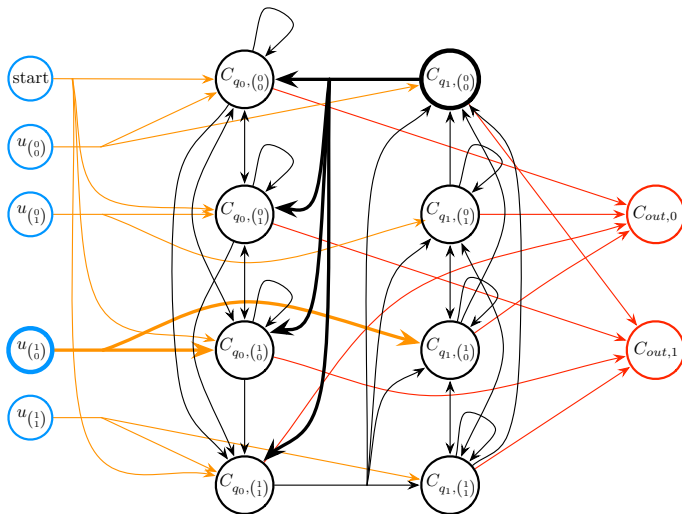
## SIMULATION



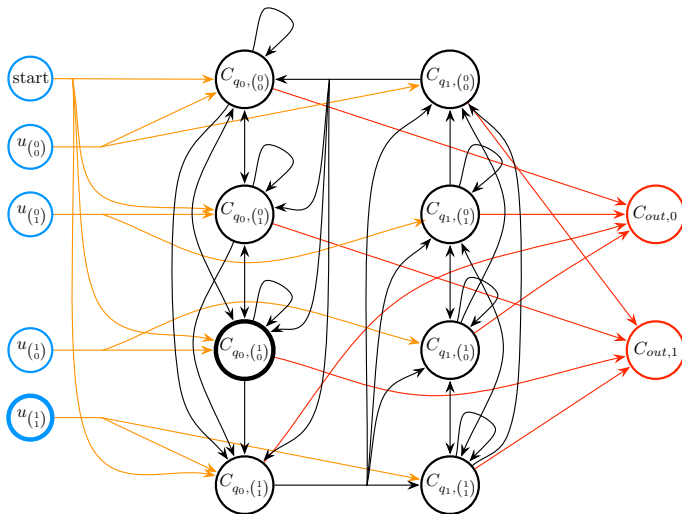
## SIMULATION



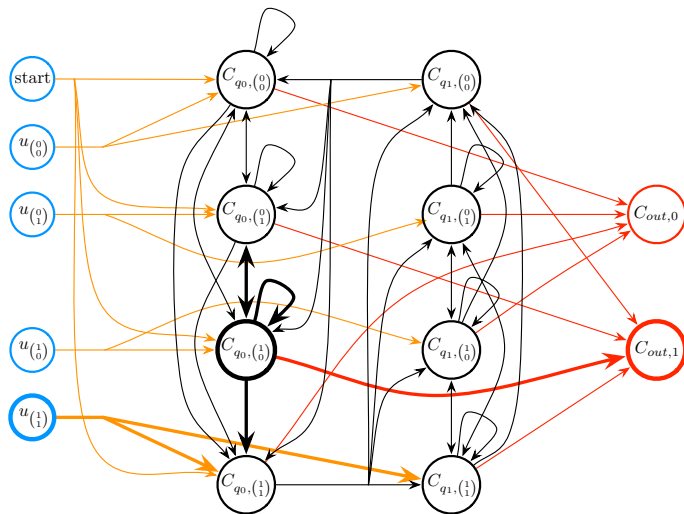
## SIMULATION



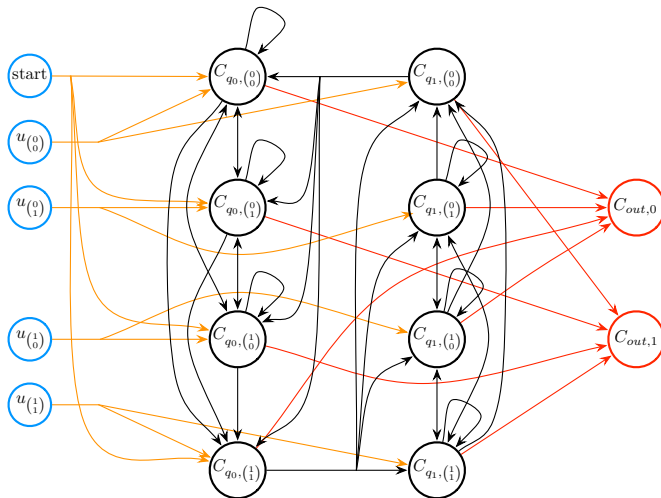
## SIMULATION



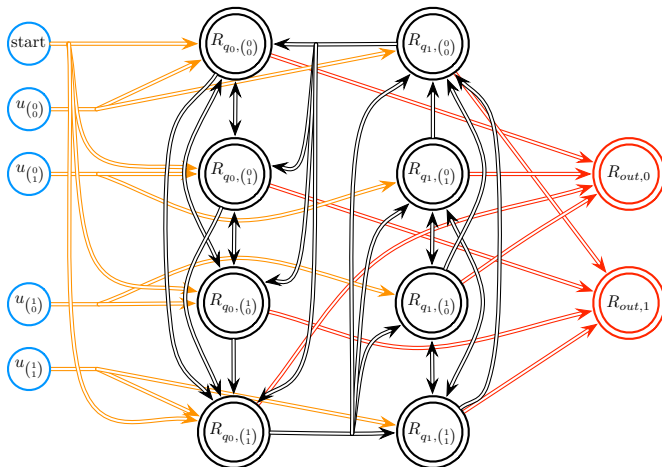
## SIMULATION



# GENERALIZATION TO SYNFIRED RING RNNs

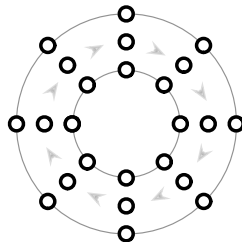
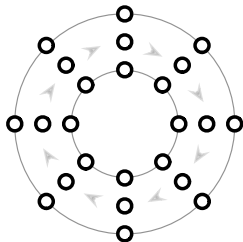


# GENERALIZATION TO SYNFIRED RING RNNs

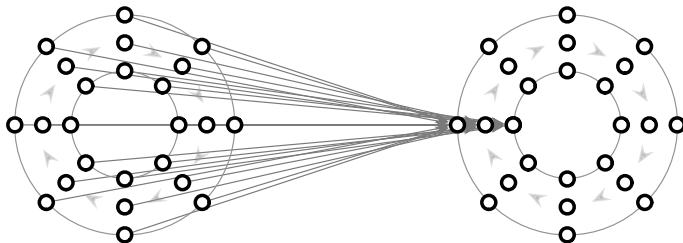




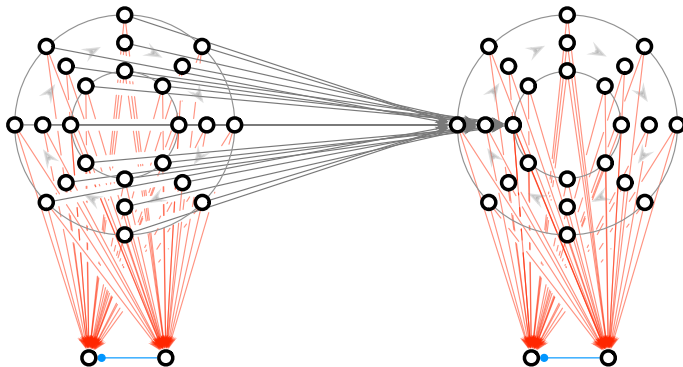
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



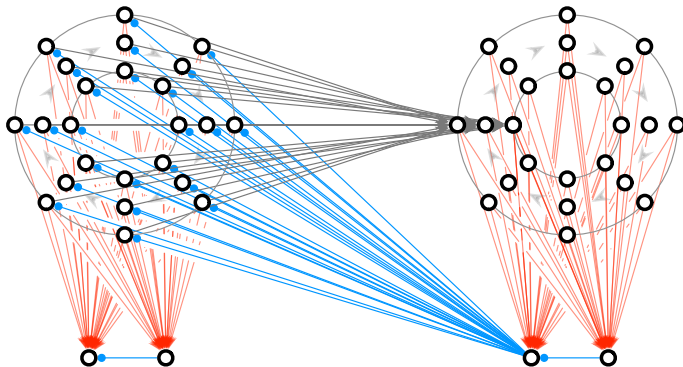
## FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



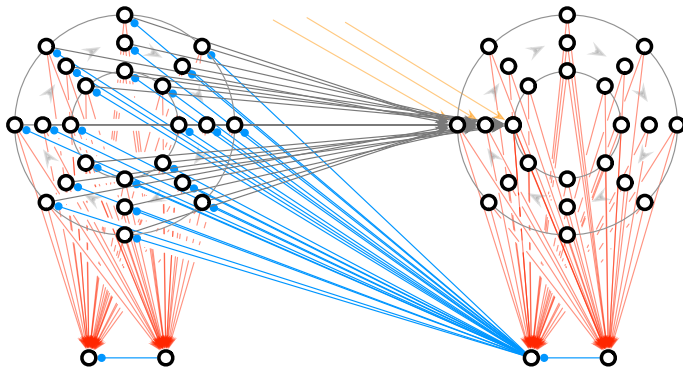
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



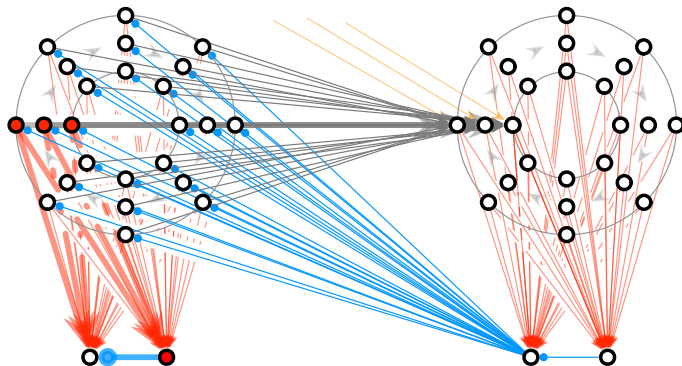
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



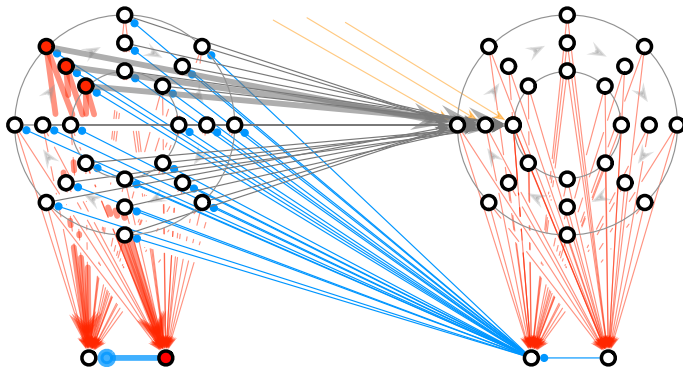
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



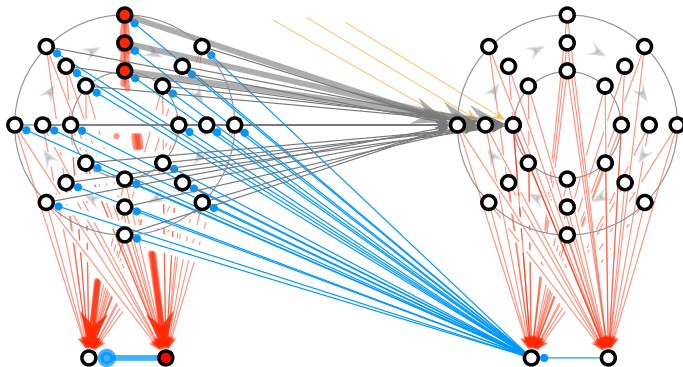
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

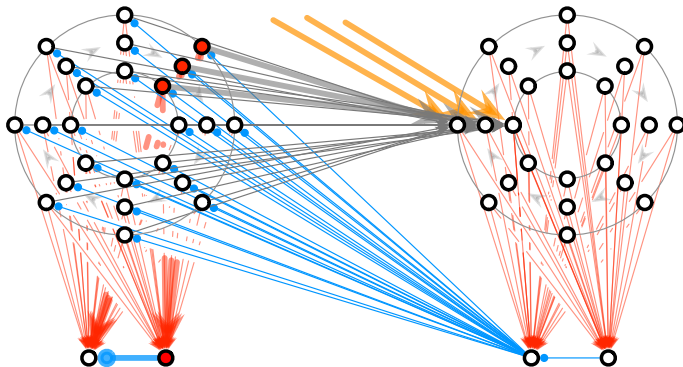


# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

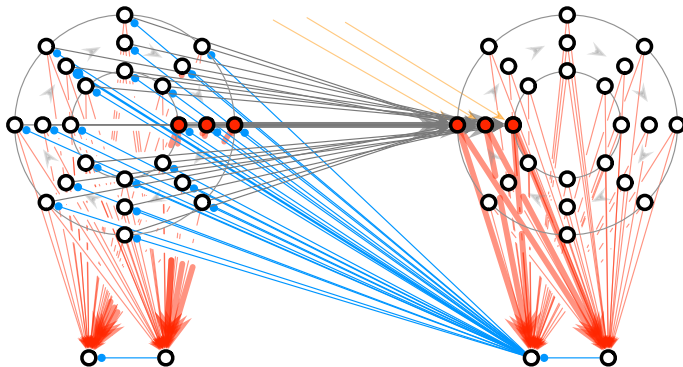




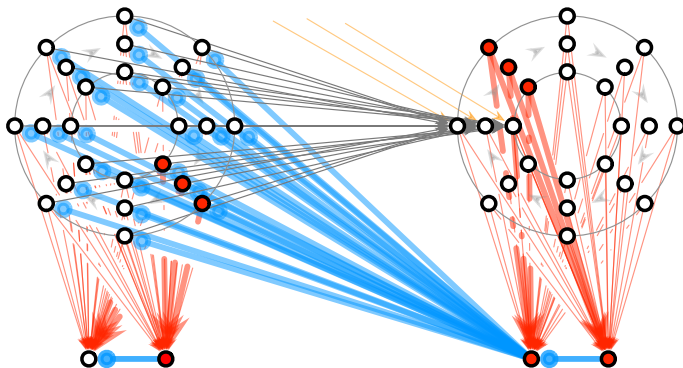
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



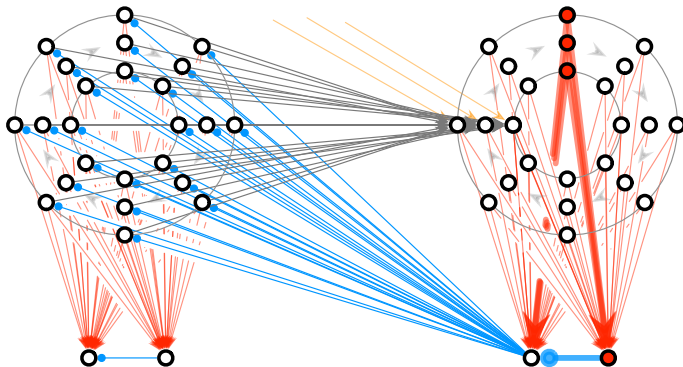
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



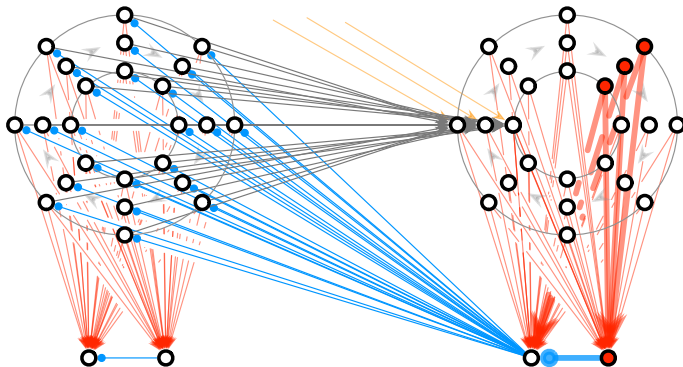
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



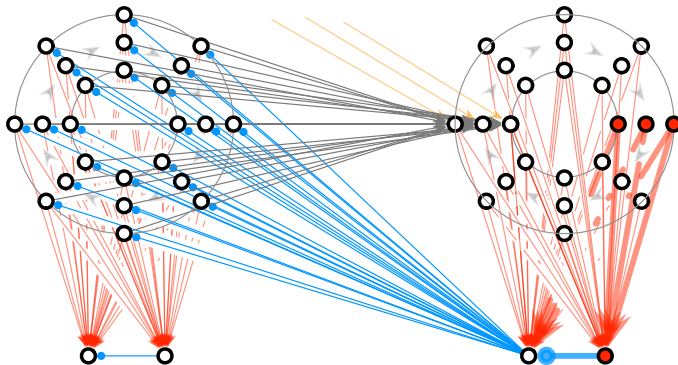
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



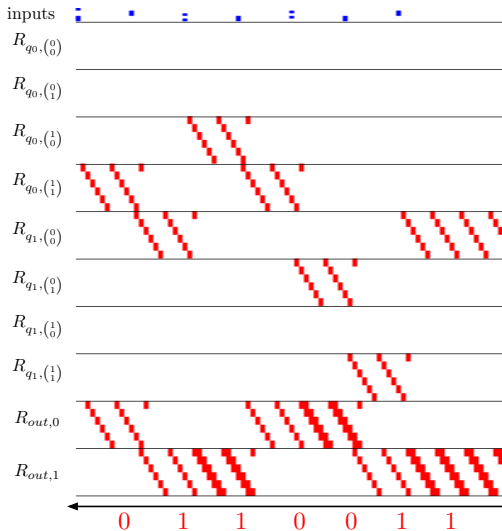
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



# SIMULATION

Play movie...

# SIMULATION





# AUTOMATA & BOOLEAN RNNs WITH SYNFIRES

Since the construction is generic, one has the following result:

## THEOREM

*Any finite state automaton can be simulated by some Boolean neural network composed of synfire rings.*

# GENERALIZATIONS

- ▶ We generalize these results to the contexts of more biological neural networks:
  1. Izhikevich spiking neural networks
  2. Hodgkin-Huxley neural networks

# IZHIKEVICH SPIKING NEURAL NETWORKS

The Izhikevich differential equations are:

$$\begin{cases} v' &= 0.04v^2 + 5v + 140 - u + I \\ u' &= a(bv - u) \end{cases}$$

with the auxiliary after-spike resetting:

if  $v \geq 30$  mV, then  $v \leftarrow c$  and  $u \leftarrow u + d$

- ▶  $v$ : membrane potential
- ▶  $u$ : membrane recovery variable
- ▶  $I$ : synaptic currents
- ▶  $a, b, c, d$ : dimensionless parameters

# AUTOMATA & IZHIKEVICH RNNs WITH SYNFIRES

## RINGS

Since the construction is generic, one has the following result:

### RESULT

Any finite state automaton can be simulated by some (noisy) Izhikevich spiking neural network composed of synfire rings.

# HODGKIN-HUXLEY NEURONS

$$\alpha_n(V_m) = \frac{0.01(10 - V_m)}{\exp(\frac{10 - V_m}{10}) - 1}$$

$$\beta_n(V_m) = 0.125 \exp(\frac{-V_m}{80})$$

$$\alpha_m(V_m) = \frac{0.1(25 - V_m)}{\exp(\frac{25 - V_m}{10}) - 1}$$

$$\beta_m(V_m) = 4 \exp(\frac{-V_m}{18})$$

$$\alpha_h(V_m) = 0.07 \exp(\frac{-V_m}{20})$$

$$\beta_h(V_m) = \frac{1}{\exp(\frac{30 - V_m}{10}) + 1}$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1 - n) - \beta_n(V_m)n$$

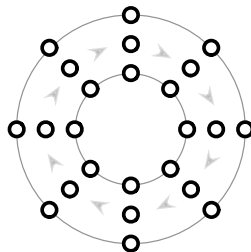
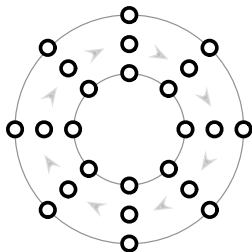
$$\frac{dm}{dt} = \alpha_m(V_m)(1 - m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1 - h) - \beta_h(V_m)h$$

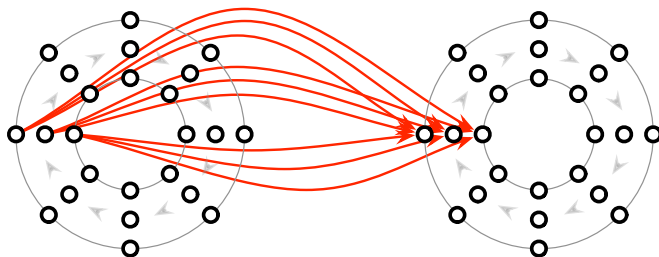
$$C_m \frac{dV_m}{dt} = I - \bar{g}_K n^4 (V_m - V_K) - \bar{g}_{Na} m^3 h (V_m - V_{Na}) - \bar{g}_l (V_m - V_l)$$



# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

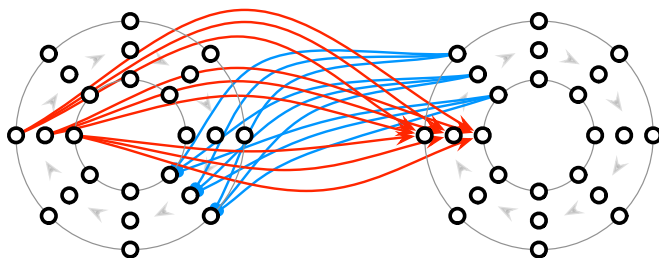


# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

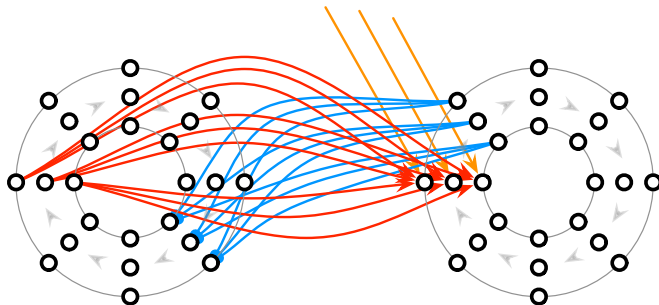




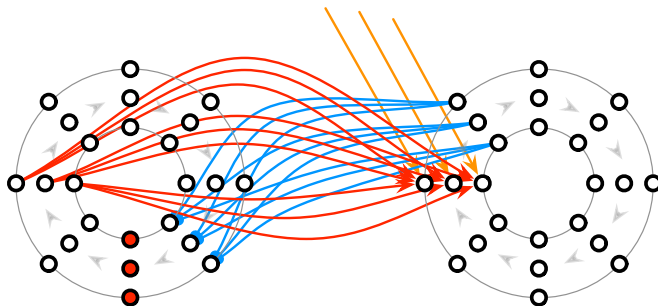
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



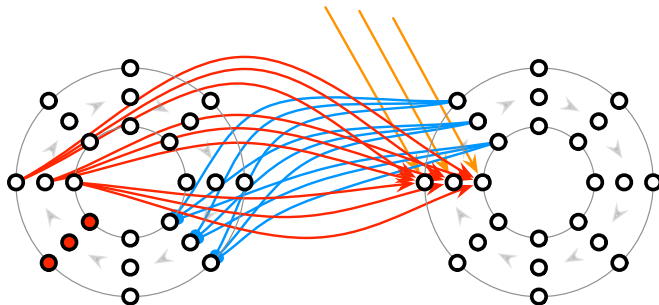
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



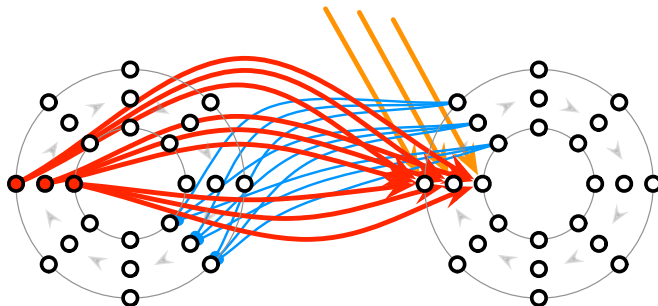
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



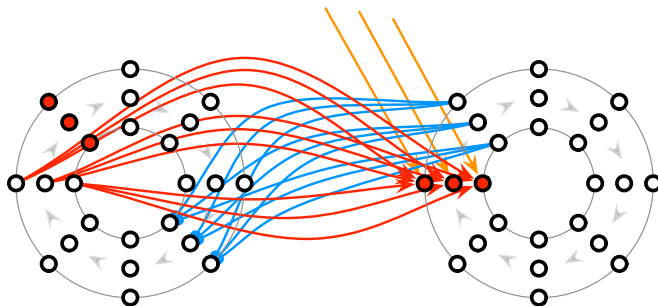
## FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



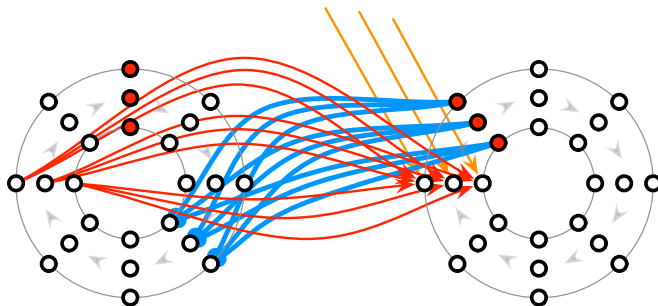
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



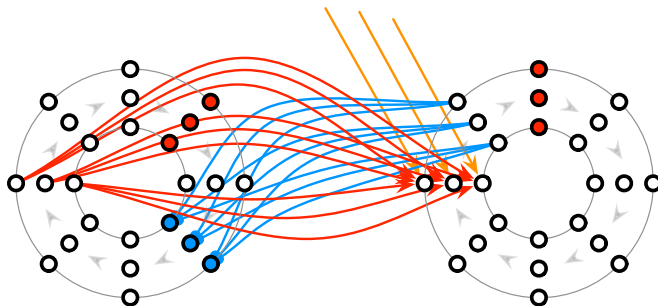
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

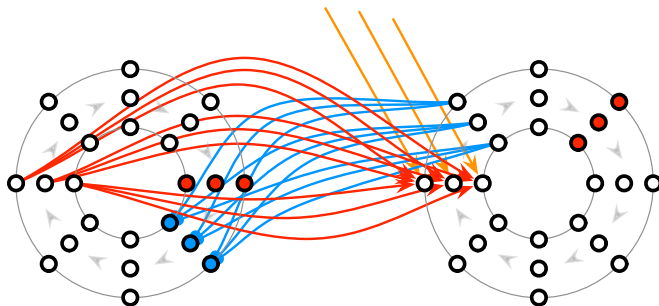


# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

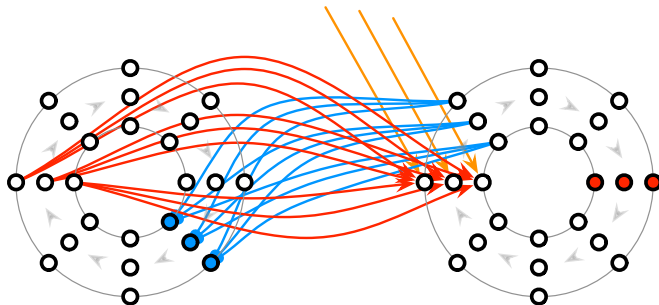




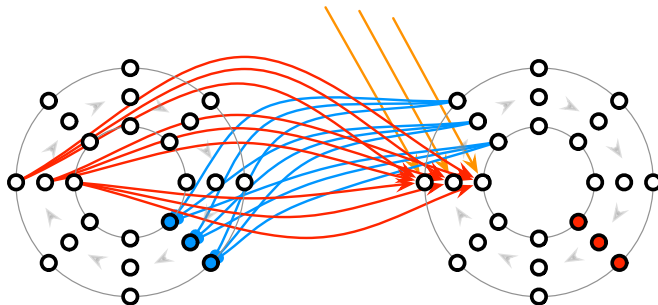
## FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

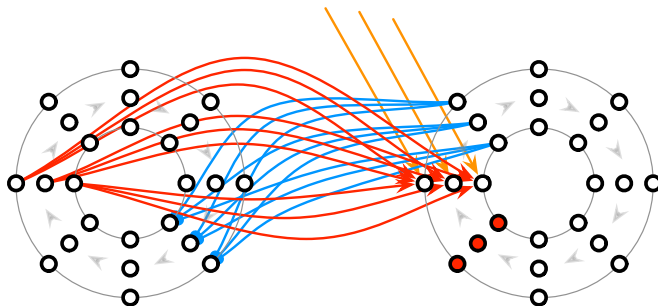


# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM



The diagram shows two sets of nodes, each arranged in a circular pattern. The left set has 12 nodes, and the right set has 12 nodes. Directed edges (arrows) connect nodes between the two sets. Red edges originate from the left set and point to the right set. Blue edges originate from the right set and point to the left set. Three orange lines point to specific nodes in the right set, which are highlighted with red circles. The overall structure suggests a bipartite graph or a network flow problem.

# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM





# AUTOMATA & HODGKIN-HUXLEY RNNs WITH SYNFIRES

Since the construction is generic, one has the following result:

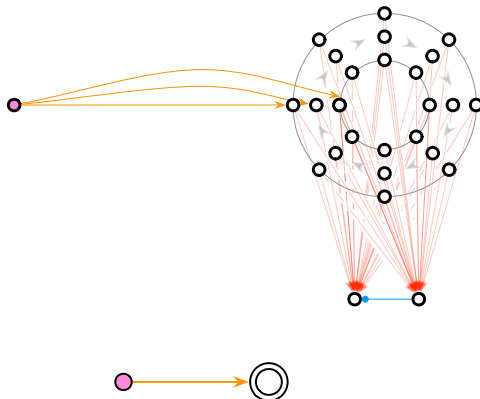
## RESULT

Any finite state automaton can be simulated by some Hodgkin-Huxley based neural network composed of synfire rings.

# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

We consider the following kinds of (fibres of) connections.

- cell to ring excitatory

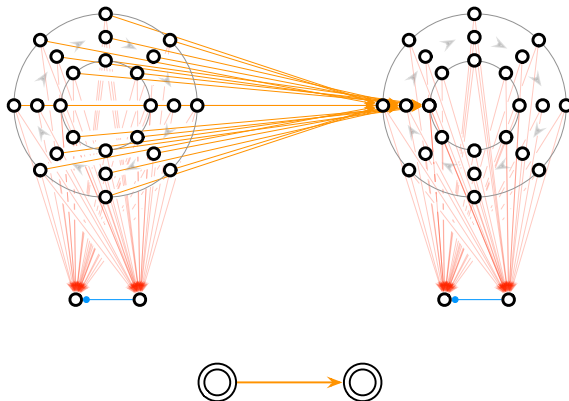




# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

We consider the following kinds of (fibres of) connections.

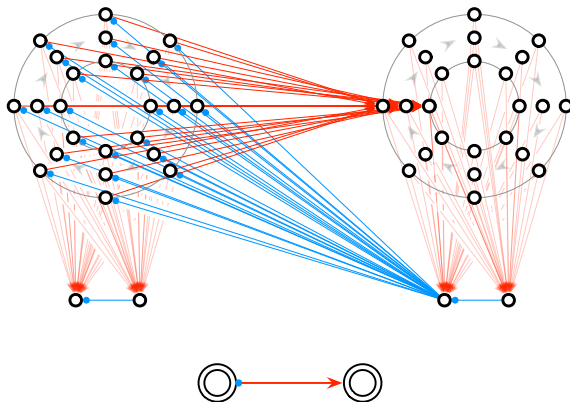
- ▶ constant excitatory



# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

We consider the following kinds of (fibres of) connections.

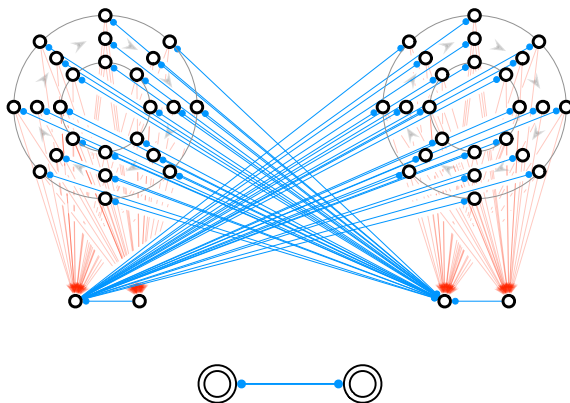
- constant excitatory / one-shot inhibitory



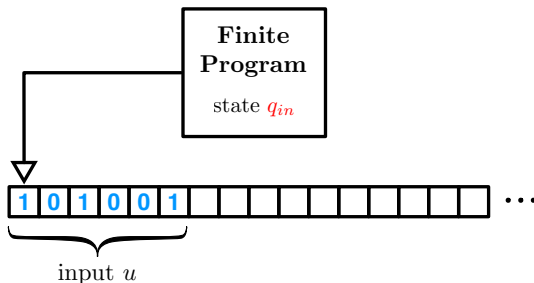
# FIBRES OF CONNECTIONS & INHIBITORY SYSTEM

We consider the following kinds of (fibres of) connections.

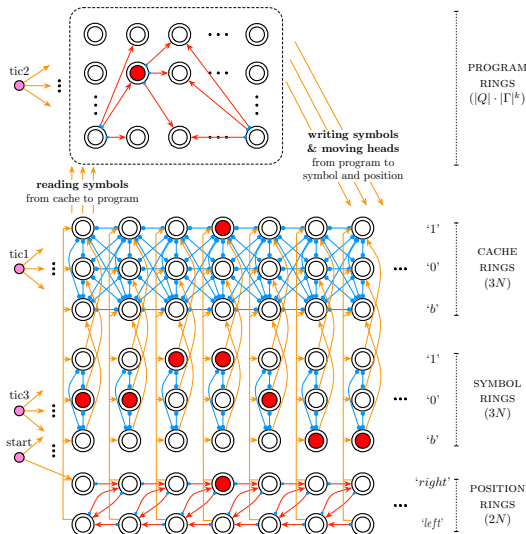
- ▶ one-shot inhibitory / one-shot inhibitory



# GENERAL ARCHITECTURE

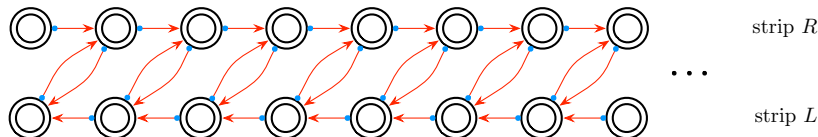


# GENERAL ARCHITECTURE



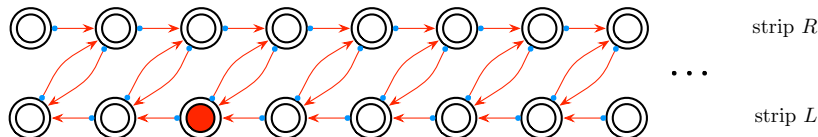
# POSITION RINGS

- ▶ Used to store the current position of the head.
- ▶ Composed of a “left” and a “right” strip.
- ▶ excitatory/inhibitory connections.



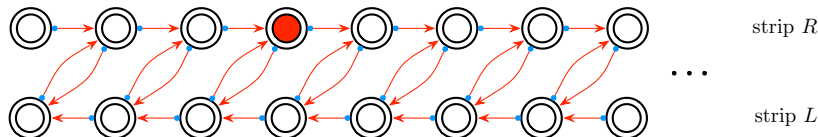
# POSITION RINGS

- ▶ Used to store the current position of the head.
- ▶ Composed of a “left” and a “right” strip.
- ▶ excitatory/inhibitory connections.



# POSITION RINGS

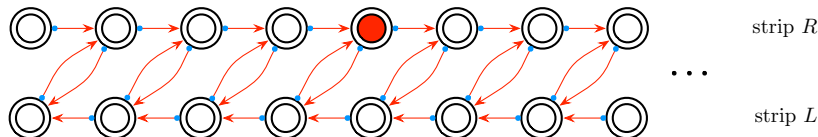
- ▶ Used to store the current position of the head.
- ▶ Composed of a “left” and a “right” strip.
- ▶ excitatory/inhibitory connections.





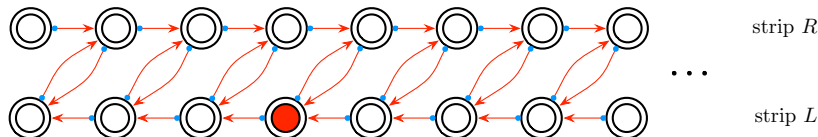
# POSITION RINGS

- ▶ Used to store the current position of the head.
- ▶ Composed of a “left” and a “right” strip.
- ▶ excitatory/inhibitory connections.



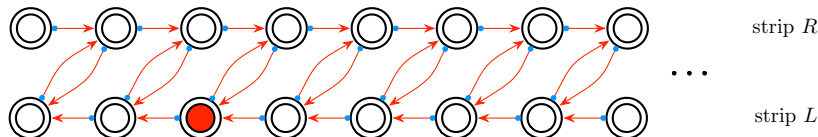
# POSITION RINGS

- ▶ Used to store the current position of the head.
- ▶ Composed of a “left” and a “right” strip.
- ▶ excitatory/inhibitory connections.



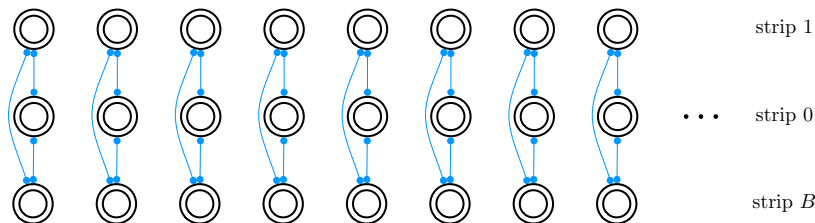
# POSITION RINGS

- ▶ Used to store the current position of the head.
- ▶ Composed of a “left” and a “right” strip.
- ▶ excitatory/inhibitory connections.



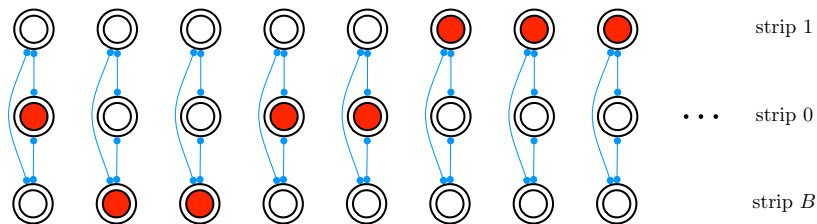
# SYMBOL RINGS

- ▶ Used to store the symbols written on the tape.
- ▶ Composed of a “blank”, a “0” and a “1” strip.
- ▶ inhibitory/inhibitory connections.



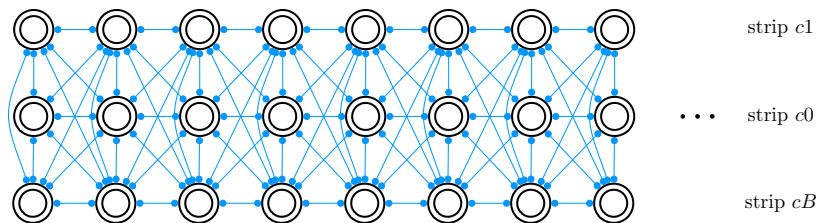
# SYMBOL RINGS

- ▶ Used to store the symbols written on the tape.
- ▶ Composed of a “blank”, a “0” and a “1” strip.
- ▶ inhibitory/inhibitory connections.



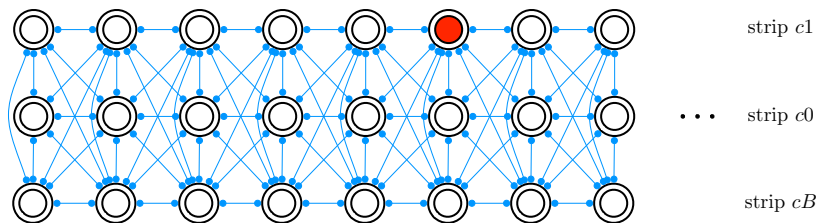
# CACHE RINGS

- ▶ Used to store the symbol under the current head's position.
- ▶ Composed of a “blank”, a “0” and a “1” strip.
- ▶ inhibitory/inhibitory connections.

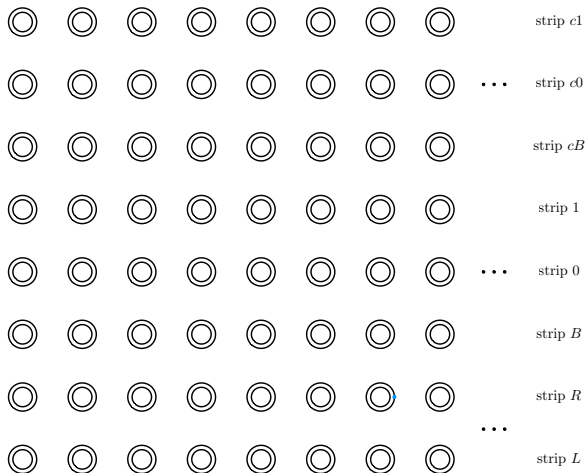


# CACHE RINGS

- ▶ Used to store the symbol under the current head's position.
- ▶ Composed of a “blank”, a “0” and a “1” strip.
- ▶ inhibitory/inhibitory connections.

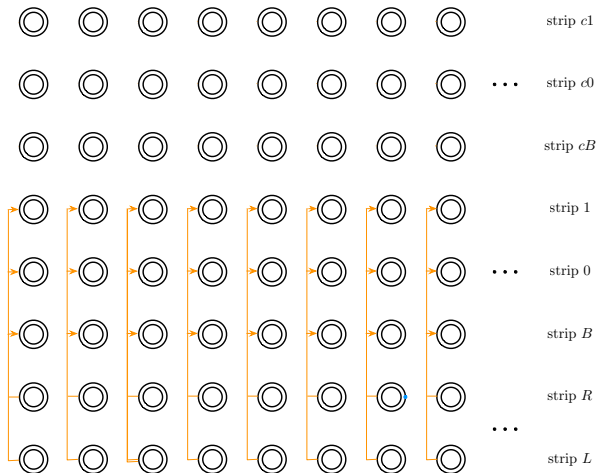


# POSITION-SYMBOL-CACHE CONNECTIONS

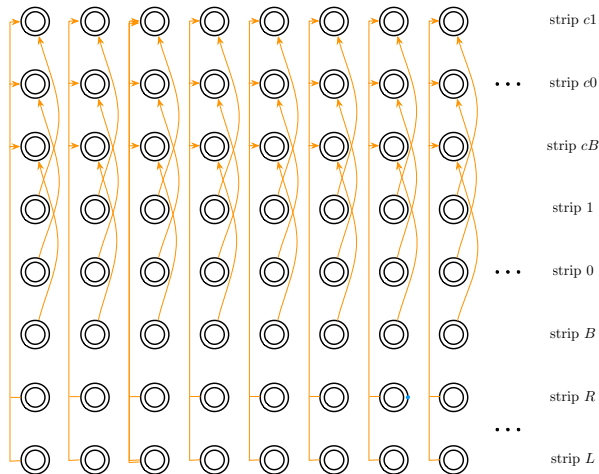




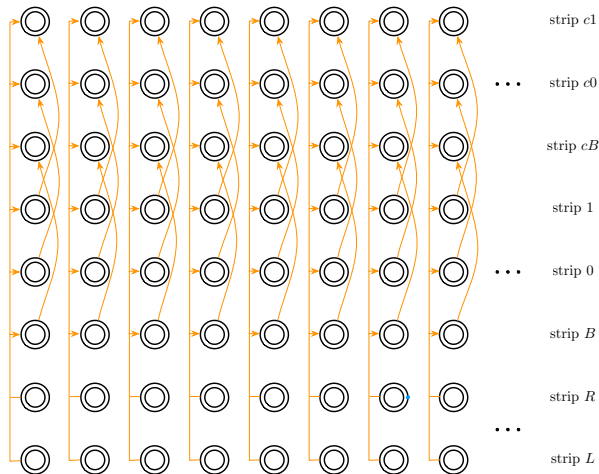
# POSITION-SYMBOL-CACHE CONNECTIONS



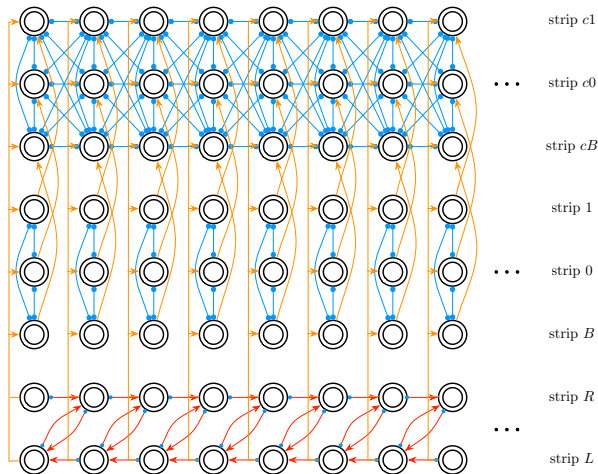
# POSITION-SYMBOL-CACHE CONNECTIONS



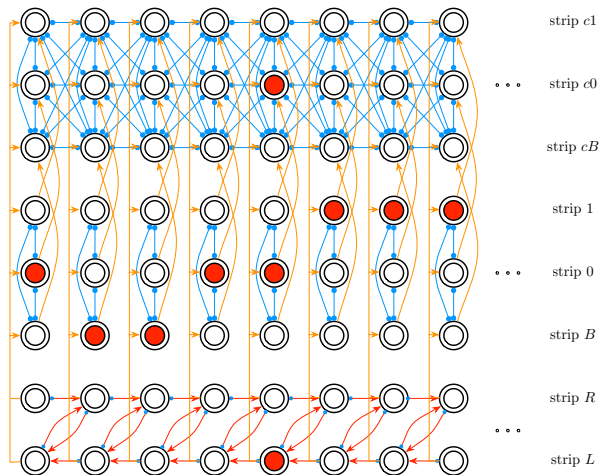
# POSITION-SYMBOL-CACHE CONNECTIONS



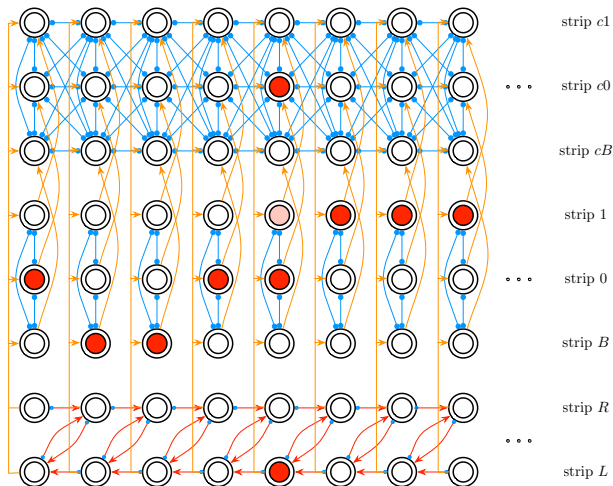
# POSITION-SYMBOL-CACHE CONNECTIONS



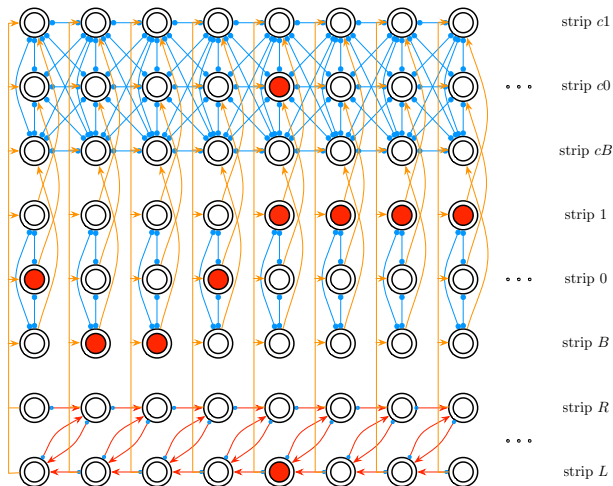
# POSITION-SYMBOL-CACHE CONNECTIONS



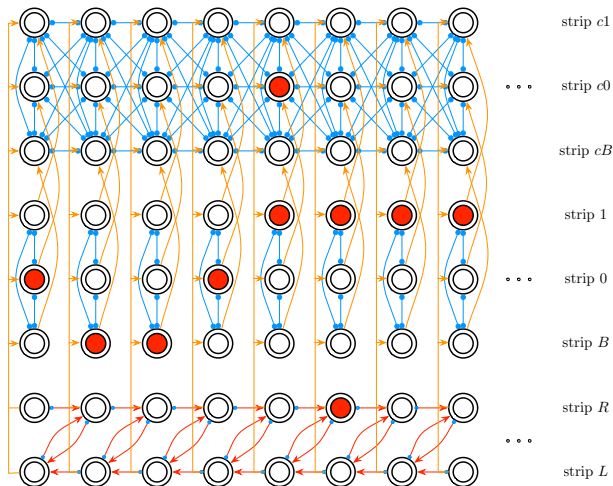
# POSITION-SYMBOL-CACHE CONNECTIONS



# POSITION-SYMBOL-CACHE CONNECTIONS

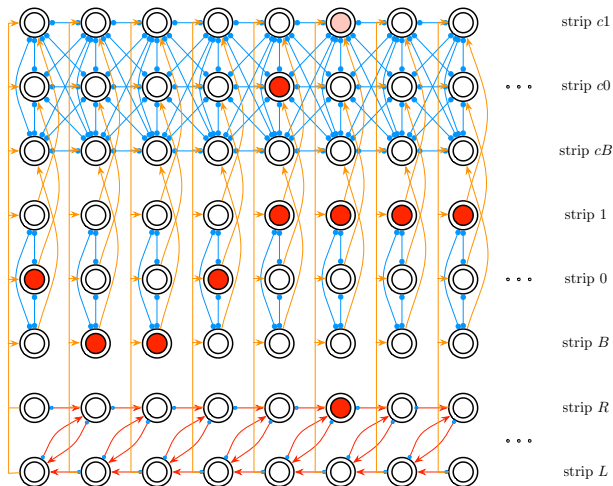


# POSITION-SYMBOL-CACHE CONNECTIONS

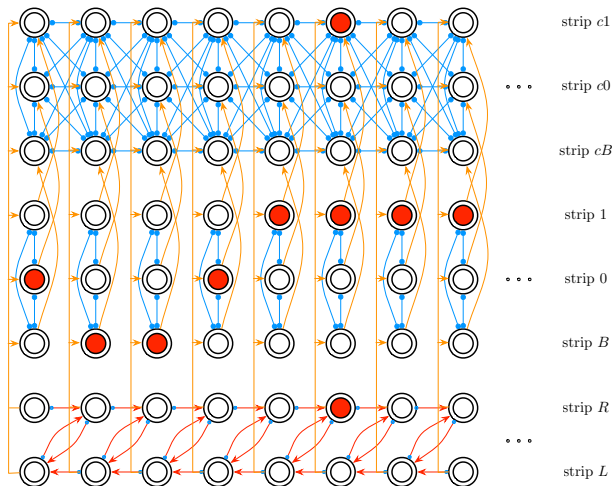




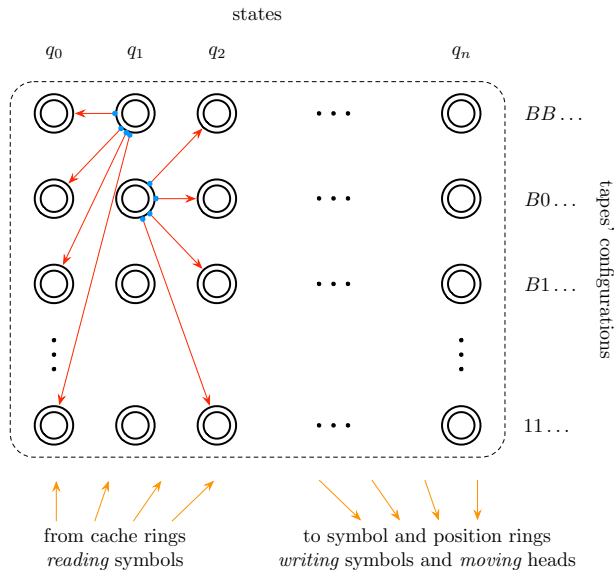
# POSITION-SYMBOL-CACHE CONNECTIONS



# POSITION-SYMBOL-CACHE CONNECTIONS



# PROGRAM RINGS



# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:  
⇒ change state
- ▶ Cache to program: excitatory  
⇒ read current symbol
- ▶ Program to symbol: excitatory  
⇒ write new symbol
- ▶ Program to position: excitatory  
⇒ move head

# PROGRAM RINGS

▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.

▶ Program to program: excitatory/inhibitory:

⇒ change state

▶ Cache to program: excitatory

⇒ read current symbol

▶ Program to symbol: excitatory

⇒ write new symbol

▶ Program to position: excitatory

⇒ move head

# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:
  - ⇒ change state
- ▶ Cache to program: excitatory
  - ⇒ read current symbol
- ▶ Program to symbol: excitatory
  - ⇒ write new symbol
- ▶ Program to position: excitatory
  - ⇒ move head

# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:  
 $\Rightarrow$  change state
- ▶ Cache to program: excitatory  
 $\Rightarrow$  read current symbol
- ▶ Program to symbol: excitatory  
 $\Rightarrow$  write new symbol
- ▶ Program to position: excitatory  
 $\Rightarrow$  move head

# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:
  - ⇒ change state
- ▶ Cache to program: excitatory
  - ⇒ read current symbol
- ▶ Program to symbol: excitatory
  - ⇒ write new symbol
- ▶ Program to position: excitatory
  - ⇒ move head



# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:
  - ⇒ change state
- ▶ Cache to program: excitatory
  - ⇒ read current symbol
- ▶ Program to symbol: excitatory
  - ⇒ write new symbol
- ▶ Program to position: excitatory
  - ⇒ move head

# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:
  - ⇒ change state
- ▶ Cache to program: excitatory
  - ⇒ read current symbol
- ▶ Program to symbol: excitatory
  - ⇒ write new symbol
- ▶ Program to position: excitatory
  - ⇒ move head

# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:
  - ⇒ change state
- ▶ Cache to program: excitatory
  - ⇒ read current symbol
- ▶ Program to symbol: excitatory
  - ⇒ write new symbol
- ▶ Program to position: excitatory
  - ⇒ move head

# PROGRAM RINGS

- ▶ One ring per event “state  $q_i$  & current symbols  $a_1, \dots, a_k$ ”.
- ▶ Program to program: excitatory/inhibitory:
  - ⇒ change state
- ▶ Cache to program: excitatory
  - ⇒ read current symbol
- ▶ Program to symbol: excitatory
  - ⇒ write new symbol
- ▶ Program to position: excitatory
  - ⇒ move head

## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings

## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings

## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings

## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings



## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings

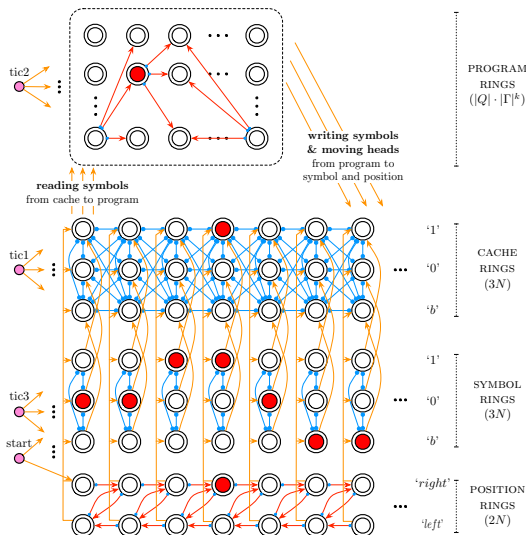
## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings

## START-TIC1-TIC2-TIC3 PROCEDURE

- ▶ “start” cell sets the initial configuration of the tape(s).
- ▶ “tic1” cell triggers the *cache rings update*: reading symbol:
  - ⇒ its excitations combines with those of the position rings and symbol rings
- ▶ “tic2” cell triggers the *program rings update*: switching state:
  - ⇒ its excitations combines with those of other program rings and cache rings
- ▶ “tic3” cell triggers the *tape rings update*: writing symbol and moving heads:
  - ⇒ its excitations combines with those of program rings and position rings

# PUTTING EVERYTHING TOGETHER...



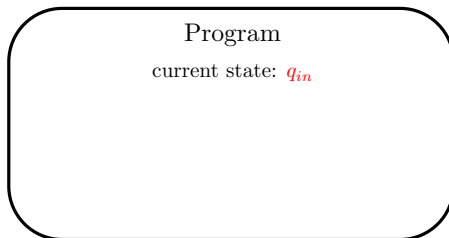
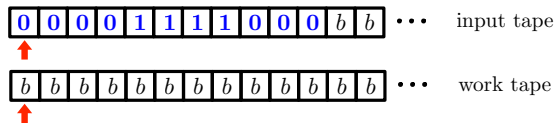
# TURING MACHINES & BOOLEAN RNNs WITH SYNFiRE RINGS

Since the construction is generic, one has the following result:

## THEOREM

*Any Turing machine can be simulated by some Boolean neural network composed of synfire rings (up to assuming that the number of rings can be extended in an unbounded manner).*

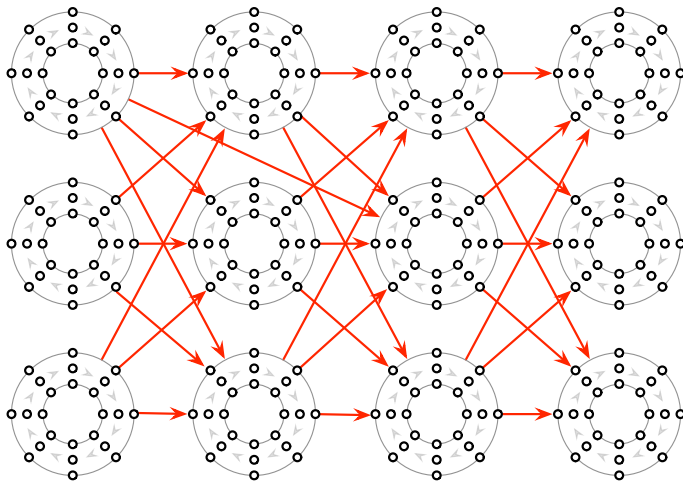
# TURING MACHINE RECOGNIZING THE NON-REGULAR LANGUAGE $L = \{0^n 1^n 0^n : n \geq 0\}$



# SIMULATION OF THE TURING MACHINE

Play movie...

# FUTURE WORK: LEARNING WITHIN THE SYNfire RING ARCHITECTURE





# CONCLUSIONS

- ▶ We introduced a new paradigm of neural computation based on the concept of synfire rings.
- ▶ We intend to study the issue of *learning* within the synfire ring architecture.
- ▶ Towards *neuronal computers*... By growing cultures of neurons according to the synfire ring architecture, one could simulate finite state machines with biological neural networks.

# CONCLUSIONS

- ▶ We introduced a new paradigm of neural computation based on the concept of synfire rings.
- ▶ We intend to study the issue of *learning* within the synfire ring architecture.
- ▶ Towards *neuronal computers*... By growing cultures of neurons according to the synfire ring architecture, one could simulate finite state machines with biological neural networks.

# CONCLUSIONS

- ▶ We introduced a new paradigm of neural computation based on the concept of synfire rings.
- ▶ We intend to study the issue of *learning* within the synfire ring architecture.
- ▶ Towards *neuronal computers*... By growing cultures of neurons according to the synfire ring architecture, one could simulate finite state machines with biological neural networks.