Three small universal spiking neural P systems

Turlough Neary

Institute for Neuroinformatics, University of Zürich and ETH Zürich, Switzerland

Abstract

In this work we give three small spiking neural P systems. We begin by constructing a universal spiking neural P system with extended rules and only 4 neurons. This is the smallest possible number of neurons for a universal system of its kind. We prove this by showing that the set of problems solved by spiking neural P systems with 3 neurons is bounded above by NL, and so there exists no such universal system with 3 neurons. If we generalise the output technique we immediately find a universal spiking neural P system with extended rules that has only 3 neurons. This is also the smallest possible number of neurons for a universal system of its kind. Finally, we give a universal spiking neural P system with standard rules and only 7 neurons. In addition to giving a significant improvement in terms of reducing the number of neurons, our systems also offer an exponential improvement on the time and space overheads of the small universal spiking neural P systems of other authors.

Keywords: universal spiking neural P system, computational complexity, counter machine.

1. Introduction

The search for the simplest Turing universal models of computation helps us to determine the trade-offs between the various parameters of a model that allow computational completeness. In other words, it helps us find the minimum requirements for universality in a model. Since their recent inception, spiking neural P (SN P) systems [1] have been the subject of such endeavors [2, 3, 4, 5].

The search to find the smallest universal SN P systems, where size is the number of neurons, was initiated by Păun and Păun [2]. We begin by solving the problem of finding the smallest possible number of neurons needed to give a universal SN P system with extended rules, one of the open problems given in [6]. To achieve this, we begin by giving a universal extended SN P system that has only 4 neurons. Then, we prove that the set of problems solved by SN P systems with 3 neurons is bounded above by NL, and so there exists no such universal system with 3 neurons. Thus, our 4-neuron system has the smallest possible number of neurons for a universal extended SN P system. We also find that if a generalised output technique is used we can give a universal SN P system with extended rules that has only 3 neurons. This is also the smallest possible number of neurons for a universal

Email address: tneary@ini.phys.ethz.ch (Turlough Neary)

number of neurons	simulation time/space	type of rules	exhaustive use of rules	author
125	double-exponential/ triple-exponential	extended	yes	Zhang et al. [8, 9]
18	polynomial/exponential	extended	yes	Neary [10]
10	linear/exponential	extended	yes	Neary [11]
49	double-exponential	extended	no	Păun and Păun [2]
41	double-exponential	extended	no	Zhang et al. [12]
18	exponential	extended	no	Neary [13, 14]†
12	double-exponential	extended	no	Neary [13]
9	double-exponential	extended	no	Zeng et al. [15]
4	**exponential**	**extended**	**no**	**Section 4***
3	**exponential**	**extended**	**no**	**Section 5*‡**
84	double-exponential	standard	no	Păun and Păun [2]
67	double-exponential	standard	no	Zhang et al. [12]
17	exponential	standard	no	Neary [16]
11	exponential	standard	**no**	Neary [17]
7	**exponential**	**standard**	**no**	**Section 6**

Table 1: Small universal SN P systems. The "simulation time/space" column gives the overheads used by each system when simulating a standard single tape Turing machine. † The 18 neuron system is not explicitly given in [13]; it was presented at [14] and is easily derived from the other system in [13]. ‡ A more generalised output technique is used. * The smallest possible number of neurons for a universal system of its kind. Further explanation of the time/space complexity overheads and comparisons between some of the systems in this table can be found in Section 3.

system of its kind. The results for the extended model previously appeared in [7]. For the standard model, finding a universal SN P system with a small number of neurons is made difficult due to the very limited way in which the neurons can communicate with each other. Here we give a universal SN P system for the standard model that has only 7 neurons. This represents a significant improvement with the lowest number of neurons used to give such a universal system by any other author standing at 67.

The systems we present in this work offer an exponential improvement on the time and space overheads of the small universal SN P systems of other authors [2, 12, 15]. Our systems simulate Turing machines with exponential time and space overheads whereas the systems in [2, 12, 15] simulate Turing machines with double-exponential time and space overheads. In other works [10, 11], it is shown that universal SN P systems require exponential time to simulate Turing machines, and so significant improvement on the time and space overheads of our systems is not possible. Table 1 gives the state of the art in small universal SN P systems and their respective simulation time and space overheads[1].

[1]In a similar table given in [7] the time/space complexity for a number of the systems is incorrect due to an error copied from Korec's paper [18]. For more see Section 3.

Păun and Păun [2] state that a significant decrease on the number of neurons of their two universal SN P systems is improbable (also stated in [6]). The huge reduction in the number of neurons that is given by Theorems 1 and 4 is in part due to the method we use to encode the instructions of the counter machines being simulated. With the exception of the 18 and 10-neurons systems with exhaustive use of rules, all of the SN P systems given in Table 1 simulate counter machines. The number of neurons in previous small universal systems [2, 12] were dependent on the number of instructions in the counter machine being simulated. In our systems each unique counter machine instruction is encoded as a unique number of spikes, and thus the number of neurons in our SN P systems is independent of the number of counter machine instructions. The technique of encoding the instructions as spikes was first used to construct small universal SN P systems in [13] (see Table 1). Another reason for the small number of neurons in our systems is that we have done away with the need for a dedicated input module as the neurons that deal with the input from the environment also perform other functions.

2. SN P Systems

Definition 1 (Spiking neural P system). *A spiking neural P system (SN P system) is a tuple* $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$*, where:*

1. $O = \{s\}$ *is the unary alphabet (s is known as a spike),*
2. $\sigma_1, \sigma_2, \cdots, \sigma_m$ *are neurons, of the form* $\sigma_i = (n_i, R_i), 1 \leqslant i \leqslant m$*, where:*
   (a) $n_i \geqslant 0$ *is the initial number of spikes contained in* σ_i*,*
   (b) R_i *is a finite set of rules of the following two forms:*
      i. $E/s^b \rightarrow s; d$*, where* E *is a regular expression over* s*,* $b \geqslant 1$ *and* $d \geqslant 0$*,*
      ii. $s^e \rightarrow \lambda$*, where* λ *is the empty word,* $e \geqslant 1$*, and for all* $E/s^b \rightarrow s; d$ *from* R_i $s^e \notin L(E)$ *where* $L(E)$ *is the language defined by* E*,*
3. $syn \subseteq \{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\}$ *is the set of synapses between neurons, where* $i \neq j$ *for all* $(i, j) \in syn$*,*
4. $in, out \in \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$ *are the input and output neurons, respectively.*

A firing rule $r = E/s^b \rightarrow s; d$ is applicable in a neuron σ_i if there are $j \geqslant b$ spikes in σ_i and $s^j \in L(E)$ where $L(E)$ is the set of words defined by the regular expression E. If, at time t, rule r is executed then b spikes are removed from the neuron, and at time $t + d$ the neuron fires. When a neuron σ_i fires a spike is sent to each neuron σ_j for every synapse (i, j) in Π. Also, the neuron σ_i remains closed and does not receive spikes until time $t + d$ and no other rule may execute in σ_i until time $t+d+1$. A forgetting rule $r' = s^e \rightarrow \lambda$ is applicable in a neuron σ_i if there are exactly e spikes in σ_i. If r' is executed then e spikes are removed from the neuron. At each timestep t a rule must be applied in each neuron if there is one or more applicable rules. Thus, while the application of rules in each individual neuron is sequential the neurons operate in parallel with each other.

Note from 2b(i) of Definition 1 that there may be two rules of the form $E/s^b \rightarrow s; d$, that are applicable in a single neuron at a given time. If this is the case then the next rule to execute is chosen non-deterministically.

An *extended* SN P system [2] has more general rules of the form $E/s^b \to s^p; d$, where $b \geqslant p \geqslant 1$. The application of this rule sends p spikes through each of the outgoing synapses of a neuron. The SN P systems we present in this work use rules without delay, and thus in the sequel we write rules as $E/s^b \to s^p$. Also, if $E = s^b$ then we write the rule as $s^b \to s^p$.

Spikes are introduced into the system from the environment by reading in a binary sequence (or word) $w \in \{0,1\}^*$ via the input neuron σ_1. The sequence w is read from left to right one symbol at each timestep and a spike enters the input neuron from the environment on a given timestep iff the read symbol is 1. In general the output of an SN P system is a spike train [19] (a binary sequence that is determined by whether or not the output neurons fires at each timestep). Unless noted otherwise, we will assume that the output of an SN P system Π is interpreted as the time interval between the first and second timesteps a firing rule is applied in the output neuron. Given the input sequence w the value of this (output) interval is denoted $\Pi(w) \in \mathbb{N}$. This output technique is used by most of the systems given in Table 1.

Note from Definition 1 that we could look at other parameters when considering the size of universal SN P systems. For example the total number of synapses in the system. The 4 and 7-neuron systems we give in this work have much smaller numbers of synapses than the systems in [2, 12, 15]. As another example, one could also consider how many different types of rules are used by the system and the complexity of the rules. The systems in [2, 12] use a small number of rule types all of which are quite simple. The systems in [7, 15, 16] and in this work use a larger number of rule types and the size of some of these rules is dependent on the number of instructions of the *universal* counter machine that is simulated. However, in another sense our 17 and 7-neuron systems use a more restricted form of rule than the other *standard* SN P systems in Table 1 as they use rules without delay. In another simplification, a universal SN P system was given in [20] with only one type of neuron, in other words each neuron in the system had the same set of rules. For the simplification of other aspects of SN P systems see [3, 4, 5], where the authors investigate the computational power of simplified forms of SN P systems.

3. Counter machines, universality and time/space complexity

Definition 2. *A counter machine is a tuple $C = (z, R, c_m, Q, q_1, q_h)$, where z gives the number of counters, R is the set of input counters, c_m is the output counter, $Q = \{q_1, q_2, \cdots, q_h\}$ is the set of instructions, and $q_1, q_h \in Q$ are the initial and halt instructions, respectively.*

Each counter c_j stores a natural number value $v_j \geqslant 0$. Each instruction $q_i \neq q_h$ is of one of the following two forms:

- $q_i : INC(j), q_l$ increment the value v_j stored in counter c_j by 1 and move to instruction q_l.
- $q_i : DEC(j), q_l, q_k$ if the value v_j stored in counter c_j is greater than 0 then decrement this value by 1 and move to instruction q_l, otherwise if $v_j = 0$ move to instruction q_k.

At the beginning of a computation the first instruction executed is q_1, and the input to the counter machine is stored in the input counters with all other counters having an initial value of 0. If the counter machine's control enters instruction q_h, then the computation halts at that timestep. The result of the computation is the value v_m stored in the output counter c_m when the computation halts.

We now consider the universal counter machines of Korec [18] and his definitions of universality in counter machines.

Definition 3 (Korec [18]). *A counter machine M will be called strongly universal if there is a recursive function g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = \Phi^2_M(g(x), y)$.*

Here ϕ_x is the x^{th} unary partial recursive function in a Gödel enumeration of all unary partial recursive functions. Also, $\Phi^2_M(g(x), y)$ is the value stored in the output counter at the end of a computation when M is started with the values $g(x)$ and y in its input counters. If we consider computing an n-ary function with a Korec-strong universal counter machine then it is clear that n arguments must be encoded as a single input y. For this reason, it is more appropriate to refer to Korec's definition as strongly universal for unary partial recursive functions instead of simply strongly universal. In fact, towards the end of his paper Korec [18] sketches how to construct simple counter machines that are strongly universal for n-ary partial recursive functions. It is worth noting that Korec's definition of strong universality is not concerned with the time/space efficiency of the computation. As we will see later, the strongly universal counter machines of Korec given in [18] suffer exponential slowdown when simulating 3-counter machines.

In [18] Korec also gives a number of other definitions of universality. If the equation $\phi_x(y) = \Phi^2_M(g(x), y)$ in Definition 3 above is replaced with any one of the equations $\phi_x(y) = \Phi^1_M(g_2(x, y))$, $\phi_x(y) = f(\Phi^2_M(g(x), y))$ or $\phi_x(y) = f(\Phi^1_M(g_2(x, y)))$ then Korec refers to counter machine M as weakly universal. Korec gives another definition where the equation $\phi_x(y) = \Phi^2_M(g(x), y)$ in Definition 3 is replaced with the equation $\phi_x(y) = f(\Phi^2_M(g(x), e(y)))$. However, he does not include this definition in his list of weakly universal machines even though the equation $\phi_x(y) = f(\Phi^2_M(g(x), e(y)))$ allows for a more relaxed encoding than the equation $\phi_x(y) = f(\Phi^2_M(g(x), y))$ and thus gives a weaker form of universality.

In [2] Korec's notion of strong universality was adopted for SN P systems[2] as follows: An SN P system Π is strongly universal if $\Pi(10^{g(x)-1}10^{y-1}1) = \phi_x(y)$ for all x and y (here if $\phi_x(y)$ is undefined so too is $\Pi(10^{g(x)-1}10^{y-1}1)$). Korec noted that his reason for distinguishing strong universality for counter machines was that because they take natural numbers as input, no encoding was necessary when computing unary partial recursive functions. This is clearly not the case for SN P systems as some encoding will always be necessary when computing such functions. For this and the other reasons mentioned above, we rely on time/space complexity analysis to compare small SN P systems and their encodings (see Table 1).

Despite the above critic, it still remains of interest to compare the input encodings used by the various small universal SN P systems. It would be of interest to find if there is any trade-off between the size of universal SN P systems and the

[2]Note that a formal definition of this notion is not explicitly given in [2].

complexity of the input encoding. For example, if we place more restrictions on the input encoding will the number of neurons required to give a universal SN P system increase? Here we give no definitive answers in this respect, but we can use the notions already discussed to give somewhat simplistic comparisons. In Table 1, the 49, 41, and 18-neuron extended systems also satisfy the notion of strong universality mentioned in the previous paragraph. Surprisingly, our extended system with 4 neurons given in Theorem 1 also satisfies this notion of strong universality. The 84, 67, and 17-neuron systems from Table 1 once again satisfy the above mentioned notion of strong universality. The 7-neuron system we give in Theorem 4 takes the sequence $10^{2hx}10^{2hy}1$ as input (where h is a constant) and thus does not satisfy the above notion of strong universality. It is worth noting that this system significantly reduces the number of neurons used to give such a universal system even though it uses only a slight generalisation of the input encoding.

The below definition of universality allows for recursive encoding and decoding functions. It is common to allow such encoding and decoding functions in definitions of universality (for example see [21, 22]).

Definition 4. *An SN P system* Π *is universal if there are recursive functions* g *and* f *such that for all* $x, y \in N$ *for which* $\phi_x(y)$ *is defined we have* $\phi_x(y) = f(\Pi(g(x, y)))$.

In Definition 4 the function g maps the pair (x, y) to a binary input sequence to be read into Π. Also, for all values of x and y for which $\phi_x(y)$ is defined, $\Pi(g(x, y))$ gives the output sequence of Π when started on the input $g(x, y)$, and if $\phi_x(y)$ is undefined so too is $\Pi(g(x, y))$. Finally, the function f maps the output sequence $\Pi(g(x, y))$ to a natural number. With the exception of the 18 and 10-neuron systems with exhaustive use of rules, all of the SN P systems in Table 1 satisfy Definition 4. The 12 and 9-neuron systems in Table 1 are the only systems that (using current algorithms) must encode x and y together. The other systems in Table 1 may use encoding functions that encode x and y separately. Also, excluding the 18-neuron system with exhaustive use of rules and the 12 and 9-neuron systems, all of the other systems in Table 1 use input and output encodings that are linear in y and $\phi_x(y)$, respectively.

For the purposes of explanation, in this work we say that an abstract machine M' simulates another abstract machine M if $M(w) = g(M'(f(w)))$, where $M(w)$ is the output of M when started on the input w, $M'(f(w))$ is the output of M' when started on the input $f(w)$, f and g are the encoding and decoding functions, and if $M(w)$ is undefined so too is $M'(f(w))$. Note that we assume that appropriate restrictions are placed on f and g so that they are not too powerful (for example, if M is universal then we insist that they are total recursive). In addition to the above, we will say that M' simulates M with an exponential time (space) overhead if for all values of w for which M is defined, M' completes its computation in time (space) $O(2^p)$, where $p = u^k$, k is a constant and u is the time (space) used by M to complete its computation. Linear $O(u)$, polynomial $O(u^k)$, double-exponential $O(2^{2^p})$ and triple-exponential $O(2^{2^{2^p}})$ simulation overheads are defined in a similar manner. In this work the space used by an SN P system is the maximum number of spikes in the system at any timestep during its computation.

In [18] Korec gives a number of universal counter machines that use very few instructions. At the end of his paper Korec states that his universal counter machine

with 32 instructions simulates R3-machines in linear time. This is incorrect and is possibly a typographical error (he may have intended to write "R3a-machines" instead of "R3-machines"). His 32-instruction machine simulates R3a-machines with a linear time overhead, and his R3a-machines simulate counter machines with an exponential time overhead. To see this note that he proves R3a-machines universal by showing that they compute unary partial recursive functions as follows: Step 1. The R3a-machine computes 2^y from its initial input value y. Step 2. The R3a-machine computes the value $2^{f(y)}$ using only 2 of its 3 counters. Step 3. The R3a-machine computes the output $f(y)$ from $2^{f(y)}$. Using current algorithms 2-counter machines are exponentially slower than 3-counter machines [23], and so because of Step 2 above, Korec's R3a-machines and 32-instruction machine are exponentially slower than 3-counter machines. It is known that counter machines simulate Turing machines with an exponential time overhead [24], and thus Korec's 32-instruction machine simulates Turing machines in double-exponential time. All of the SN P systems in [2, 12] simulate the 23-instruction (the halt instruction is included here) universal counter machine given by Korec in [18]. This 23-instruction machine uses the same algorithm as the 32-instruction machine, and thus also suffers from a double-exponential time overhead when simulating Turing machines. The SN P systems given in [2, 12] simulate Korec's 23-instruction machine with linear time and space overheads, and so have double-exponential time and space overheads when simulating Turing machines. The SN P system given in [8] simulates Korec's 23-instruction machine with linear time and exponential space overheads, and thus has double-exponential time and triple-exponential space overheads when simulating Turing machines. We end our complexity analysis by noting that many of the above simulation overheads (including those of Korec's small counter machines) could be exponentially improved by showing that Korec's R3a-machines simulate 3-counter machines in polynomial time.

The 12 and 9-neuron systems in Table 1 are proved universal by showing that they simulate 2-counter machines in linear time and thus have double exponential time and space overheads when simulating Turing machines. The systems we give in this work are proved universal by showing that they simulate 3-counter machines in linear time and thus have exponential time and space overheads when simulating Turing machines.

4. A Small Universal Extended SN P System

Theorem 1. *Let C be a universal counter machine with 3 counters that completes its computation in time t. Then there is a universal extended SN P system Π_C that simulates the computation of C in time $O(t)$ and has only 4 neurons.*

Proof. Let $C = (3, \{c_1, c_2\}, c_3, Q, q_1, q_h)$ where $Q = \{q_1, q_2, \cdots, q_h\}$. Our SN P system Π_C is given by Figure 1 and Table 2. In each neuron σ_i of Π_C, for every possible pair of rules $E/s^b \to s^p$ and $E'/s^{b'} \to s^{p'}$ in R_i, the sets $L(E)$ and $L(E')$ are disjoint. This property ensures that the operation of Π_C is deterministic as there is no more than one rule applicable in each neuron at a given timestep. For example, in Table 2 we have $i < h$ giving $6(h + i) < 12h$, which means that this disjoint property holds for each rule when simulating an arbitrary counter machine instruction q_i. Given the values $1 \leqslant i < h$, $1 \leqslant l \leqslant h$ and $1 \leqslant k \leqslant h$, it is a straightforward mater to verify from Table 2 that the disjoint property holds for

neuron	rules
σ_1	$(s^{12h})^* s^{6h+8} / s^{6h} \to s^{6h}$, $(s^{12h})^* s^{6h+9} / s^{9h+5} \to s^{3h+2}$, $(s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}$, if $[q_i : INC(1), q_l]$, $[q_i : DEC(1), q_l, q_k] \notin \{Q\}$ $(s^{12h})^* s^{6(h+i)+4} / s^{12h+6i+4-6l} \to s^{6l}$, if $[q_i : INC(1), q_l] \in \{Q\}$ $(s^{12h})^* s^{54h+6i+4} / s^{36h+6i+4-6l} \to s^{6l}$, if $[q_i : DEC(1), q_l, q_k] \in \{Q\}$ $s^{42h+6i+4} / s^{24h+6i+4-6k} \to s^{6k}$, if $[q_i : DEC(1), q_l, q_k] \in \{Q\}$
σ_2	$s^{18h+7} / s^{6h} \to s^{6h}$, $(s^{12h})^* s^{6h+9} / s^{9h+5} \to s^{3h+2}$, $(s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}$, if $[q_i : INC(2), q_l]$, $[q_i : DEC(2), q_l, q_k] \notin \{Q\}$ $(s^{12h})^* s^{6(h+i)+4} / s^{12h+6i+4-6l} \to s^{6l}$, if $[q_i : INC(2), q_l] \in \{Q\}$ $(s^{12h})^* s^{54h+6i+4} / s^{36h+6i+4-6l} \to s^{6l}$, if $[q_i : DEC(2), q_l, q_k] \in \{Q\}$ $s^{42h+6i+4} / s^{24h+6i+4-6k} \to s^{6k}$, if $[q_i : DEC(2), q_l, q_k] \in \{Q\}$
σ_3	$s^{18h+7} / s^{6h} \to s^{6h}$, $(s^{12h})^* / s^{12h+1} \to s^{12h}$, $s^{18h+2} / s^{6h} \to s^{6h}$, $s^{24h-1} \to s^{12h}$, $s^{18h+3} / s^{6h+1} \to s^{6h+1}$, $s^{24h-1} \to s^{12h}$, $s^{18h+8} / s^{6h+6} \to s^{6h+1}$, $(s^{12h})^* s^{36h-1} / s^{12h} \to s^{6h}$, $(s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}$, if $[q_i : INC(3), q_l]$, $[q_i : DEC(3), q_l, q_k] \notin \{Q\}$ $(s^{12h})^* s^{6(h+i)+4} / s^{12h+6i+4-6l} \to s^{6l}$, if $[q_i : INC(3), q_l] \in \{Q\}$ $(s^{12h})^* s^{54h+6i+4} / s^{36h+6i+4-6l} \to s^{6l}$, if $[q_i : DEC(3), q_l, q_k] \in \{Q\}$ $s^{42h+6i+10} / s^{24h+6i+4-6k} \to s^{6k}$, if $[q_i : DEC(3), q_l, q_k] \in \{Q\}$
σ_4	$s^{6h+1} \to \lambda$, $s^{12h+2} \to \lambda$, $s^{6l} \to \lambda$, $s^{12h} \to s$

Table 2: Rules for each of the neurons of Π_C. Here $1 \leqslant i < h$, $1 \leqslant l \leqslant h$ and $1 \leqslant k \leqslant h$.

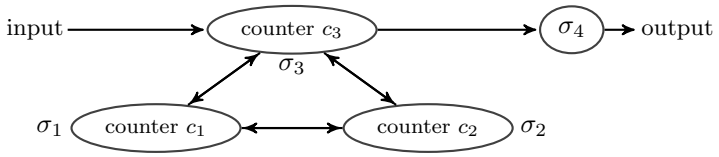

Figure 1: Universal extended SN P system Π_C. Each oval labeled σ_i is a neuron. An arrow going from neuron σ_i to neuron σ_j illustrates a synapse (i, j).

each neuron in Π_C. Here the variables i, j and k come from the two types of counter machine instruction given immediately after Definition 2.

4.0.1. Encoding of a configuration of C and reading input into Π_C

A configuration of C is stored as spikes in the neurons of Π_C. The next instruction q_i to be executed is stored in each of the neurons σ_1, σ_2 and σ_3 as $6(h + i)$ spikes. Let x_1, x_2 and x_3 be the values stored in counters c_1, c_2 and c_3, respectively. Then the values x_1, x_2 and x_3 are stored as $12h(x_1+1)$, $12h(x_2+1)$ and $12h(x_3+1)$ spikes in neurons σ_1, σ_2 and σ_3, respectively. The input to Π_C is read into the system via the input neuron σ_3 (see Figure 1). If C begins its computation with the values x_1 and x_2 in counters c_1 and c_2, respectively, then the binary sequence $w = 10^{x_1-1}10^{x_2-1}1$ is read in via the input neuron σ_3. Thus, σ_3 receives a single spike from the environment at times t_1, t_{x_1+1} and $t_{x_1+x_2+1}$. We explain how the

system is initialised to encode an initial configuration of C by giving the number of spikes in each neuron and the rule that is to be applied in each neuron at time t. Before the computation begins neuron σ_1 contains $6h+7$ spikes, σ_2 contains $18h+7$ spikes, σ_3 contains $18h+6$ spikes and σ_4 contains no spikes. Thus, when σ_3 receives its first spike at time t_1 we have

$$\begin{array}{lll} t_1: & \sigma_1 : 6h+7, & \\ & \sigma_2, \sigma_3 : 18h+7, & s^{18h+7}/s^{6h} \to s^{6h}. \end{array}$$

where on the left $\sigma_j : v$ gives the number v of spikes in neuron σ_j at time t and on the right is the rule that is to be applied at time t, if there is an applicable rule at that time. Thus, from Figure 1, when we apply the rule $s^{18h+7}/s^{6h} \to s^{6h}$ in neurons σ_2 and σ_3 at time t_1 we get

$$\begin{array}{lll} t_2: & \sigma_1 : 18h+7, & \\ & \sigma_2, \sigma_3 : 18h+7, & s^{18h+7}/s^{6h} \to s^{6h}, \\ & \sigma_4 : 6h, & s^{6h} \to \lambda, \\ & & \\ t_3: & \sigma_1 : 30h+7, & \\ & \sigma_2, \sigma_3 : 18h+7, & s^{18h+7}/s^{6h} \to s^{6h}, \\ & \sigma_4 : 6h, & s^{6h} \to \lambda. \end{array}$$

Neurons σ_2 and σ_3 send $12h$ spikes to neuron σ_1 on each timestep between times t_1 and t_{x_1+1}. This gives a total of $12hx_1$ spikes sent to σ_1 during the time interval t_1 to t_{x_1+1}. Thus when σ_3 receives the second spike from the environment we get

$$\begin{array}{lll} t_{x_1+1}: & \sigma_1 : 12hx_1+6h+7, & \\ & \sigma_2 : 18h+7, & s^{18h+7}/s^{6h} \to s^{6h}, \\ & \sigma_3 : 18h+8, & s^{18h+8}/s^{6h+6} \to s^{6h+1}, \\ & \sigma_4 : 6h, & s^{6h} \to \lambda, \\ & & \\ t_{x_1+2}: & \sigma_1 : 12h(x_1+1)+6h+8, & (s^{12h})^* s^{6h+8}/s^{6h} \to s^{6h}, \\ & \sigma_2 : 18h+8, & \\ & \sigma_3 : 18h+2, & s^{18h+2}/s^{6h} \to s^{6h}, \\ & \sigma_4 : 6h+1, & s^{6h+1} \to \lambda, \\ & & \\ t_{x_1+3}: & \sigma_1 : 12h(x_1+1)+6h+8, & (s^{12h})^* s^{6h+8}/s^{6h} \to s^{6h}, \\ & \sigma_2 : 30h+8, & \\ & \sigma_3 : 18h+2, & s^{18h+2}/s^{6h} \to s^{6h}, \\ & \sigma_4 : 6h, & s^{6h} \to \lambda. \end{array}$$

Neurons σ_1 and σ_3 fire on every timestep between times t_{x_1+2} and $t_{x_1+x_2+2}$ to send a total of $12hx_2$ spikes to σ_2. Thus, when σ_3 receives the last spike from the

environment we have

$$
\begin{array}{lll}
t_{x_1+x_2+1}: & \sigma_1: 12h(x_1+1)+6h+8, & (s^{12h})^* s^{6h+8}/s^{6h} \to s^{6h}, \\
& \sigma_2: 12hx_2+6h+8, & \\
& \sigma_3: 18h+3, & s^{18h+3}/s^{6h+1} \to s^{6h+1}, \\
& \sigma_4: 6h, & s^{6h} \to \lambda, \\
\\
t_{x_1+x_2+2}: & \sigma_1: 12h(x_1+1)+6h+9, & (s^{12h})^* s^{6h+9}/s^{9h+5} \to s^{3h+2}, \\
& \sigma_2: 12h(x_2+1)+6h+9, & (s^{12h})^* s^{6h+9}/s^{9h+5} \to s^{3h+2} \\
& \sigma_3: 18h+2, & s^{18h+2}/s^{6h} \to s^{6h}, \\
& \sigma_4: 6h+1, & s^{6h+1} \to \lambda, \\
\\
t_{x_1+x_2+3}: & \sigma_1: 12h(x_1+1)+6(h+1), & \\
& \sigma_2: 12h(x_2+1)+6(h+1), & \\
& \sigma_3: 12h+6(h+1). &
\end{array}
$$

At time $t_{x_1+x_2+3}$ neuron σ_1 contains $12h(x_1+1)+6(h+1)$ spikes, σ_2 contains $12h(x_2+1)+6(h+1)$ spikes and σ_3 contains $12h+6(h+1)$ spikes. Thus at time $t_{x_1+x_2+3}$ the SN P system encodes an initial configuration of C.

4.1. Π_C *simulating* $q_i : INC(1), q_l$

Let counters c_1, c_2, and c_3 have values x_1, x_2, and x_3, respectively. Then the simulation of $q_i : INC(1)$ begins at time t_j with $12h(x_1+1)+6(h+i)$ spikes in σ_1, $12h(x_2+1)+6(h+i)$ spikes in σ_2 and $12h(x_3+1)+6(h+i)$ spikes in σ_3. Thus, at time t_j we have

$$
\begin{array}{lll}
t_j: & \sigma_1: 12h(x_1+1)+6(h+i), & \\
& \sigma_2: 12h(x_2+1)+6(h+i), & (s^{12h})^* s^{6(h+i)}/s^{12h+6i+2} \to s^{12h+2}, \\
& \sigma_3: 12h(x_3+1)+6(h+i), & (s^{12h})^* s^{6(h+i)}/s^{12h+6i+2} \to s^{12h+2}.
\end{array}
$$

From Figure 1, when we apply the rule $(s^{12h})^* s^{6(h+i)}/s^{12h+6i+2} \to s^{12h+2}$ in neurons σ_2 and σ_3 at time t_j we get

$$
\begin{array}{lll}
t_{j+1}: & \sigma_1: 12h(x_1+3)+6(h+i)+4, & (s^{12h})^* s^{6(h+i)+4}/s^{12h+6i+4-6l} \to s^{6l}, \\
& \sigma_2: 12h(x_2+1)+6h, & \\
& \sigma_3: 12h(x_3+1)+6h, & \\
& \sigma_4: 12h+2, & s^{12h+2} \to \lambda, \\
\\
t_{j+2}: & \sigma_1: 12h(x_1+2)+6(h+l), & \\
& \sigma_2: 12h(x_2+1)+6(h+l), & \\
& \sigma_3: 12h(x_3+1)+6(h+l). &
\end{array}
$$

At time t_{j+2} the simulation of $q_i : INC(1), q_l$ is complete. Note that an increment on the value x_1 in counter c_1 was simulated by increasing the $12h(x_1+1)$ spikes in

σ_1 to $12h(x_1+2)$ spikes. Note also that the encoding $6(h+l)$ of the next instruction q_l has been established in neurons σ_1, σ_2 and σ_3.

4.2. Π_C *simulating* $q_i : DEC(1), q_l, q_k$

There are two cases to consider here. Case 1: if counter c_1 has value $x_1 > 0$, then decrement c_1 and move to instruction q_l. Case 2: if counter c_1 has value $x_1 = 0$, then move to instruction q_k. As with the previous example, our simulation begins at time t_j. Thus Case 1 ($x_1 > 0$) gives

$$
\begin{array}{lll}
t_j: & \sigma_1 : 12h(x_1+1)+6(h+i), & \\
& \sigma_2 : 12h(x_2+1)+6(h+i), & (s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}, \\
& \sigma_3 : 12h(x_3+1)+6(h+i), & (s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}, \\
& & \\
t_{j+1}: & \sigma_1 : 12h(x_1+3)+6(h+i)+4, & (s^{12h})^* s^{54h+6i+4} / s^{36h+6i+4-6l} \to s^{6l}, \\
& \sigma_2 : 12h(x_2+1)+6h, & \\
& \sigma_3 : 12h(x_3+1)+6h, & \\
& \sigma_4 : 12h+2, & s^{12h+2} \to \lambda, \\
& & \\
t_{j+2}: & \sigma_1 : 12hx_1+6(h+l), & \\
& \sigma_2 : 12h(x_2+1)+6(h+l), & \\
& \sigma_3 : 12h(x_3+1)+6(h+l). &
\end{array}
$$

At time t_{j+2} the simulation of $q_i : DEC(1), q_l, q_k$ for Case 1 ($x_1 > 0$) is complete. Note that a decrement on the value x_1 in counter c_1 was simulated by decreasing the $12h(x_1+1)$ spikes in σ_1 to $12hx_1$ spikes. Note also that the encoding $6(h+l)$ of the next instruction q_l has been established in neurons σ_1, σ_2 and σ_3. Alternatively, if we have Case 2 ($x_1 = 0$) then we get

$$
\begin{array}{lll}
t_j: & \sigma_1 : 12h+6(h+i), & \\
& \sigma_2 : 12h(x_2+1)+6(h+i), & (s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}, \\
& \sigma_3 : 12h(x_3+1)+6(h+i), & (s^{12h})^* s^{6(h+i)} / s^{12h+6i+2} \to s^{12h+2}, \\
& & \\
t_{j+1}: & \sigma_1 : 42h+6i+4, & s^{42h+6i+4} / s^{24h+6i+4-6k} \to s^{6k}, \\
& \sigma_2 : 12h(x_2+1)+6h, & \\
& \sigma_3 : 12h(x_3+1)+6h, & \\
& \sigma_4 : 12h+2, & s^{12h+2} \to \lambda, \\
& & \\
t_{j+2}: & \sigma_1 : 12h+6(h+k), & \\
& \sigma_2 : 12h(x_2+1)+6(h+k), & \\
& \sigma_3 : 12h(x_3+1)+6(h+k). &
\end{array}
$$

At time t_{j+2} the simulation of $q_i : DEC(1), q_l, q_k$ for Case 2 is complete. The encoding $6(h+k)$ of the next instruction q_k has been established in σ_1, σ_2 and σ_3.

4.2.1. Halting

The halt instruction q_h is encoded as $12h$ spikes. Thus if C halts we get

$$\begin{array}{lll} t_j: & \sigma_1: 12h(x_1+2), & \\ & \sigma_2: 12h(x_2+2), & \\ & \sigma_3: 12h(x_3+2), & (s^{12h})^*/s^{12h+1} \to s^{12h}, \\ & & \\ t_{j+1}: & \sigma_1: 12h(x_1+3), & \\ & \sigma_2: 12h(x_2+3), & \\ & \sigma_3: 12h(x_3+1)-1, & (s^{12h})^* s^{36h-1}/s^{12h} \to s^{6h}, \\ & \sigma_4: 12h, & s^{12h} \to s, \\ & & \\ t_{j+2}: & \sigma_1: 12h(x_1+3)+6h, & \\ & \sigma_2: 12h(x_2+3)+6h, & \\ & \sigma_3: 12hx_3-1, & (s^{12h})^* s^{36h-1}/s^{12h} \to s^{6h}, \\ & \sigma_4: 6h, & s^{6h} \to \lambda. \end{array}$$

The rule $(s^{12h})^* s^{36h-1}/s^{12h} \to s^{6h}$ is applied a further $x_3 - 2$ times in σ_3 to give

$$\begin{array}{lll} t_{j+x_3}: & \sigma_1: 12h(x_1+3)+6h(x_3-1), & \\ & \sigma_2: 12h(x_2+3)+6h(x_3-1), & \\ & \sigma_3: 24h-1, & s^{24h-1} \to s^{12h}, \\ & \sigma_4: 6h, & s^{6h} \to \lambda, \\ & & \\ t_{j+x_3+1}: & \sigma_1: 12h(x_1+4)+6h(x_3-1), & \\ & \sigma_2: 12h(x_2+4)+6h(x_3-1), & \\ & \sigma_4: 12h, & s^{12h} \to s. \end{array}$$

As usual the output is the time interval between the first and second timesteps when a firing rule is applied in the output neuron. Note from above that the output neuron σ_4 fires for the first time at timestep t_{j+1} and for the second time at timestep t_{j+x_3+1}. Thus, the output of Π_C is x_3, the value of the output counter c_3 when C enters the halt instruction q_h. Note that if $x_3 = 0$ then the rule $s^{24h-1} \to s^{12h}$ can not be executed as there are only $12h-1$ spikes in σ_3 at timestep t_{j+1}. Thus if $x_3 = 0$ the output neuron will fire only once. It is also worth noting that no rule is applicable in any neuron after time t_{j+x_3+1} so the entire system halts.

We have shown how to simulate arbitrary instructions of the form $q_i : INC(1), q_l$ and $q_i : DEC(1), q_l, q_k$ that operate on counter c_1. Instructions which operate on counters c_2 and c_3 are simulated in a similar manner. Immediately following the simulation of an instruction Π_C is configured to simulate the next instruction. Each instruction of C is simulated in 2 timesteps. The pair of input values (x_1, x_2) is read into the system in $x_1 + x_2 + 3$ timesteps and sending the output value x_3 out of the system takes $x_3 + 1$ timesteps. Thus, if C completes it computation in time t, then Π_C simulates the computation of C in time $O(t)$. □

The algorithm for our system in Theorem 1 can easily be adapted to simulate non-deterministic 3-counter machines where we have increment instructions of the form $q_i : INC(j), q_l, q_k$ (instead of $q_i : INC(j), q_l$). After incrementing counter j by 1, each increment instruction makes a non-deterministic choice between q_l and q_k as to which instruction is executed next. Simulating such a 3-counter machine with a non-deterministic version of our system only requires one extra rule for each increment instruction. For example, each instruction of the form $q_i : INC(1), q_l, q_k$ would be encoded as the rules $(s^{12h})^* s^{6(h+i)+4} / s^{12h+6i+4-6l} \rightarrow s^{6l}$ and $(s^{12h})^* s^{6(h+i)+4} / s^{12h+6i+4-6k} \rightarrow s^{6k}$ in σ_1 of Table 1. To see that this single change is sufficent to allow the simulation of non-deterministic 3-counter machines, note from Section 4.1 that at timestep t_{j+1} a single neuron (in this case neuron σ_1) decides which instruction will be executed next, and so a non-deterministic choice is made in only one neuron with the operation of the other neurons independent of the non-deterministic choice made.

5. Lower Bounds for Small Universal SN P Systems

In this and other work [2, 12] on small SN P systems the input neuron only receives a constant number of spikes from the environment and the output neuron fires no more than a constant number of times. Hence, we call the input standard if the input neuron receives no more than x spikes from the environment, where x is a constant independent of the input (i.e. the number of 1s in its input sequence is $< x$). Similarly, we call the output standard if the output neuron fires no more than y times, where y is a constant independent of the input. Here we say an SN P system has generalised input if the input neuron is permitted to receive $\leqslant n$ spikes from the environment where $n \in \mathbb{N}$ is the length of its input sequence.

Theorem 2. *Let Π be any extended SN P system with only 3 neurons, generalised input and standard output. Then there is a non-deterministic Turing machine T_Π that simulates the computation of Π in space $O(\log n)$ where n is the length of the input to Π.*

Proof. Let Π be any extended SN P system with generalised input, standard output, and neurons σ_1, σ_2 and σ_3. Also, let y be the maximum number of times the output neuron σ_3 is permitted to fire and let q and r be the maximum value for b and p respectively, for all $E/s^b \rightarrow s^p; d$ in Π.

We begin by explaining how the activity of σ_3 may be simulated using only the states of T_Π (i.e. no workspace is required to simulate σ_3). Recall that the applicability of each rule is determined by a regular expression over a unary alphabet. We can give a single regular expression R that is the union of all the regular expressions for the firing rules of σ_3. This regular expression R determines whether or not there is any applicable rule in σ_3 at each timestep. Figure 2 gives the deterministic finite automaton G that accepts $L(R)$, the language generated by R. If $L(R)$ is finite, we only need to store a constant number of spikes to determine the behaviour of σ_3. To see this note that when the number of spikes is greater than the length of the longest word in $L(R)$ no more rules will be applied in σ_3. Thus, we need only consider the case where $L(R)$ is infinite. Because $L(R)$ is an infinite unary language, automata G in Figure 2 contains a single cycle that begins

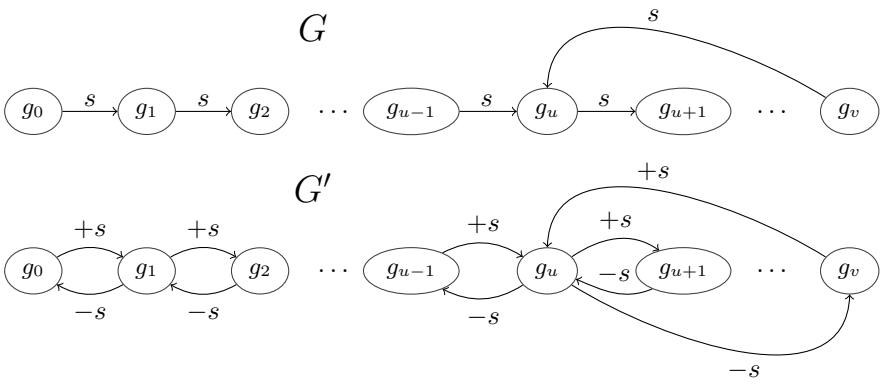

Figure 2: Finite state machine G decides if there is any rule applicable in a neuron given the number of spikes in the neuron *at a given time* in the computation. Each s represents a spike in the neuron. Machine G' keeps track of the movement of spikes into and out of the neuron and decides whether or not any rule is applicable *at each timestep* in the computation. $+s$ represents a single spike entering the neuron and $-s$ represents a single spike exiting the neuron.

at some state g_u. During a computation we may use G to decide which rules are applicable in σ_3 by passing an s to G each time a spike enters σ_3. However, G may not give the correct result if spikes leave the neuron as it does not record spikes leaving σ_3. Thus, using G we may construct a second machine G' such that G' records the movement of spikes going into and out of the neuron. G' is constructed as follows: G' has all the same states (including accept states) and transitions as G along with an extra set of transitions that record spikes leaving the neuron. This extra set of transitions are given as follows: for each transition on s from a state g_i to a state g_j in G there is a new transition on $-s$ going from state g_j to g_i in G' that records the removal of a spike from σ_3. By recording the dynamic movement of spikes, G' is able to decide which rules are applicable in σ_3 at each timestep during the computation. G' is also given in Figure 2. To simulate the operation of σ_3 we emulate the operation of G' in the states of T_Π. Note that there is a single non-deterministic choice to be made in G'. This choice is at state g_u if a spike is being removed ($-s$). It would seem that in order to make the correct choice in this situation we need to know the exact number of spikes in σ_3. However, we need only store at most $u + yq$ spikes. The reason for this is that if there are $\geqslant u + yq$ spikes in σ_3, then G' will not enter state g_{u-1} again. To see this, note that σ_3 spikes a maximum of y times using at most q spikes each time, and so once there are $> u + yq$ spikes the number of spikes in σ_3 will be $> u - 1$ for the remainder of the computation. Thus, T_Π simulates the activity of σ_3 by simulating the operation of G' and encoding at most $u + yq$ spikes in its states.

In this paragraph we explain the operation of T_Π. Following this, we give an analysis of the space complexity of T_Π. T_Π has 4 tapes including an output tape, which is initially blank, and a read only input tape. The tape head on both the input and output tapes is permitted to only move right. Each of the remaining tapes, tapes 1 and 2 simulate the activity of the neurons σ_1 and σ_2, respectively. These tapes record the number of spikes in σ_1 and σ_2. A timestep of Π is simulated

as follows: T_Π scans tapes 1 and 2 to determine if there are any applicable rules in σ_1 and σ_2 at that timestep. The applicability of each neural rule in Π is determined by a regular expression and so a decider for each rule is easily implemented in the states of T_Π. Recall from the previous paragraph that the applicability of the rules in σ_3 is already recorded in the states of T_Π. Also, T_Π is non-deterministic and so if more than one rule is applicable in a neuron T_Π simply chooses the rule to simulate in the same manner as Π. Once T_Π has determined which rules are applicable in each of the three neurons at that timestep it changes the encodings on tapes 1 and 2 to simulate the change in the number of spikes in neurons σ_1 and σ_2 during that timestep. As mentioned in the previous paragraph any change in the number of spikes in σ_3 is recorded in the states of T_Π. The input sequence of Π may be given as binary input to T_Π by placing it on its input tape. Also, if at a given timestep a 1 is read on the input tape then T_Π simulates a spike entering the simulated input neuron. At each simulated timestep, if the output neuron σ_3 spikes then a 1 is placed on the output tape, and if σ_3 does not spike a 0 is placed on the output tape. Thus the output of Π is encoded on the output tape when the simulation ends.

In a two neuron system each neuron has at most one out-going synapse and so the number of spikes in the system does not increase over time. Thus, the total number of spikes in neurons σ_1 and σ_2 can only increase when σ_3 fires or a spike is sent into the system from the environment. The input is of length n, and so σ_1 and σ_2 receive a maximum of n spikes from the environment. Neuron σ_3 fires no more than y times sending at most r spikes each time to σ_1 and σ_2. Thus the maximum number of spikes in σ_1 and σ_2 during the computation is $n + 2ry$. Using a binary encoding tapes 1 and 2 of T_Π encode the number of spikes in σ_1 and σ_2 using space of $\log_2(n + 2ry)$. As mentioned earlier no space is used to simulate σ_3, and thus T_Π simulates Π using space of $O(\log n)$. □

If we remove the restriction that allows the output neuron to fire only a constant number of times then we may construct a universal system with 3 neurons.

Theorem 3. *Let C be a universal counter machine with 3 counters that completes its computation in time t. Then there is a universal extended SN P system Π'_C with standard input and generalised output that simulates the computation of C in time $O(t)$ and has only 3 neurons.*

Proof. A graph of Π'_C is constructed by removing the output neuron σ_4 from the graph in Figure 1 and making σ_3 the new output neuron by adding a synapse to the environment. The rules for Π'_C are given by the first 3 rows of Table 2. The operation of Π'_C is identical to the operation of Π_C from Theorem 1 with the exception of the new output technique. The output of Π'_C is the time interval between the first and second timesteps where exactly $12h$ spikes are sent out of the output neuron σ_3. □

From the last paragraph of the proof of Theorem 2 we get Corollary 1.

Corollary 1. *Let Π be any extended SN P system with only 2 neurons and generalised input and output. Then there is a non-deterministic Turing machine T_Π that simulates the computation of Π in space $O(\log n)$ where n is the length of the input to Π.*

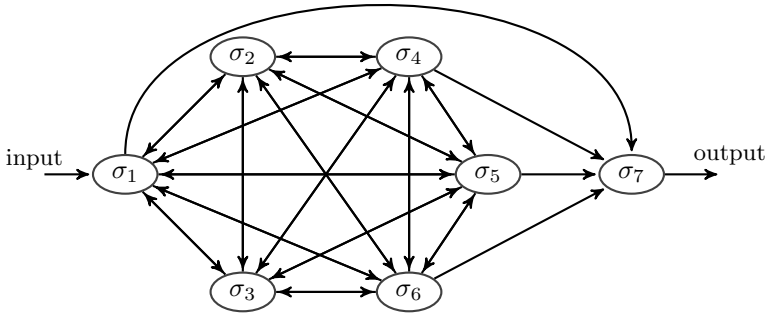

Figure 3: Universal SN P system Π_C. Each oval labeled σ_i is a neuron. An arrow going from neuron σ_i to neuron σ_j illustrates a synapse (i, j).

6. A small universal SN P system

Theorem 4. *Let C be a universal counter machine with 3 counters and h instructions that completes its computation in time t. Then there is a universal SN P system Π_C that simulates the computation of C in time $O(ht)$ and has only 7 neurons.*

Proof. Let $C = (3, \{c_1, c_2\}, c_3, Q, q_1, q_h)$ where $Q = \{q_1, q_2, \cdots, q_h\}$. Note that to operate on all three of its counters there must be at least 3 non-halting instructions and so we assume $h \geqslant 4$. The SN P system Π_C that simulates C is given by Figure 3 and Table 3. The operation of Π_C is deterministic. This follows from the fact that in each neuron σ_i of Π_C, for every possible pair of rules $E/s^b \to s$ and $E'/s^{b'} \to s$ in R_i, the sets $L(E)$ and $L(E')$ are disjoint. This property ensures that no more than one rule is applicable in each neuron at a given timestep, and so the operation of Π_C is deterministic. Given the values $h \geqslant 4$, $1 \leqslant i < h$, $1 \leqslant l \leqslant h$, $1 \leqslant k \leqslant h$, $4h - 1 \leqslant r \leqslant 8h - 4$ and $0 \leqslant m \leqslant 6h - 1$, it is a straightforward mater to verify from Table 3 that the disjoint property holds for each neuron in Π_C. Here the variables i, j and k come from the two types of counter machine instruction given immediately after Definition 2, and the range of values for r and m are chosen to maintain the disjoint property.

6.0.2. Encoding of a configuration of C and reading input into Π_C.

A configuration of C is stored as spikes in the neurons of Π_C. The next instruction q_i to be executed is stored in each of the neurons σ_3, σ_4, σ_5 and σ_6 as $8i$ spikes. Let x_1, x_2 and x_3 be the values stored in counters c_1, c_2 and c_3, respectively. Then the value x_1 is stored as $8h(x_1 + 3)$ spikes in neuron σ_4, x_2 is stored as $8h(x_2 + 3)$ spikes in σ_5, and x_3 is stored as $8h(x_3 + 3)$ spikes in σ_6.

The input to Π_C is read into the system via the input neuron σ_1 (see Figure 3). If C begins its computation with the input values x_1 and x_2 in counters c_1 and c_2, respectively, then the binary sequence $w = 10^{2hx_1}10^{2hx_2}1$ is read in via the input neuron σ_1. Thus, σ_1 receives a spike from the environment at times t_1, t_{2hx_1+2} and $t_{2hx_1+2hx_2+3}$. We explain how the system is initialised to encode an initial

configuration of C by giving the number of spikes in each neuron and the rule that is to be applied in each neuron at time t. Before the computation begins neuron σ_1 contains $16h+5$ spikes, neurons σ_2, σ_3 and σ_6 contain 10 spikes, σ_4 contains 17 spikes, and σ_5 contains 14 spikes. Thus, when σ_1 receives it first spike at time t_1 we have

$$\begin{array}{lll} t_1: & \sigma_1 : 16h+6, & s^{16h+6}/s^2 \to s, \\ & \sigma_2, \sigma_3, \sigma_6 : 10, & \\ & \sigma_4 : 17, & \\ & \sigma_5 : 14. & \end{array}$$

where on the left $\sigma_j : v$ gives the number v of spikes in neuron σ_j at time t_i and on the right is the next rule that is to be applied at time t_i if there is an applicable rule at that time. Thus from Figure 3, when we apply the rule $s^{16h+6}/s^2 \to s$ in neuron σ_1 at time t_1 we get

$$\begin{array}{lll} t_2: & \sigma_1 : 16h+4, & s^{16h+4}/s^5 \to s, \\ & \sigma_2, \sigma_3, \sigma_6 : 11, & s^{11}/s^3 \to s, \\ & \sigma_4 : 18, & (s^4)^* s^{18}/s \to s, \\ & \sigma_5 : 15, & s^{15}/s \to s, \\ & \sigma_7 : 1, & s \to \lambda, \end{array}$$

$$\begin{array}{lll} t_3: & \sigma_1 : 16h+4, & s^{16h+4}/s^5 \to s, \\ & \sigma_2, \sigma_3, \sigma_6 : 13, & s^{13}/s^5 \to s, \\ & \sigma_4 : 22, & (s^4)^* s^{18}/s \to s, \\ & \sigma_5 : 19, & s^{19}/s^5 \to s, \\ & \sigma_7 : 4, & s^4 \to \lambda, \end{array}$$

$$\begin{array}{lll} t_4: & \sigma_1 : 16h+4, & s^{16h+4}/s^5 \to s, \\ & \sigma_2, \sigma_3, \sigma_6 : 13, & s^{13}/s^5 \to s, \\ & \sigma_4 : 26, & (s^4)^* s^{18}/s \to s, \\ & \sigma_5 : 19, & s^{19}/s^5 \to s, \\ & \sigma_7 : 4, & s^4 \to \lambda. \end{array}$$

Neurons σ_1 to σ_6 fire on every timestep between times t_2 and t_{2hx_1+2} to accumulate a total of $8hx_1+18$ spikes in σ_4. Thus, when σ_1 receives the second spike from its environment at time t_{2hx_1+2} we have

$$\begin{array}{lll} t_{2hx_1+2}: & \sigma_1 : 16h+5, & \\ & \sigma_2, \sigma_3, \sigma_6 : 13, & s^{13}/s^5 \to s, \\ & \sigma_4 : 8hx_1+18, & (s^4)^* s^{18}/s \to s, \\ & \sigma_5 : 19, & s^{19}/s^5 \to s, \\ & \sigma_7 : 4, & s^4 \to \lambda, \end{array}$$

$$
\begin{aligned}
t_{2hx_1+3}: \quad & \sigma_1 : 16h+10, && s^{16h+10}/s^5 \to s,\\
& \sigma_2, \sigma_3, \sigma_6 : 12, && s^{12}/s \to s,\\
& \sigma_4 : 8hx_1+21, && (s^{8h})^* s^{2r+1}/s^5 \to s,\\
& \sigma_5 : 18, && (s^4)^* s^{18}/s \to s,\\
& \sigma_7 : 3, && s^3 \to \lambda,\\
t_{2hx_1+4}: \quad & \sigma_1 : 16h+10, && s^{16h+10}/s^5 \to s,\\
& \sigma_2, \sigma_3, \sigma_6 : 16, && s^{16}/s^5 \to s,\\
& \sigma_4 : 8hx_1+21, && (s^{8h})^* s^{2r+1}/s^5 \to s,\\
& \sigma_5 : 22, && (s^4)^* s^{18}/s \to s,\\
& \sigma_7 : 4, && s^4 \to \lambda,
\end{aligned}
$$

$$
\begin{aligned}
t_{2hx_1+5}: \quad & \sigma_1 : 16h+10, && s^{16h+10}/s^5 \to s,\\
& \sigma_2, \sigma_3, \sigma_6 : 16, && s^{16}/s^5 \to s,\\
& \sigma_4 : 8hx_1+21, && (s^{8h})^* s^{2r+1}/s^5 \to s,\\
& \sigma_5 : 26, && (s^4)^* s^{18}/s \to s,\\
& \sigma_7 : 4, && s^4 \to \lambda.
\end{aligned}
$$

Neurons σ_1, to σ_6 fire on every timestep between times t_{2hx_1+3} and $t_{2hx_1+2hx_2+3}$ to accumulate a total of $8hx_2+18$ spikes in σ_5. Thus, when σ_1 receives the last spike from its environment we have

$$
\begin{aligned}
t_{2hx_1+2hx_2+3}: \quad & \sigma_1 : 16h+11, && \\
& \sigma_2, \sigma_3, \sigma_6 : 16, && s^{16}/s^5 \to s,\\
& \sigma_4 : 8hx_1+21, && (s^{8h})^* s^{2r+1}/s^5 \to s,\\
& \sigma_5 : 8hx_2+18, && (s^4)^* s^{18}/s \to s,\\
& \sigma_7 : 4, && s^4 \to \lambda,
\end{aligned}
$$

$$
\begin{aligned}
t_{2hx_1+2hx_2+4}: \quad & \sigma_1 : 16h+16, && s^{16h+16}/s^{4h+18} \to s,\\
& \sigma_2, \sigma_3, \sigma_6 : 15, && \\
& \sigma_4 : 8hx_1+20, && (s^8)^* s^4/s^4 \to s,\\
& \sigma_5 : 8hx_2+21, && (s^{8h})^* s^{2r+1}/s^5 \to s,\\
& \sigma_7 : 3, && s^3 \to \lambda,
\end{aligned}
$$

$$
\begin{aligned}
t_{2hx_1+2hx_2+5}: \quad & \sigma_1 : 12h, && s^{2m+8}/s^7 \to s,\\
& \sigma_2 : 18, && s^{18} \to s,\\
& \sigma_3, \sigma_6 : 18, && (s^4)^* s^{18}/s \to s,\\
& \sigma_4 : 8hx_1+18, && (s^4)^* s^{18}/s \to s,\\
& \sigma_5 : 8hx_2+18, && (s^4)^* s^{18}/s \to s,\\
& \sigma_7 : 3, && s^3 \to \lambda,
\end{aligned}
$$

$$
\begin{array}{lll}
t_{2hx_1+2hx_2+6}: & \sigma_1 : 12h-2, & s^{2m+8}/s^7 \to s, \\
& \sigma_2 : 5, & s^5 \to s, \\
& \sigma_3, \sigma_6 : 22, & (s^4)^* s^{18}/s \to s, \\
& \sigma_4 : 8hx_1+22, & (s^4)^* s^{18}/s \to s, \\
& \sigma_5 : 8hx_2+22, & (s^4)^* s^{18}/s \to s, \\
& \sigma_7 : 4, & s^4 \to \lambda.
\end{array}
$$

We continue to execute the rules used at time $t_{2hx_1+2hx_2+6}$ until time $t_{2hx_1+2hx_2+6h+2}$ when we get

$$
\begin{array}{lll}
t_{2hx_1+2hx_2+6h+2}: & \sigma_1 : 6, & \\
& \sigma_2 : 5, & s^5 \to s, \\
& \sigma_3, \sigma_6 : 24h+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_4 : 8h(x_1+3)+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_5 : 8h(x_2+3)+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_7 : 4, & s^4 \to \lambda, \\
& & \\
t_{2hx_1+2hx_2+6h+3}: & \sigma_1 : 11, & s^{11}/s^2 \to s \\
& \sigma_2 : 4, & \\
& \sigma_3 : 24h+9, & (s^{8h})^* s^{2r+3}/s^5 \to s, \\
& \sigma_4 : 8h(x_1+3)+9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_5 : 8h(x_2+3)+9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_6 : 24h+9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_7 : 3, & s^3 \to \lambda, \\
& & \\
t_{2hx_1+2hx_2+6h+4}: & \sigma_1 : 13, & \\
& \sigma_2 : 9, & \\
& \sigma_3, \sigma_6 : 24h+8, & \\
& \sigma_4 : 8h(x_1+3)+8, & \\
& \sigma_5 : 8h(x_2+3)+8, & \\
& \sigma_7 : 4. &
\end{array}
$$

At time $t_{2hx_1+2hx_2+6h+4}$ the neurons of Π_C encode an initial configuration of C. To see this note that neuron σ_4 encodes the initial value x_1 of counter c_1 with $8h(x_1+3)$ spikes and the start instruction q_1 with 8 spikes, neuron σ_5 encodes the initial value x_2 of counter c_2 with $8h(x_2+3)$ spikes and instruction q_1 with 8 spikes, and σ_6 encodes the initial value 0 of counter c_3 with $24h$ spikes and instruction q_1 with 8 spikes.

neuron	rules
σ_1	$s^{16h+6}/s^2 \to s, \quad s^{16h+4}/s^5 \to s, \quad s^{16h+10}/s^5 \to s,$ $s^{16h+16}/s^{4h+18} \to s, \quad s^{2m+8}/s^7 \to s, \quad s^{11}/s^2 \to s$ $s^{12h+4i+6}/s^{4h+4i-4l+7} \to s, \quad s^{12h+4i+7}/s^{4h+4i-4k+5} \to s$ $s^{15} \to \lambda, \quad s^3 \to \lambda, \quad s^2 \to \lambda,$
σ_2	$s^{11}/s^3 \to s, \quad s^{13}/s^5 \to s, \quad s^{12}/s \to s, \quad s^{16}/s^5 \to s,$ $s^{18} \to s, \quad s^5 \to s, \quad s^9 \to \lambda, \quad s^8 \to \lambda, \quad s \to s$ $s^6 \to \lambda, \quad s^2 \to \lambda, \quad s^3 \to \lambda,$
σ_3	$s^{11}/s^3 \to s, \quad s^{13}/s^5 \to s, \quad s^{12}/s \to s, \quad s^{16}/s^5 \to s,$ $(s^4)^* s^{18}/s \to s, \quad (s^{8h})^* s^{2r+3}/s^5 \to s, \quad s^{24h+8i}/s^{10h+6i+10} \to s,$ $s^{24h+8i+4}/s^{10h+6i+14} \to s, \quad s^{8h-1}/s \to s, \quad s^{32h} \to \lambda,$ $s^{32h+4} \to \lambda, \quad s^2 \to \lambda, \quad s^3 \to \lambda,$
σ_4	$(s^4)^* s^{18}/s \to s, \quad (s^{8h})^* s^{2r+1}/s^5 \to s, \quad s^2 \to s, \quad s \to \lambda,$ $(s^8)^* s^4/s^4 \to s, \quad s^{8h-5}/s \to s, \quad s^{32h} \to \lambda$ $(s^{8h})^* s^{8i}/s^{2h+6i+12} \to s \quad \text{if } [q_i : INC(1), q_l] \in \{Q\}$ $(s^{8h})^* s^{32h+8i}/s^{18h+6i+12} \to s \quad \text{if } [q_i : DEC(1), q_l, q_k] \in \{Q\}$ $s^{24h+8i}/s^{10h+6i+10} \to s \quad \text{if } [q_i : DEC(1), q_l, q_k] \in \{Q\}$ $(s^{8h})^* s^{8i}/s^{10h+6i+12} \to s \quad \text{if } [q_i : INC(1), q_l],\ [q_i : DEC(1), q_l, q_k] \notin \{Q\}$
σ_5	$s^{15}/s \to s, \quad s^{19}/s^5 \to s, \quad (s^4)^* s^{18}/s \to s, \quad s^{8h-5}/s \to s,$ $(s^{8h})^* s^{2r+1}/s^5 \to s, \quad s^{32h}/s^{32h-1} \to s, \quad s^2 \to s, \quad s \to \lambda,$ $(s^{8h})^* s^{8i}/s^{2h+6i+12} \to s \quad \text{if } [q_i : INC(2), q_l] \in \{Q\}$ $(s^{8h})^* s^{32h+8i}/s^{18h+6i+12} \to s \quad \text{if } [q_i : DEC(2), q_l, q_k] \in \{Q\}$ $s^{24h+8i}/s^{10h+6i+10} \to s \quad \text{if } q_i : DEC(2), q_l, q_k] \in \{Q\}$ $(s^{8h})^* s^{8i}/s^{10h+6i+12} \to s \quad \text{if } [q_i : INC(2), q_l],\ [q_i : DEC(2), q_l, q_k] \notin \{Q\}$
σ_6	$s^{11}/s^3 \to s, \quad s^{13}/s^5 \to s, \quad s^{12}/s \to s, \quad s^{16}/s^5 \to s,$ $(s^4)^* s^{18}/s \to s, \quad (s^{8h})^* s^{2r+1}/s^5 \to s, \quad s^{8h-4} \to \lambda, \quad s^2 \to \lambda,$ $s^{8h-5}/s \to s, \quad (s^{8h})^*/s^{24h+5} \to s, \quad (s^{8h})^* s^{16h-4}/s^{8h-2} \to s$ $(s^{8h})^* s^{8i}/s^{2h+6i+12} \to s \quad \text{if } [q_i : INC(3), q_l] \in \{Q\}$ $(s^{8h})^* s^{32h+8i}/s^{18h+6i+12} \to s \quad \text{if } [q_i : DEC(3), q_l, q_k] \in \{Q\}$ $s^{24h+8i}/s^{10h+6i+10} \to s \quad \text{if } [q_i : DEC(3), q_l, q_k] \in \{Q\}$ $(s^{8h})^* s^{8i}/s^{10h+6i+12} \to s \quad \text{if } [q_i : INC(3), q_l],\ [q_i : DEC(3), q_l, q_k] \notin \{Q\}$
σ_7	$s \to \lambda, \quad s^4 \to \lambda, \quad s^3 \to \lambda, \quad (s^3)^* s^5/s^2 \to s,$

Table 3: Rules for neurons σ_1 to σ_6 of Π_C, where $(1 \leqslant i < h)$, $(1 \leqslant l, k \leqslant h)$, $(4h-1 \leqslant r \leqslant 8h-4)$ and $(0 \leqslant m \leqslant 6h-1)$.

6.0.3. Π_C *simulating* $q_i : INC(1), q_l$.

Let x_1, x_2 and x_3 be the values in counters c_1, c_2 and c_3, respectively. Then our simulation of q_i begins with $24h + 8i$ spikes in σ_3, $8h(x_1 + 3) + 8i$ spikes in σ_4, $8h(x_2 + 3) + 8i$ spikes in σ_5, and $8h(x_3 + 3) + 8i$ spikes in σ_6. Beginning our simulation of $INC(1)$ at time t_j, we have

$$
\begin{array}{lll}
t_j: & \sigma_1 : 13, & \\
& \sigma_2 : 9, & s^9 \to \lambda, \\
& \sigma_3 : 24h + 8i, & s^{24h+8i}/s^{10h+6i+10} \to s, \\
& \sigma_4 : 8h(x_1 + 3) + 8i, & (s^{8h})^* s^{8i}/s^{2h+6i+12} \to s, \\
& \sigma_5 : 8h(x_2 + 3) + 8i, & (s^{8h})^* s^{8i}/s^{10h+6i+12} \to s, \\
& \sigma_6 : 8h(x_3 + 3) + 8i, & (s^{8h})^* s^{8i}/s^{10h+6i+12} \to s, \\
& \sigma_7 : 4, & s^4 \to \lambda.
\end{array}
$$

Thus, from Figure 3 we get

$$
\begin{array}{lll}
t_{j+1}: & \sigma_1 : 17, & \\
& \sigma_2 : 4, & \\
& \sigma_3 : 14h + 2i - 7 & (s^{8h})^* s^{2r+3}/s^5 \to s, \\
& \sigma_4 : 8h(x_1 + 2) + 6h + 2i - 9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_5 : 8h(x_2 + 1) + 6h + 2i - 9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_6 : 8h(x_3 + 1) + 6h + 2i - 9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_7 : 3, & s^3 \to \lambda,
\end{array}
$$

$$
\begin{array}{lll}
t_{j+2}: & \sigma_1 : 21, & \\
& \sigma_2 : 8, & s^8 \to \lambda \\
& \sigma_3 : 14h + 2i - 9, & (s^{8h})^* s^{2r+3}/s^5 \to s, \\
& \sigma_4 : 8h(x_1 + 2) + 6h + 2i - 11, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_5 : 8h(x_2 + 1) + 6h + 2i - 11, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_6 : 8h(x_3 + 1) + 6h + 2i - 11, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_7 : 3, & s^3 \to \lambda.
\end{array}
$$

Neurons σ_3, σ_4, σ_5 and σ_6 fire on every timestep between times t_{j+1} and $t_{j+3h+i-2}$ to accumulate a total of $12h+4i+5$ spikes to σ_1 and so assuming $3h+i-2 \equiv 1 \bmod 2$ we get the configurations shown below. If $3h + i - 2 \equiv 0 \bmod 2$ we have a minor difference in the configurations below: There are 8 spikes (instead of 4) in σ_2 at timestep $t_{j+3h+i-2}$, and so the rule $s^8 \to \lambda$ is applied which means that there is exactly 1 spike in σ_2 at timestep $t_{j+3h+i-1}$. In this case the rule $s \to s$ is applied in σ_2 at timestep $t_{j+3h+i-1}$ which gives a configuration of the form shown at timestep t_{j+3h+i} below.

$$
\begin{array}{lll}
t_{j+3h+i-2}: & \sigma_1 : 12h+4i+5, & \\
& \sigma_2 : 4, & \\
& \sigma_3 : 8h-1, & s^{8h-1}/s \to s \\
& \sigma_4 : 8h(x_1+2)-3, & \\
& \sigma_5 : 8h(x_2+1)-3, & \\
& \sigma_6 : 8h(x_3+1)-3, & \\
& \sigma_7 : 3, & s^3 \to \lambda,
\end{array}
$$

$$
\begin{array}{lll}
t_{j+3h+i-1}: & \sigma_1 : 12h+4i+6, & s^{12h+4i+6}/s^{4h+4i-4l+7} \to s, \\
& \sigma_2 : 5, & s^5 \to s, \\
& \sigma_3 : 8h-2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_4 : 8h(x_1+2)-2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_5 : 8h(x_2+1)-2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_6 : 8h(x_3+1)-2, & (s^4)^* s^{18}/s \to s,
\end{array}
$$

$$
\begin{array}{lll}
t_{j+3h+i}: & \sigma_1 : 8h+4l+4, & s^{2m+8}/s^7 \to s, \\
& \sigma_2 : 5, & s^5 \to s, \\
& \sigma_3 : 8h+2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_4 : 8h(x_1+2)+2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_5 : 8h(x_2+1)+2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_6 : 8h(x_3+1)+2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_7 : 4, & s^4 \to \lambda.
\end{array}
$$

At time $t_{j+3h+i-1}$ the rule $s^{12h+4i+6}/s^{4h+4i-4l+7} \to s$ is executed in σ_1 to begin the process of moving from instruction q_i to the next instruction q_l and to place the correct values in neurons σ_4, σ_5 and σ_6 to simulate the counter values x_1+1, x_2 and x_3. After the rule $s^{12h+4i+6}/s^{4(h+i-l)+7} \to s$ is executed the value $8h+4l+4$ in σ_1 records how many timesteps to reapply the rules used at timestep t_{j+3h+i}. Specifically, the value in σ_1 is decremented by 2 at each timestep until only 6 spikes remain in σ_1 and during each of these timestep the number of spikes in σ_3, σ_4, σ_5 and σ_6 increases by 4. This gives

$$
\begin{array}{lll}
t_{j+7h+i+2l-1}: & \sigma_1 : 6, & \\
& \sigma_2 : 5, & s^5 \to s, \\
& \sigma_3 : 24h+8(l-1)+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_4 : 8h(x_1+4)+8(l-1)+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_5 : 8h(x_2+3)+8(l-1)+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_6 : 8h(x_3+3)+8(l-1)+6, & (s^4)^* s^{18}/s \to s, \\
& \sigma_7 : 4, & s^4 \to \lambda,
\end{array}
$$

$$
\begin{array}{lll}
t_{j+7h+i+2l}: & \sigma_1: 11, & s^{11}/s^2 \to s \\
& \sigma_2: 4, & \\
& \sigma_3: 24h+8l+1, & (s^{8h})^* s^{2r+3}/s^5 \to s, \\
& \sigma_4: 8h(x_1+4)+8l+1, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_5: 8h(x_2+3)+8l+1, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_6: 8h(x_2+3)+8l+1, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_7: 3, & s^3 \to \lambda,
\end{array}
$$

$$
\begin{array}{ll}
t_{j+7h+i+2l+1}: & \sigma_1: 13, \\
& \sigma_2: 9, \\
& \sigma_3: 24h+8l, \\
& \sigma_4: 8h(x_1+4)+8l, \\
& \sigma_5: 8h(x_2+3)+8l, \\
& \sigma_6: 8h(x_2+3)+8l, \\
& \sigma_7: 4.
\end{array}
$$

In the above configurations $4h-1 \leqslant r \leqslant 8h-4$. At time $t_{j+7h+i+2l+1}$ the simulation of $q_i : INC(1), q_l$ is complete. Note that an increment on the value x_1 in counter c_1 was simulated by increasing the number of spikes in σ_4 from $8h(x_1+3)$ to $8h(x_1+4)$. Note also that the encoding $8l$ of the next instruction q_l has been established in neurons σ_3, σ_4, σ_5 and σ_6.

6.0.4. Π_C *simulating* $q_i : DEC(1), q_l, q_k$ *when* $x_1 > 0$.

If we are simulating $DEC(1)$ for $x_1 > 0$ then we have

$$
\begin{array}{lll}
t_j: & \sigma_1: 13, & \\
& \sigma_2: 9, & s^9 \to \lambda, \\
& \sigma_3: 24h+8i, & s^{24h+8i}/s^{10h+6i+10} \to s, \\
& \sigma_4: 8h(x_1+3)+8i, & (s^{8h})^* s^{32h+8i}/s^{18h+6i+12} \to s, \\
& \sigma_5: 8h(x_2+3)+8i, & (s^{8h})^* s^{8i}/s^{10h+6i+12} \to s, \\
& \sigma_6: 8h(x_3+3)+8i, & (s^{8h})^* s^{8i}/s^{10h+6i+12} \to s, \\
& \sigma_7: 4, & s^4 \to \lambda.
\end{array}
$$

The only difference between the simulation of $INC(1), q_l$ and $DEC(1), q_l, q_k$ (for $x_1 > 0$) is that an extra $16h$ spikes are used up in neurons σ_4 at time t_j. To see this note the difference in the number of spikes used by the rules executed in σ_4 at time t_j in this example and the example for $INC(1)$. The remainder of the simulation of $DEC(1), q_l, q_k$ for $x_1 > 0$ between times t_{j+1} and $t_{j+7h+i+2l+1}$ proceeds in the same manner as in the previous example $INC(1), q_l$. Thus at time $t_{j+7h+i+2l+1}$

we have

$$\begin{aligned} t_{j+7h+i+2l+1}: \quad & \sigma_1 : 13, \\ & \sigma_2 : 9, \\ & \sigma_3 : 24h + 8l, \\ & \sigma_4 : 8h(x_1 + 2) + 8l, \\ & \sigma_5 : 8h(x_2 + 3) + 8l, \\ & \sigma_6 : 8h(x_2 + 3) + 8l, \\ & \sigma_7 : 4. \end{aligned}$$

At time $t_{j+7h+i+2l+1}$ the simulation of $q_i : DEC(1), q_l, q_k$ for $x_1 > 0$ is complete. Note that a decrement on the value x_1 in counter c_1 was simulated by decreasing the number of spikes in σ_4 from $8h(x_1 + 3)$ to $8h(x_1 + 2)$ spikes. Note also that the encoding $8l$ of the next instruction q_l has been established in neurons σ_4, σ_5 and σ_6.

6.0.5. Π_C *simulating* $q_i : DEC(1), q_l, q_k$ *when* $x_1 = 0$.

If we are simulating $DEC(1)$ for $x_1 = 0$ then we have

$$\begin{aligned} t_j: \quad & \sigma_1 : 13, & \\ & \sigma_2 : 9, & s^9 \to \lambda, \\ & \sigma_3, \sigma_4 : 24h + 8i, & s^{24h+8i}/s^{10h+6i+10} \to s, \\ & \sigma_5 : 8h(x_2 + 3) + 8i, & (s^{8h})^* s^{8i}/s^{10h+6i+12} \to s, \\ & \sigma_6 : 8h(x_3 + 3) + 8i, & (s^{8h})^* s^{8i}/s^{10h+6i+12} \to s, \\ & \sigma_7 : 4, & s^4 \to \lambda \end{aligned}$$

$$\begin{aligned} t_{j+1}: \quad & \sigma_1 : 17, & \\ & \sigma_2 : 4, & \\ & \sigma_3 : 14h + 2i - 7, & (s^{8h})^* s^{2r+3}/s^5 \to s, \\ & \sigma_4 : 14h + 2i - 7, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\ & \sigma_5 : 8h(x_2 + 1) + 6h + 2i - 9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\ & \sigma_6 : 8h(x_3 + 1) + 6h + 2i - 9, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\ & \sigma_7 : 3, & s^3 \to \lambda. \end{aligned}$$

In a manner similar to the $INC(1)$ example the rules executed at timestep t_{j+1} continue to be executed until timestep $t_{j+3h+i-2}$, and so assuming $3h + i - 2 \equiv 1 \bmod 2$ we get the configurations shown below. If $3h + i - 2 \equiv 0 \bmod 2$ we have a minor difference in the configurations below: There are 8 spikes (instead of 4) in σ_2 at timestep $t_{j+3h+i-2}$, and so the rule $s^8 \to \lambda$ is applied which means that there are exactly 2 spikes in σ_2 at timestep $t_{j+3h+i-1}$. In this case the rule $s^2 \to \lambda$ is applied in σ_2 at timestep $t_{j+3h+i-1}$ which gives a configuration of the form shown at timestep t_{j+3h+i} below.

$$
\begin{array}{lll}
t_{j+3h+i-2}: & \sigma_1 : 12h+4i+5, & \\
& \sigma_2 : 4, & \\
& \sigma_3 : 8h-1, & s^{8h-1}/s \to s \\
& \sigma_4 : 8h-1, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_5 : 8h(x_2+1)-3, & \\
& \sigma_6 : 8h(x_3+1)-3, & \\
& \sigma_7 : 3, & s^3 \to \lambda,
\end{array}
$$

$$
\begin{array}{lll}
t_{j+3h+i-1}: & \sigma_1 : 12h+4i+7, & s^{12h+4i+7}/s^{4h+4i-4k+5} \to s, \\
& \sigma_2 : 6, & s^6 \to \lambda, \\
& \sigma_3 : 8h-1, & s^{8h-1}/s \to s \\
& \sigma_4 : 8h-5, & s^{8h-5}/s \to s, \\
& \sigma_5 : 8h(x_2+1)-1, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_6 : 8h(x_3+1)-1, & (s^{8h})^* s^{2r+1}/s^5 \to s, \\
& \sigma_7 : 1, & s \to \lambda,
\end{array}
$$

$$
\begin{array}{lll}
t_{j+3h+i}: & \sigma_1 : 8h+4k+6, & s^{2m+8}/s^7 \to s, \\
& \sigma_2 : 5, & s^5 \to s, \\
& \sigma_3 : 8h+2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_4 : 8h-2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_5 : 8h(x_2+1)-2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_6 : 8h(x_3+1)-2, & (s^4)^* s^{18}/s \to s, \\
& \sigma_7 : 4, & s^4 \to \lambda.
\end{array}
$$

In the previous two examples only σ_3 fired at time $t_{j+3h+i-2}$, whereas in this example both σ_3 and σ_4 fire at time $t_{j+3h+i-2}$ sending an extra spike to neuron σ_1. This only occurs during the simulation of a $DEC(1)$ instruction when $x_1 = 0$, and this allows neuron σ_1 to determine when the counter has value 0 as it contains $12h + 4i + 7$ spikes (instead of $12h + 4i + 6$ spikes) at time $t_{j+3h+i-1}$. Following time t_{j+3h+i}, the remainder of the simulation of $DEC(1)$ for $x_1 = 0$ proceeds in the same manner as the $INC(1)$ example to give

$$
\begin{array}{ll}
t_{j+7h+i+2k+2}: & \sigma_1 : 13, \\
& \sigma_2 : 9, \\
& \sigma_3 : 24h+8k+4, \\
& \sigma_4 : 24h+8k, \\
& \sigma_5 : 8h(x_2+3)+8k, \\
& \sigma_6 : 8h(x_3+3)+8k, \\
& \sigma_7 : 4.
\end{array}
$$

At time $t_{j+7h+i+2k+2}$ the simulation of $q_i : DEC(1), q_l, q_k$ for $x_1 = 0$ is complete. The encoding $8k$ of the next instruction q_k has been established in neurons σ_4, σ_5 and σ_6. Note that the number of spikes in neuron σ_3 is $24h + 8k + 4$ instead of the usual $24h + 8k$. This does not cause a problem during the simulation of the next counter machine instruction as a rule of the from $s^{24h+8i+4}/s^{10h+6i+14} \to s$ (instead of $s^{24h+8i}/s^{10h+6i+10} \to s$) is applied which uses up these extra 4 spikes at the beginning of the next simulated instruction.

6.0.6. Halting.

If C enters the halt instruction q_h at time t_j then we get the following

$$
\begin{array}{rll}
t_j : & \sigma_1 : 13, & \\
& \sigma_2 : 9, & s^9 \to \lambda, \\
& \sigma_3, \sigma_4 : 32h, & s^{32h} \to \lambda, \\
& \sigma_5 : 32h, & s^{32h}/s^{32h-1} \to s, \\
& \sigma_6 : 8h(x_3 + 4), & (s^{8h})^*/s^{24h+5} \to s \\
& \sigma_7 : 4, & s^4 \to \lambda \\
\\
t_{j+1} : & \sigma_1 : 15, & s^{15} \to \lambda, \\
& \sigma_2, \sigma_3 : 2, & s^2 \to \lambda, \\
& \sigma_4, \sigma_5 : 2, & s^2 \to s \\
& \sigma_6 : 8h(x_3 + 1) - 4, & (s^{8h})^* s^{16h-4}/s^{8h-2} \to s \\
& \sigma_7 : 2, & \\
\\
t_{j+2} : & \sigma_1, \sigma_2, \sigma_3 : 3, & s^3 \to \lambda, \\
& \sigma_4, \sigma_5 : 2, & s^2 \to s \\
& \sigma_6 : 8hx_3 - 4, & (s^{8h})^* s^{16h-4}/s^{8h-2} \to s \\
& \sigma_7 : 5, & (s^3)^* s^5/s^2 \to s \\
\\
t_{j+3} : & \sigma_1, \sigma_2, \sigma_3 : 3, & s^3 \to \lambda, \\
& \sigma_4, \sigma_5 : 2, & s^2 \to s \\
& \sigma_6 : 8h(x_3 - 1) - 4, & (s^{8h})^* s^{16h-4}/s^{8h-2} \to s \\
& \sigma_7 : 6, &
\end{array}
$$

With each timestep the simulated output counter is decremented by decreasing the number of spikes in neuron σ_4 by $8h$ until we get

$$
\begin{array}{rll}
t_{j+x_3+1} : & \sigma_1, \sigma_2, \sigma_3 : 3, & s^3 \to \lambda, \\
& \sigma_4, \sigma_5 : 2, & s^2 \to s \\
& \sigma_6 : 8h - 4, & s^{8h-4} \to \lambda \\
& \sigma_7 : 3x_3, &
\end{array}
$$

$$
\begin{array}{lll}
t_{j+x_3+2}: & \sigma_1, \sigma_2, \sigma_3, \sigma_6 : 2, & s^2 \to \lambda \\
& \sigma_4, \sigma_5 : 1, & s \to \lambda, \\
& \sigma_7 : 3x_3 + 2, & (s^3)^* s^5 / s^2 \to s.
\end{array}
$$

Recall from Section 2 that the output of an SN P system is the time interval between the first and second spikes that are sent out of the output neuron. Note from above that the output neuron σ_7 fires for the first time at timestep t_{j+2} and for the second time at timestep t_{j+x_3+2}. Thus, the output of Π_C is x_3 the value of the output counter c_3 when C enters the halt instruction q_h. It is also worth noting that no rule is applicable in any neuron after time t_{j+x_3+2} so the entire system halts. Note that in the case of $x_3 = 0$ the rule $s^{8h-4} \to \lambda$ is applied in σ_6 at time t_{j+1} and so the system halts immediately after timestep t_{j+2} without the output neuron firing.

We have shown how to simulate arbitrary instructions of the form $q_i : INC(1), q_l$ and $q_i : DEC(1), q_l, q_k$. Instructions that operate on counters c_2 and c_3 are simulated in a similar manner. Immediately following the simulation of an instruction Π_C is configured to begin simulation of the next instruction. Each instruction of C is simulated in $7h + i + 2l + 1$ timesteps. The pair of input values (x_1, x_2) is read into the system in $2hx_1 + 2hx_2 + 3h + 4$ timesteps and sending the output value x_3 out of the system takes $x_3 + 2$ timesteps. Thus, if C completes its computation in time t then Π_C simulates the computation of C in linear time $O(ht)$. □

7. Conclusion

Our 4-neuron system given in Theorem 1 has the smallest possible number of neurons needed to give a universal SN P system with extended rules. This result shows that there is no significant trade-off between the time/space complexity and the number of neurons in universal extended SN P systems (see Table 1). From Theorem 2 in [10, 11], we know that when simulating Turing machines our 4-neuron system suffers no exponential slow-down when compared with universal SN P systems with a greater number of neurons. In addition, this system does not generalise on the input and output encodings of earlier small SN P systems [2, 12].

As a further application of our 4-neuron system, we also showed at the end of Section 4 how it may be adapted to simulate non-deterministic counter machines simply by adding one extra rule for each increment instruction. It is not such a straightforward matter to adapt our 7 neuron system from Theorem 4 to simulate non-deterministic counter machines. The reason for this is that the next instruction to be simulated is determined by 2 neurons instead of 1. To ensure the correct simulation of non-deterministic instructions one would need to ensure that both neurons chose the same next instruction.

In Theorem 2 it is shown that there exists no universal extended SN P system with 3 neurons. This lower bound is also applicable to standard SN P systems, and so our 7-neuron system shows that the smallest number of neurons needed to give a universal standard SN P system is between 4 and 7 neurons. It seems likely that this lower bound of 4 neurons could be increased due to the fact that the rules used in standard systems are quite limited when compared with those used in extended SN P systems. Standard rules can send no more than one spike along a synapse

in a single timestep and this means the number of possible incoming signals to a neuron is quite limited in standard systems with a small number of neurons.

A related problem that is of interest is that of a possible trade-off between the time efficiency and the size of small standard SN P systems. To date all of the small standard and extended SN P systems were proved universal by simulating counter machines. The number of neurons in these SN P systems is partially dependent on the number of counters in the counter machine that is simulated to prove universality. As a result, one might expect a trade-off between the time efficiency and the number of neurons as (currently) 2-counter machines are exponentially slower than 3-counter machines when simulating Turing machines. However, as we mentioned above there is no significant trade-off between the number of neurons and the time efficiency for the extended model. It remains to be seen if this is the case for the standard model.

In Section 3 we discussed the input encodings used by the various small universal SN P systems given in Table 1. We mentioned that it would be of interest to find out if there is any trade-off between the size of universal SN P systems and the complexity of the input encoding allowed. In other computational models such as cellular automata and Turing machines more general initial conditions (or input) have allowed even simpler universal models to be constructed [25, 26, 27, 28, 29, 30]. Note that with these other models it is assumed that they are already initialised at the beginning of the computation. This is not the case for our small universal SN P systems and those given in Table 1. SN P systems have the added task of reading in the input at the beginning of the computation. The small universal SN P systems of other authors have a number of neurons that act as a dedicated input module. The systems we have given in this paper do not have a dedicated input module; the neurons that deal with the input also have other functions. Getting the neurons to perform more than one function adds to the complexity of our inputing technique. Note that a significant portion of the proofs of Theorems 1 and 4 are devoted to the reading in of input. For this reason, placing more restrictions on the input encoding could increase the difficulty of finding small universal SN P systems for the standard model.

References

[1] M. Ionescu, G. Păun, T. Yokomori, Spiking neural P systems, Fundamenta Informaticae 71 (2-3) (2006) 279–308.

[2] A. Păun, G. Păun, Small universal spiking neural P systems, BioSystems 90 (1) (2007) 48–60.

[3] O. H. Ibarra, A. Păun, G. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth, Normal forms for spiking neural P systems, Theoretical Computer Science 372 (2-3) (2010) 196–217.

[4] L. Pan, G. Păun, Spiking neural P systems: An improved normal form, Theoretical Computer Science 411 (6) (2010) 906–918.

[5] R. Freund, M. Kogler, Computationally complete spiking neural P systems without delay: Two types of neurons are enough, in: M. Gheorghe, T. Hinze,

G. Paun (Eds.), Proceedings of the Eleventh International Conference on Membrane Computing, Jena, Germany, 2010, pp. 193–204.

[6] G. Păun, M. J. Pérez-Jiménez, Algorithmic Bioprocesses, Natural Computing Series, Springer, 2009, Ch. Spiking Neural P Systems. Recent Results, Research Topics, pp. 273–291.

[7] T. Neary, A boundary between universality and non-universality in extended spiking neural P systems, in: In Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Vol. 6031 of LNCS, Springer, Trier, 2010, pp. 475–487.

[8] X. Zhang, Y. Jiang, L. Pan, Small universal spiking neural P systems with exhaustive use of rules, in: 3rd International Conference on Bio-Inspired Computing: Theories and Applications(BICTA 2008), IEEE, Adelaide, Australia, 2008, pp. 117–128.

[9] X. Zhang, Y. Jiang, L. Pan, Small universal spiking neural P systems with exhaustive use of rules, Journal of Computational and Theoretical Nanoscience 7 (5) (2010) 890–899.

[10] T. Neary, On the computational complexity of spiking neural P systems, in: C. S. Calude, J. F. Costa, R. Freund, M. Oswald, G. Rozenberg (Eds.), UC 2008, Vol. 5204 of LNCS, Springer, 2008, pp. 189–205.

[11] T. Neary, On the computational complexity of spiking neural P systems, Natural Computing 9 (4) (2010) 831–851.

[12] X. Zhang, X. Zeng, L. Pan, Smaller universal spiking neural P systems, Fundamenta Informaticae 87 (1) (2008) 117–136.

[13] T. Neary, A small universal spiking neural P system, in: E. Csuhaj-Varjú, R. Freund, M. Oswald, K. Salomma (Eds.), International Workshop on Computing with Biomolecules, Austrian Computer Society, Vienna, 2008, pp. 65–74.

[14] T. Neary, presentation at Computing with Biomolecules (CBM 2008). Available at http://www.emcc.at/UC2008/Presentations/CBM5.pdf (2008).

[15] X. Zeng, X. Zhang, L. Pan, A weakly universal spiking neural P system, Mathematical and Computer Modelling 52 (11-12) (2010) 1940–1946.

[16] T. Neary, A boundary between universality and non-universality in spiking neural P systems, Tech. Rep. arXiv:0912.0741v3 [cs.CC] (Dec. 2009).

[17] T. Neary, A universal spiking neural P system with 11 neurons, in: M. Gheorghe, T. Hinze, G. Paun (Eds.), Proceedings of the Eleventh International Conference on Membrane Computing, Jena, Germany, 2010, pp. 327–346.

[18] I. Korec, Small universal register machines, Theoretical Computer Science 168 (2) (1996) 267–301.

[19] G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, Spike trains in spiking neural P systems, International Journal of Foundations of Computer Science 17 (4) (2006) 975–1002.

[20] X. Zeng, X. Zhang, L. Pan, Homogeneous spiking neural p systems, Fundamenta Informaticae 97 (1-2) (2009) 275–294.

[21] M. Davis, The definition of universal Turing machine, Proceedings of the American Mathematical Society 8 (6) (1957) 1125–1126.

[22] Y. Rogozhin, Small universal Turing machines, Theoretical Computer Science 168 (2) (1996) 215–240.

[23] R. Schroeppel, A two counter machine cannot calculate 2^n, Tech. Rep. AIM-257, A.I. memo 257, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA (1972).

[24] P. C. Fischer, A. R. Meyer, A. L. Rosenberg, Counter machines and counter languages, Mathematical Systems Theory 2 (3) (1968) 265–283.

[25] M. Cook, Universality in elementary cellular automata, Complex Systems 15 (1) (2004) 1–40.

[26] K. Lindgren, M. G. Nordahl, Universal computation in simple one-dimensional cellular automata, Complex Systems 4 (3) (1990) 299–318.

[27] T. Neary, D. Woods, Small weakly universal Turing machines, in: Fundamentals of Computation Theory 17th International Symposium, FCT 2009, Vol. 5699 of LNCS, 2009, pp. 262–273.

[28] S. Watanabe, 5-symbol 8-state and 5-symbol 6-state universal Turing machines, Journal of the Association for Computing Machinery (ACM) 8 (4) (1961) 476–483.

[29] S. Watanabe, Four-symbol five-state universal Turing machine, Information Processing Society of Japan Magazine 13 (9) (1972) 588–592.

[30] D. Woods, T. Neary, Small semi-weakly universal Turing machines, Fundamenta Informaticae 91 (1) (2009) 179–195.