On the computational complexity of spiking neural P systems

Turlough Neary*

Boole Centre for Research in Informatics,
University College Cork, Ireland.
tneary@cs.may.ie

Abstract. It is shown here that there is no standard spiking neural P system that simulates Turing machines with less than exponential time and space overheads. The spiking neural P systems considered here have a constant number of neurons that is independent of the input length. Following this, we construct a universal spiking neural P system with exhaustive use of rules that simulates Turing machines in linear time and has only 10 neurons.

1 Introduction

Since their inception inside of the last decade P systems [16] have spawned a variety of hybrid systems. One such hybrid, that of spiking neural P systems (SN P systems) [3], results from a fusion with spiking neural networks. These systems have been shown to be computationally universal.

In this work the time/space computational complexity of SN P systems is examined. We begin by showing that counter machines simulate standard SN P systems with linear time and space overheads. Fischer et al. [2] have previously shown that counter machines require exponential time and space to simulate Turing machines. Thus it immediately follows that there exists no SN P system that simulates Turing machines with less than exponential time and space overheads. These results are for SN P systems that have a constant number of neurons independent of the input length.

Extended SN P systems with exhaustive use of rules were proved computationally universal in [4]. This was achieved by showing that they simulate counter machines in linear time. Thus, using the simulation algorithm in [4] gives an exponential time overhead when simulating Turing machines. Zhang et al. [18] gave a small universal SN P system with exhaustive use of rules (without delay) that has 125 neurons. This system was proved universal by giving a linear time simulation of Korec's [6] 23-instruction universal register machine. Because Korec's machine has a double-exponential time overhead when simulating Turing machines, the 125 neuron system also suffers from a double-exponential slowdown when simulating Turing machines. In an earlier version [10] of the work

* The author is funded by Science Foundation Ireland Research Frontiers Programme grant number 07/RFP/CSMFz1.

number of neurons	simulation time/space	type of rules	exhaustive use of rules	author
84	double-exponential	standard	no	Păun and Păun [15]
67	double-exponential	standard	no	Zhang et al. [19]
17	exponential	standard†	no	Neary [12]
11	exponential	standard†	no	Neary [14]
49	double-exponential	extended†	no	Păun and Păun [15]
41	double-exponential	extended†	no	Zhang et al. [19]
18	exponential	extended†	no	Neary [9, 11]*
12	double-exponential	extended†	no	Neary [11]
4	exponential	extended†	no	Neary [13]
3	exponential	extended‡	no	Neary [13]
125	double-exponential/ triple-exponential	extended†	yes	Zhang et al. [18]
18	polynomial/exponential	extended	yes	Neary [10]
10	**linear/exponential**	**extended**	**yes**	**Section 5**

Table 1. Small universal SN P systems. The "simulation time" column gives the overheads used by each system when simulating a standard single tape Turing machine. † indicates that there is a restriction of the rules as delay is not used and ‡ indicates that a more generalised output technique is used. *The 18 neuron system is not explicitly given in [11]; It is mentioned at the end of the paper and is easily derived from the other system in [11]. Also, it is presented in [9].

we present here, we gave an extended SN P system with exhaustive use of rules that simulates Turing machines in *polynomial time* and has *18 neurons*. Here we improve on this result to give an extended SN P system with exhaustive use of rules that simulates Turing machines in *linear time* and has only *10 neurons*. It is worth noting that prior to our results the most time-efficient universal SN P systems simulated Turing machines with exponential time overheads. Further explanation of the time/space complexity overheads can be found in the last two paragraphs of Section 3.

The time/space complexity of small universal SN P systems[1] and a brief history of the area is given in Table 1. Lower bounds on the number of neurons need for universality were given in [13]. It was shown that there exists no universal SN P system with extended rules and only 3 neurons when the output technique is standard, and that there exists no universal SN P system with extended rules and only 2 neurons when the output technique is generalised. Thus, the 4- and 3-neurons systems mentioned in Table 1 are the smallest possible universal systems of their kind, where size is the number of neurons.

[1] In a similar Table given in [13] the time/space complexity given for a number of the systems is incorrect due to an error copied from Korec's paper [6]. For more see the last paragraph of Section 3.

Chen et al. [1] have shown that with exponential pre-computed resources SAT is solvable in constant time with SN P systems. Leporati et al. [8] gave a semi-uniform family of extended SN P systems that solve the SUBSET SUM problem in constant time. In other work, Leporati et al. [7] gave a uniform family of maximally parallel SN P systems with more general rules that solve the SUBSET SUM problem in polynomial time. All the above solutions to NP-complete problems rely on families of SN P systems. Specifically, the size of the problem instance determines the number of neurons in the SN P system that solves that particular instance. This is similar to solving problems with uniform circuits families where each input size has a specific circuit that solves it. Ionescu and Sburlan [5] have shown that SN P systems simulate circuits in linear time.

In the next two sections we give definitions for SN P systems and counter machines and explain the operation of both. Following this, in Section 4, we prove that counter machines simulate SN P systems in linear time, thus proving that there exists no universal SN P system that simulates Turing machines in less than exponential time. In Section 5 we present our universal SN P system with exhaustive use of rules that simulates Turing machines in linear time and has only 10 neurons. Finally, we end the paper with some discussion and conclusions.

2 Spiking neural P systems

Definition 1 (Spiking neural P systems). *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.*

In the same manner as in [15], 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. The sequence w is read from left to right one symbol at each timestep. If the read symbol is 1 then a spike enters the input neuron on that timestep.

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 rule r is executed at time t, 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 at least one applicable rule at time t. Thus, while the application of rules in each individual neuron is sequential, the neurons operate in parallel with each other.

Note from 2(b)i of Definition 1 that there may be more than one rule of the form $E/s^b \to s;d$ applicable in a single neuron at a given time. If this is the case, then the next rule to execute is chosen nondeterministically. The output is the time between the first and second spike sent out of the output neuron. If the binary sequence $w \in \{0,1\}^*$ is given as input to Π, then the output of the computation is given by $\Pi(w)$.

An extended SN P system has rules of the more general form $E/s^b \to s^p;d$, where $b \geqslant p \geqslant 0$. Note that if $p = 0$ then $E/s^b \to s^p;0$ is a forgetting rule. For all rules of this type (with $p = 0$) there is no delay (i.e. $d = 0$). Note that this type of forgetting rule is also found in [4, 18], and is more general than the forgetting rules found in [12, 13, 15, 19]. Forgetting rules in [12, 13, 15, 19] are of the form $s^e \to \lambda$. In [4, 18] more than one forgetting rule may be applicable in a single neuron at a given timestep; however, we do not need this particular generalisation in this work.

Now we explain the operation of SN P systems with exhaustive use of rules [4]. In each neuron with at least one applicable rule, a single rule is chosen using the technique given above. No more than one rule is applied in each neuron at a given timestep, however, its application is exhaustive. That is to say, if a single application of rule r uses b spikes and there are k spikes in the neuron at time t, then r is applied g times at time t, where $g = \lfloor k/b \rfloor$. For example, if neuron σ_i contains k spikes and the extended rule $E/s^b \to s^p;d$ is to be applied, then the neuron σ_i sends out gp spikes after d timesteps leaving u spikes in σ_i, where $k = bg + u$, $u < b$ and $k, g, u \in \mathbb{N}$. Thus, in an SN P system with exhaustive use of rules there is no upper bound placed on the number of spikes that may be transmitted by a synapse in a single timestep. In the sequel, for SN P systems with exhaustive use of rules, there is no bound placed on the number of spikes that can be received from the environment by the input neuron in a single timestep. This is a generalisation on the input allowed by Ionescu et al. [4]. We discuss our reasons for this choice in the conclusion.

Here the input to an SN P system with exhaustive use of rules is a finite sequence of numbers $(x_1, x_2, \ldots, x_n)$ where x_i is the number of spikes read into the input neuron at timestep i. The output of an SN P system with exhaustive use of rules is the number of spikes sent out of the output neuron the first time it fires. For an SN P system Π with exhaustive use of rules, if the sequence $(x_1, x_2, \ldots, x_n)$ is given as input to Π, then the output of the computation is given by $\Pi(x_1, x_2, \ldots, x_n)$.

In this work, the end of an SN P system computation is the timestep t on which the output neuron finishes sending the output to the environment. We are not concerned with whether or not the system still has applicable rules in the neurons after time t.

In the sequel each spike in an SN P system represents a single unit of space. The maximum number of spikes in an SN P system at any given timestep during a computation is the space used by the system.

3 Counter machines, SN P systems and time/space computational complexity

The definition we give for counter machine is similar to that of Fischer et al. [2].

Definition 2 (Counter machine).
A counter machine is a tuple $C = (z, c_m, Q, q_0, q_h, \Sigma, f)$*, where* z *gives the number of counters,* c_m *is the output counter,* $Q = \{q_0, q_1, \cdots, q_h\}$ *is the set of states,* $q_0, q_h \in Q$ *are the initial and halt states respectively,* Σ *is the input alphabet and* f *is the transition function*

$$f : (\Sigma \times Q \times g(i)) \rightarrow (\{Y, N\} \times Q \times \{INC(i), DEC(i), NULL\})$$

where $g(i)$ *is a binary valued function and* $1 \leqslant i \leqslant z$*,* Y *and* N *control the movement of the input read head, and* $INC(i)$*,* $DEC(i)$*, and* $NULL$ *indicate the operation to carry out on counter* c_i*.*

Each counter c_i stores a natural number value x. If $x > 0$ then $g(i)$ is true and if $x = 0$ then $g(i)$ is false. The input to the counter machine is read in from an input tape with alphabet Σ. The movement of the scanning head on the input tape is one-way so each input symbol is read only once. When a computation begins the scanning head is over the leftmost symbol α of the input word $\alpha w \in \Sigma^*$ and the counter machine is in state q_0. We give three examples below to explain the operation of the transition function f.

- $f(\alpha, q_j, g(i)) = (Y, q_k, INC(i))$ move the read head right on the input tape to read the next input symbol, change to state q_k and increment the value x stored in counter c_i by 1.
- $f(\alpha, q_j, g(i)) = (N, q_k, DEC(i))$ do not move the read head, change to state q_k and decrement the value x stored in counter c_i by 1. Note that $g(i)$ must evaluate to true for this rule to execute.
- $f(\alpha, q_j, g(i)) = (N, q_k, NULL)$ do not move the read head and change to state q_k.

A single application of f is a timestep. Thus in a single timestep only one counter may be incremented or decremented.

Our definition for counter machine, given above, is more restricted than the definition given by Fischer [2]. In Fischer's definition INC and DEC instructions may be applied to every counter in the machine in a single timestep. Clearly the more general counter machines of Fischer simulate our machines with no extra space or time overheads. Fischer has shown that counter machines are exponentially slow in terms of computation time as the following theorem illustrates.

Theorem 1 (Fischer [2]). *There is a language L, real-time recognizable by a one-tape TM, which is not recognizable by any k-CM in time less than $T(n) = 2^{\frac{n}{2k}}$.*

In Theorem 1 a one-tape TM is an offline Turing machine with a single read only input tape and a single work tape, a k-CM is a counter machine with k counters, n is the input length and real-time recognizable means recognizable in n timesteps. For his proof Fischer noted that the language $L = \{waw^r \mid w \in \{0,1\}^*\}$, where w^r is w reversed, is recognisable in n timesteps on a one-tape offline Turing machine. He then noted, that time of $2^{\frac{n}{2k}}$ is required to process input words of length n due to the unary data storage used by the counters of the k-CM. Note that Theorem 1 also holds for nondeterministic counter machines as they use the same unary storage method.

In Table 1, the systems with 3, 4, 11 and 17 neurons and the 18-neuron system given in [9, 11] all simulate 3-CMs with linear time and space overheads. It is known that 3-CMs simulate Turing machines with exponential time overheads, and so it immediately follows that these SN P systems simulate Turing machines with exponential time and space overheads. The 12-neuron system given in Table 1 simulates 2-CMs with linear time and space overheads. Using current algorithms 2-CMs simulate Turing machines with a double-exponential time overhead [17], and thus this 12-neuron system simulates Turing machines with double-exponential time and space overheads.

In [6] 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: Let f be a unary partial recursive function and let y be its input. 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 $2^{f(y)}$. As mentioned earlier, using current algorithms 2-CMs simulate Turing machines with a double-exponential time overhead, and so because of Step 2 above, Korec's R3a-machines and 32-instruction machine suffer from a double-exponential slowdown when simulating Turing machines. All of the SN P systems in [15, 18, 19] simulate the 23-instruction (the halt instruction is included here) universal counter machine given by Korec in [6]. 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 [15, 19] 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 [18] 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-CMs in polynomial time.

4 Non-deterministic counter machines simulate SN P systems in linear time

In this section we consider SN P systems that apply their rules in the normal manner (i.e. no exhaustive use of rules).

Theorem 2. *Let Π be an SN P system with standard rules that completes its computation in time T and space S. Then there is a nondeterministic counter machine C_Π that simulates the computation of Π in time $O(T)$ and space $O(S)$.*

Proof idea Before giving the proof of Theorem 2, we sketch the main idea behind the proof. Each neuron σ_i from the SN P system Π is simulated by a counter c_i from the counter machine C_Π. If a neuron σ_i contains y spikes, then the counter will have value y. A single synchronous update of all the neurons at a given timestep t is simulated as follows: If the number of spikes in a neuron σ_i is decreasing by b spikes in order to execute a rule, then the value y stored in the simulated neuron c_i is decremented b times using $DEC(i)$ to give $y - b$. This process is repeated for each neuron that executes a rule at time t. If neuron σ_i fires at time t and has synapses to neurons $\{\sigma_{i_1}, \ldots \sigma_{i_v}\}$, then for each open neuron σ_{i_j} in $\{\sigma_{i_1}, \ldots \sigma_{i_v}\}$ at time t we increment the simulated neuron c_{i_j} using $INC(i_j)$. This process is repeated until all firing neurons have been simulated. This simulation of the synchronous update of Π at time t is completed by C_Π in constant time. Thus we get the linear time bound given in Theorem 2.

Proof. Let $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$ be an SN P system that completes it computation in time T. We explain the operation of a nondeterministic counter machine C_Π that simulates the operation of Π in time $O(T)$ and space $O(S)$.

There are $m + 1$ counters $c_1, c_2, c_3, \cdots, c_m, c_{m+1}$ in C_Π. Each counter c_i emulates the activity of a neuron σ_i. If σ_i contains y spikes, then counter c_i will store the value y.

4.1 Input encoding

It is sufficient for C_Π to have a binary input tape. The value of the binary word $w \in \{1, 0\}^*$ that is placed on the input tape to be read into C_Π is identical to the binary sequence read in from the environment by the input neuron σ_1. A single symbol is read from the input tape at each simulated timestep. On timesteps when a 1 is read, counter c_1 (the simulated input neuron) is incremented to

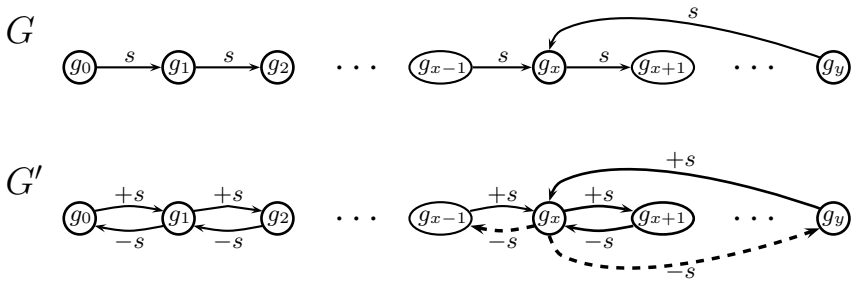

Fig. 1. Deterministic finite automaton G decides if a particular rule is applicable in a neuron given the number of spikes in the neuron at a given time in the computation. A single s-transition occurs for each spike in the neuron. For the purposes of generality we do not explicitly give the accept states of G. State transition graph of Machine G', which has a single counter, and keeps track of the movement of spikes into and out of the neuron, thus deciding whether or not a particular rule is applicable *at each timestep* in the computation. Each $+s$-transition represents a single spike entering the neuron and each $-s$-transition represents a single spike being used up in the neuron.

simulate a spike entering the input neuron from the environment. At the start of the computation, before the input is read in, each counter simulating σ_i is incremented n_i times to simulate the n_i spikes in each neuron given by 2(a) of Definition 1. This takes a constant amount of time.

4.2 Deciding which rules are applicable in neuron σ_i

We begin by explaining how C_Π deals with rules of the form $r = E/s^b \rightarrow s; d$ where the language $L(E)$ is infinite. The applicability of a rule $r = E/s^b \rightarrow s; d$ in an open neuron σ_i is dependent on a regular expression E and the value of b. Given E and b we can easily obtain a regular expression E_b that generates no words of length $< b$. Recall that for every regular expression E_b there exists a deterministic finite automaton G (given in Figure 1) that accepts the language $L(E_b)$, and thus decides if the number of spikes in σ_i permits the application of r in σ_i at a given timestep in the computation. Note that because $L(E_b)$ is an infinite regular language over a unary alphabet, any deterministic finite automata (such as G) that accepts $L(E_b)$ will contain exactly one cycle (i.e. spanning g_x to g_y). Automata G operates as follows, if σ_i contains z spikes at some timestep, then given the word s^z as input G will decide whether or not r is applicable at that timestep. Given the input s^z the computation of G will halt in state g_z if $z < y$ or in state $g_{x+((z-x) \mod (y-x+1))}$ if $z > y$. If computation halts in an accept state rule r is applicable.

Using G we will construct a machine G' that decides if rule r is applicable at each timestep by recording spikes entering and being used up in neuron σ_i. Machine G' has a single counter c' and (instead of a 0 test) we allow G' to test if the value of c' is $x - 1$ in a single timestep. The value stored in c' is equal to

the number of spikes in σ_i. The state transition graph of G', which is given in Figure 1, 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 being used up in the neuron. For each s-transition from a state g_i to a state g_j in G there is a new $-s$-transition going from state g_j to state g_i in G'. Now we describe the operation of G'. If G' is in state g_j and it reads $+s$, then it increments the value in counter c' by one and moves to state g_{j+1} if $j \neq y$ or to state g_x if $j = y$. A $+s$-transition simulates one spike entering neuron σ_i. If G' is in state $g_j \neq g_x$ and it reads $-s$, then it decrements the value in counter c' by 1 and moves to state g_{j-1}. If G' is in state g_x and it reads $-s$, then it decrements the value in counter c' and if the new value in counter c' is $x-1$ it enters state g_{x-1} otherwise state g_y is entered. A $-s$-transition simulates that the number of spikes in neuron σ_i is decreasing by one. Note that because G is a deterministic finite automaton with a unary alphabet that accepts an infinite language, it will contain exactly one cycle, and thus G' will contain only one state (g_x) that has two $-s$-transitions. To record the number of spikes in σ_i at each timestep, G' will execute one $+s$-transition for each spike entering σ_i and one $-s$-transition for each spike used up in σ_i. If the number of spikes in σ_i is z, then the state occupied by G' will be the same as the state G halts in when given s^z as input. Thus, G' decides if the number of spikes in σ_i permits the application of r in σ_i at each timestep during the computation.

During a simulation counter machine C_{Π} determines which rules are applicable in σ_i by recording the states of G' in the counter machine states. If C_{Π} is simulating a $-s$-transition from state g_x the counter machine decrements counter c_i (the counter simulating neuron σ_i) and checks if the new value in c_i is $x-1$. This check is carried out by decrementing c_i exactly $x-1$ times using the $DEC(i)$ instruction. If $c_i = 0$ after the counter has been decremented $x-1$ times, then C_{Π} simulates state g_{x-1} otherwise state g_y is simulated. Immediately after this counter c_i is incremented $x-1$ times to restore the correct value in c_i.

Let neuron σ_i have the greatest number l of rules of any neuron in Π. The applicability of rules $r_1, r_2, \cdots, r_l$ in σ_i is decided by the automata $G'_1, G'_2, \cdots, G'_l$. C_{Π} will record the current states of at most l different G' automata to simulate neuron σ_i. There are m neurons in Π. Thus, each state in our counter machine remembers the current states of at most ml different G' automata in order to determine which rules are applicable in each neuron at a given time.

The applicability of all rules of the form $r = E/s^b \rightarrow s; d$ where $L(E)$ is a finite language and all rules of the form $s^e \rightarrow \lambda$ is tested simultaneously. Let p be the longest possible word from all the regular expressions of rules of these types, then counter c_i is decremented at most p times to check if any of these rules are applicable. For example, if after the e^{th} decrement $c_i = 0$, then rule $s^e \rightarrow \lambda$ is applicable and the value of c_i remains at 0. If a rule of the form $r = E/s^b \rightarrow s; d$ is to be applied, c_i is restored to its original value before simulating the rule.

4.3 Algorithm overview

Next we explain the operation of C_Π by explaining how it simulates the synchronous update of all neurons in Π at an arbitrary timestep t. The algorithm has 3 stages. A single iteration of Stage 1 identifies which applicable rule to simulate in a simulated open neuron. Then the correct number k of simulated spikes are removed by decrementing the counter k times ($k = b$ or $k = e$ in 2(b) of Definition 1). Stage 1 is iterated until all simulated open neurons have had the correct number of simulated spikes removed. A single iteration of Stage 2 identifies all the synapses leaving a firing neuron and increments every counter that simulates an open neuron at the end of one of these synapses. Stage 2 is iterated until all firing neurons have been simulated by incrementing the appropriate counters. Stage 3 reads in the input, updates the output counter and simulates a decrement of the d values for each neuron with $d > 0$.

4.4 Algorithm details

Stage 1. Identify rules to be simulated and remove spikes from neurons Recall that the value of d indicates when a closed neuron will open and fire. Let d_i denote the current value of d for neuron σ_i. We record whether or not a neuron σ_i is due to fire with a predicate p_i, where p_i is true iff σ_i fires in the current timestep. The values d_i and p_i for each neuron σ_i are stored in the current state of the counter machine.

Stage 1 begins by determining which rule to simulate in counter c_{i_1} where $i_1 = min\{i \,|d_i = 0, \neg p_i\}$. The technique used for deciding which rules (if any) are applicable is given in Section 4.2. The applicability of rules of the form $r = E/s^b \to s; d$ where the language $L(E)$ is infinite is already known as the current state of each G' automaton is stored in the state of C_Π. The applicability of all other rules is determined using the technique given in the last paragraph of Section 4.2. If there are no applicable rules in σ_i, Stage 1 is repeated for counter c_{i_2} where $i_2 = min\{i \,|\, i_2 > i_1, d_i = 0, \neg p_i\}$.

Recall from the last paragraph of Section 4.2, that if a rule of the form $s^e \to \lambda$ is applicable, counter c_{i_1} will have value 0 immediately after testing the applicability of that rule. Thus, to complete the simulation of $s^e \to \lambda$, it only remains to update the state of C_Π to recorded that the current state of each G' automaton is g_0. If, on the other hand, more than one rule of the form $r = E/s^b \to s; d$ is applicable, the counter machine nondeterministically chooses which rule to simulate. Once a rule is chosen, the value in counter c_{i_1} is decremented k times, where k is the number of spikes used up in the neuron when that rule is executed. With each decrement of c_{i_1}, the current state of the counter machine records the new state of each of the automata G'_1 to G'_l of σ_{i_1}. Note that if C_Π is simulating a $-s$-transition on a state with 2 possible $-s$-transitions (e.g. q_x from G' in Figure 1), then the technique for updating the state from paragraph 3 of Section 4.2 is used. After k decrements of c_{i_1}, the simulation of the removal of k spikes from neuron σ_{i_1} is complete. If a rule of the form $r = E/s^b \to s; d$ is being simulated and $d > 0$, then $d_i = d$ is recorded

in the state of C_Π. Alternatively, if $d = 0$ for rule r the state of C_Π records that the value of p_{i_1} is set to true.

When the simulation of the removal of k spikes from neuron σ_{i_1} is complete, the above process is repeated with counter c_{i_2}, where $i_2 = min\{i \mid i_2 > i_1, d_i = 0, \neg p_i\}$. This process is iterated until every simulated open neuron with an applicable rule at time t has had the correct number of simulated spikes removed.

Stage 2. Simulate spikes This stage of the algorithm begins by simulating spikes traveling along synapses of the form (i_1, j) where $i_1 = min\{i \mid p_i\}$ (p_i indicates neuron σ_i is firing). Let $\{(i_1, j_1), (i_1, j_2), \cdots, (i_1, j_k)\}$ be the set of synapses leaving σ_{i_1}, where $j_u < j_{u+1}$ and $d = 0$ in σ_{j_u} at time t (if $d = 0$ the neuron is open and may receive spikes). Then the sequence of instructions INC(j_1), INC(j_2), $\cdots$, INC(j_k) is executed, thus incrementing any counter (simulated neuron) that receives a simulated spike. Following this, the state of the counter machine records that the value of p_{i_1} is set to false.

The above process is repeated for synapses of the form (i_2, j), where $i_2 = min\{i \mid i_2 > i_1, p_i\}$. This process is iterated until every simulated neuron c_i that is open has been incremented once for each spike σ_i receives at time t.

Stage 3. Reading input, decrementing *d*, updating the output counter and halting If the entire word w has not been read from the input tape then the next symbol is read. If this is the case and the symbol read is a 1, then counter c_1 is incremented, thus simulating a spike entering the input neuron σ_1 from the environment. In this stage the state of the counter machine changes to record the fact that each d_i is decremented to $d_i - 1$. Recall that d_i stores the number of timesteps until a currently closed neuron σ_i will open (and fire). Thus if $d_i = 1$ and we decrement its value to $d_i = 0$, the counter machine state records that p_i is set to true indicating that neuron σ_i will fire during the next timestep. If counter c_m, which simulates the output neuron, has spiked only once prior to the simulation of timestep $t + 1$, then this stage will also increment the output counter c_{m+1}. If, during the simulation of timestep t, counter c_m has simulated a firing rule for the second time in the computation, then the counter machine enters the halt state. When the halt state is entered the number stored in counter c_{m+1} is equal to the unary output that is given by the time interval between the first two timesteps where σ_m has spiked.

4.5 Space/time analysis

Space analysis The input word on the binary tape of C_Π is identical to the length of the binary sequence read in by the input neuron of Π. Counters c_1 to c_m use the same space as neurons σ_1 to σ_m. Counter c_{m+1} uses the same amount of space as the unary output of the computation of Π. Thus C_Π simulates Π in space of $O(S)$.

Time analysis The simulation involves 3 stages.

Stage 1. Let q be the maximum of all the values for x, b and e, where x is the first loop state of any G automaton (see g_x in Figure 1) and b and e are given by the neural rules of Π (see 2(b) Definition 1). In order to simulate the deletion of a single spike in the worst case the counter will have to by decremented $q-1 \geqslant x-1$ times and incremented $q-1$ times (see paragraph 3 of Section 4.2). This is repeated a maximum of q times (where $b \leqslant q$ is the number of spikes removed). Thus, a single iteration of Stage 1 takes $O(q^2)$ time. Stage 1 is iterated a maximum of m times per simulated timestep, giving $O(q^2 m)$ time.

Stage 2. The maximum number of synapses leaving a neuron i is $m-1$. A single spike traveling along a neuron is simulated in one step. Stage 2 is iterated a maximum of m times per simulated timestep, giving $O(m^2)$ time.

Stage 3. Takes a small constant number of steps.

A single timestep of Π is simulated by C_Π in $O(q^2 m + m^2)$ time and T timesteps of Π are simulated by C_Π in $O(Tq^2 m + Tm^2)$ time. Recall that q and m are constants dependent on Π, thus C_Π simulates T timesteps of Π in linear time $O(T)$. □

The following is an immediate corollary of Theorems 1 and 2.

Corollary 1. *There exists no universal SN P system with standard rules that simulates Turing machines with less than exponential time and space overheads.*

The results in Theorem 2 and the above corollary can easily be generalised to SN P systems with extended rules.

In Theorem 2, SN P systems that use a standard output technique are simulated. Recall that using this technique the SN P system outputs a single number that is given by the time interval between the first and second spikes sent out of the output neuron. Our algorithm can be adapted to deal with more general output techniques without the need for an increase in time or space resources. For example, if a constant number z of spikes is always sent out of the output neuron to give a sequence of $z-1$ numbers (instead of 1 number) as output, then our algorithm will use an extra $z-2$ counters to store these extra numbers. If the output is a spike train $w \in \{0,1\}^*$, then the counter machine could be augmented with a write-only binary output tape that stores w as a binary word. Note that such a counter machine would still require exponential time to simulate Turing machines. Our algorithm could also accommodate different halting techniques. For example, if the computation halts when there is no rule applicable in a neuron, then our algorithm could be altered to halt at the end of Stage 1 if no applicable rule is found.

5 A universal SN P system that is both small and time efficient

In this section we construct a universal SN P system that applies exhaustive use of rules, has only 10 neurons, and simulates any single-tape Turing machine in linear time.

Theorem 3. *Let M be a single tape Turing machine with $|A|$ symbols and $|Q|$ states that runs in time T. Then there is a universal extended SN P system Π_M with exhaustive use of rules that simulates the computation of M in time $O(T \log(|Q||A|))$ and space $O([|Q||A|]^T)$ and has only 10 neurons.*

If the reader would like to get a quick idea of how our SN P system with 10 neurons operates they should skip to the algorithm overview in Subsection 5.3 of the proof.

Proof. We give an SN P system Π_M that simulates an arbitrary single-tape Turing machine M in linear time and exponential space. Π_M is given by Figure 3 and Tables 2 and 3. The algorithm for Π_M is deterministic and is mainly concerned with the simulation of an arbitrary transition rule. Without loss of generality, we insist that M always finishes its computation with the tape head at the leftmost end of the tape contents. Let M be any single tape Turing machine with symbols $\alpha_1, \alpha_2, \ldots, \alpha_{|A|}$ and states $q_1, q_2, \ldots q_{|Q|}$, blank symbol α_1, and halt state $q_{|Q|}$.

5.1 Encoding a configuration of Turing machine M

Each configuration of M is encoded as three natural numbers using a well known technique. A configuration of M is given by the following equation

$$C_k = \boldsymbol{q_r},\ \cdots \alpha_1\alpha_1\alpha_1\, a_{-x} \cdots a_{-3}a_{-2}a_{-1}\underline{a_0}a_1a_2a_3 \cdots a_y\, \alpha_1\alpha_1\alpha_1 \cdots \qquad (1)$$

where q_r is the current state, each a_i is a tape cell of M and the tape head of M, given by an underline, is over a_0. Also, tape cells a_{-x} and a_y both contain α_1, and the cells between a_{-x} and a_y include all of the cells on M's tape that have either been visited by the tape head prior to configuration C_k or contain part of the input to M. The cells a_{-x} and a_y are included in the encoding for technical reasons that will become apparent towards the end of the proof.

In the sequel the encoding of object p is given by $\langle p \rangle$. The tape symbols $\alpha_1, \alpha_2, \ldots, \alpha_{|A|}$ of M are encoded as $\langle \alpha_1 \rangle = 1, \langle \alpha_2 \rangle = 3, \ldots, \langle \alpha_{|A|} \rangle = 2|A| - 1$, respectively, and the states $q_1, q_2, \ldots, q_{|Q|}$ are encoded as $\langle q_1 \rangle = 2|A|, \langle q_2 \rangle = 4|A|, \ldots, \langle q_{|Q|} \rangle = 2|Q||A|$, respectively. The contents of each tape cell a_i in configuration C_k is encoded as $\langle a_i \rangle = \langle \alpha \rangle$, where α is a tape symbol of M. The tape contents in Equation (1) to the left and right of the tape head are respectively encoded as the numbers $X = \sum_{i=1}^{x} z^i \langle a_{-i} \rangle$ and $Y = \sum_{j=1}^{y} z^j \langle a_j \rangle$ where $z = 2^v$ and $v = \lceil \log_2(2|Q||A| + 2|A|) \rceil$. Thus the entire configuration C_k is encoded as three natural numbers via the equation

$$\langle C_k \rangle \ = \ (X,\ Y,\ \langle q_r \rangle + \langle \alpha_i \rangle) \qquad (2)$$

where $\langle C_k \rangle$ is the encoding of C_k from Equation (1) and α_i is the symbol being read by the tape head in cell a_0.

A transition rule $q_r, \alpha_i, \alpha_j, D, q_u$ of M is executed on C_k as follows: If the current state is q_r and the tape head is reading the symbol α_i in cell a_0, α_j the write symbol is printed to cell a_0, the tape head moves one cell to the left to a_{-1} if $D = L$ or one cell to the right to a_1 if $D = R$, and q_u becomes the new current state. A simulation of transition rule $q_r, \alpha_i, \alpha_j, D, q_u$ on the encoded configuration $\langle C_k \rangle$ from Equation (2) is given by the equation

$$\langle C_{k+1} \rangle = \begin{cases} \big(\frac{X}{z} - (\frac{X}{z} \mod z),\ zY + z\langle \alpha_j \rangle,\ \langle q_u \rangle + (\frac{X}{z} \mod z)\big) \\ \big(zX + z\langle \alpha_j \rangle,\ \frac{Y}{z} - (\frac{Y}{z} \mod z),\ \langle q_u \rangle + (\frac{Y}{z} \mod z)\big) \end{cases} \tag{3}$$

where configuration C_{k+1} results from executing a single transition rule on configuration C_k, and $(b \mod c) = d$ where $d < c$, $b = ec + d$ and $b, c, d, e \in \mathbb{N}$. In Equation (3) the top case is simulating a left move transition rule and the bottom case is simulating a right move transition rule. In the top case, following the left move, the sequence to the right of the tape head is longer by 1 tape cell, as cell a_0 is added to the right sequence. Cell a_0 is overwritten with the write symbol α_j, and thus we compute $zY + z\langle \alpha_j \rangle$ to simulate cell a_0 becoming part of the right sequence. Also, in the top case the sequence to the left of the tape head is getting shorter by 1 tape cell, thus we compute $\frac{X}{z} - (\frac{X}{z} \mod z)$. The rightmost cell of the left sequence a_{-1} is the *new* tape head location and the tape symbol it contains is encoded as $(\frac{X}{z} \mod z)$. Thus the value $(\frac{X}{z} \mod z)$ is added to the new encoded current state $\langle q_u \rangle$. For the bottom case, a right move, the sequence to the right gets shorter which is simulated by $\frac{Y}{z} - (\frac{Y}{z} \mod z)$ and the sequence to the left gets longer which is simulated by $zX + z\langle \alpha_j \rangle$. The leftmost cell of the right sequence a_1 is the new tape head location and the tape symbol it contains is encoded as $(\frac{Y}{z} \mod z)$.

5.2 Input to Π_M

Here we give an explanation of how the input is read into Π_M. We also give a rough outline of how the input to Π_M is encoded in linear time.

A configuration C_k given by Equation (2) is read into Π_M as follows: All the neurons of the system initially have no spikes with the exception of σ_{10} which has 31 spikes. The input neuron σ_5 receives $X + 2$ spikes at the first timestep t_1, Y spikes at time t_2, and $\langle q_r \rangle + \langle \alpha_i \rangle$ spikes at time t_4. We explain how the system is initialised to encode an initial configuration of M by giving the number of spikes in each neuron and the rule that is to be applied in each neuron at time t. Thus at time t_1 we have

$$\begin{aligned} t_1 : \sigma_5 &= X + 2, & s^2(s^z)^*/s &\to s; 0, \\ \sigma_{10} &= 31, & s^{31}/s^{16} &\to \lambda. \end{aligned}$$

where on the left $\sigma_j = k$ gives the number k 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

$s^2(s^z)^*/s \to s; 0$ in neuron σ_5 and the rule $s^{31}/s^{16} \to \lambda$ in neuron σ_{10} at time t_1 we get

$$
\begin{aligned}
t_2 : \sigma_4 &= X + 2, & s^2(s^z)^*/s^z &\to s^z; 1,\\
\sigma_5 &= Y, & s^{2z}(s^z)^*/s &\to s; 0,\\
\sigma_6, \sigma_7, \sigma_8, \sigma_9 &= X + 2, & s^2(s^z)^*/s &\to \lambda,\\
\sigma_{10} &= 15, & s^{15}/s^8 &\to \lambda.
\end{aligned}
$$

$$
\begin{aligned}
t_3 : \sigma_4 &= X + 2, & s^2(s^z)^*/s^z &\to s^z; 0,\\
\sigma_6 &= Y, & (s^z)^*/s &\to s; 0,\\
\sigma_7, \sigma_8, \sigma_9 &= Y, & (s^z)^*/s &\to \lambda,\\
\sigma_{10} &= 7, & s^7/s^4 &\to \lambda.
\end{aligned}
$$

$$
\begin{aligned}
t_4 : \sigma_1 &= X, & & \\
\sigma_2 &= Y, & & \\
\sigma_4 &= 2, & s^2/s^2 &\to \lambda,\\
\sigma_5 &= \langle q_r \rangle + \langle \alpha_i \rangle, & (s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to s; 0,\\
\sigma_{10} &= 3, & s^3/s^2 &\to \lambda.
\end{aligned}
$$

$$
\begin{aligned}
t_5 : \sigma_1 &= X, & & \\
\sigma_2 &= Y, & & \\
\sigma_4, \sigma_6 &= \langle q_r \rangle + \langle \alpha_i \rangle, & & \\
\sigma_7, \sigma_8, \sigma_9 &= \langle q_r \rangle + \langle \alpha_i \rangle, & s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to \lambda,\\
\sigma_{10} &= 1, & s/s &\to s; \log_2(z) + 2.
\end{aligned}
$$

Note that σ_4 is closed at time t_2 as there is a delay of 1 on the rule $(s^2(s^z)^*/s^z \to s^z; 1)$ to be executed in σ_4. This prevents the Y spikes from entering neuron σ_4 when σ_5 fires at time t_2. At time t_5 the SN P system has X spikes in σ_1, Y spikes in σ_2, and $\langle q_r \rangle + \langle \alpha_i \rangle$ spikes in σ_4 and σ_6. Thus at time t_5 the SN P system encodes an initial configuration of M.

Given an initial configuration of M, the input to our SN P system in Figure 3 is computed in linear time. To do this we must compute the three numbers that give $\langle C_k \rangle$ from Equation 2 in linear time. The number X is computed as follows: Given a sequence $a_{-x}a_{-x+1}\ldots a_{-2}a_{-1}$ the sequence $w = \langle a_{-x} \rangle 0^{\log_2(z)-1} \langle a_{-x+1} \rangle 0^{\log_2(z)-1} \ldots \langle a_{-2} \rangle 0^{\log_2(z)-1} \langle a_{-1} \rangle 0^{\log_2(z)-1} 2$ is easily computed in a time that is linear in x. The subsequence $0^{\log_2(z)-1}$ after each $\langle a_{-x} \rangle$ allows us to simplify the SN P system Π_{input} in Figure 2 that completes the computation of X. Given the sequence w, Π_{input} converts it into the X spikes that form part of the input to our system in Figure 3. We give a rough idea of

how Π_{input} operates (if the reader wishes to pursue a more detailed view the rules for Π_{input} are to be found in Table 4). The input neuron of Π_{input} receives the sequence w as a sequence of spikes and no-spikes. On each timestep where $\langle a \rangle$ is read $\langle a \rangle$ spikes are passed to the input neuron σ_1, and on each timestep where 0 is read no spikes are passed to the input neuron. Thus at timestep t_1 neuron σ_1 receives $\langle a_{-x} \rangle$ spikes, and at timestep t_2 neurons σ_2, σ_3, and σ_4 each receive $\langle a_{-x} \rangle$ spikes from σ_1. Following timestep t_2, the number of spikes in neurons σ_2, σ_3, and σ_4 double with each timestep. So at timestep $t_{\log_2(z)+1}$ the number of spikes in each of the neurons σ_2, σ_3, and σ_4 is $\frac{z}{2}\langle a_{-x} \rangle$. At timestep $t_{\log_2(z)+1}$ neurons σ_2, σ_3 and σ_4 also receive $\langle a_{-x+1} \rangle$ spikes from σ_1 giving a total of $z\langle a_{-x} \rangle + \langle a_{-x+1} \rangle$ spikes in each of these neurons at time $t_{\log_2(z)+2}$. Proceeding to time $t_{2\log_2(z)+2}$ neurons σ_2, σ_3 and σ_4 each have $z^2\langle a_{-x} \rangle + z\langle a_{-x+1} \rangle + \langle a_{-x+2} \rangle$ spikes. This process continues until $X = \sum_{i=1}^{x} z^i \langle a_{-i} \rangle$ is computed. The end of the process is signaled when the rightmost number in the sequence is read. When this number (2) is read it allows the result to be passed to σ_6 via σ_5. Following this, σ_6 sends X spikes out of the system. Note that prior to this 2 being read only forgetting rules are executed in σ_6, thus preventing any spikes from being sent out of the system. Π_{input} computes X in time $x\log_2(z) + 3$. Recall from Section 5.1 that the value of z is dependent on the number of states and symbols in M, thus X is computed in time that is linear in x. In a similar manner, the value Y is computed by Π_{input} in time linear in y. The number $\langle q_r \rangle + \langle \alpha_i \rangle$ is computed in constant time. Thus the input $\langle C_k \rangle$ for Π_M is computed in linear time.

5.3 Algorithm overview

To help simplify the explanation, some of the rules given here differ slightly from those in the more detailed simulation that follows this overview. The numbers from Equation (2), encoding a Turing machine configuration, are stored in the neurons of our system as X, Y and $\langle q_r \rangle + \langle \alpha_i \rangle$ spikes. Equation (3) is implemented in Figure 3 to give an SN P system Π_M that simulates the transition rules of M. The two values X and Y are stored in neurons σ_1 and σ_2, respectively. If X or Y is to be multiplied the spikes that encode X or Y are sent down through the network of neurons from either σ_1 or σ_2 respectively, until they reach σ_{10}. Note in Figure 3 that each neuron from σ_7, σ_8 and σ_9 has incoming synapses coming from the other two neurons in σ_7, σ_8 and σ_9. Thus if σ_7, σ_8 and σ_9 each contain N spikes at time t_k, and they each fire sending N spikes, then each of the neurons σ_7, σ_8 and σ_9 will contain $2N$ spikes at time t_{k+1}. Given Y, the value $zY = 2^v Y$ is computed as follows: First we calculate $2Y$ by firing σ_7, σ_8 and σ_9, then $4Y$ by firing σ_7, σ_8, and σ_9 again. After v timesteps the value zY is computed. zX is computed using the same technique.

Now, we give the general idea of how the neurons compute $\frac{X}{z} - (\frac{X}{z} \mod z)$ and $(\frac{X}{z} \mod z)$ from Equation (3) (a slightly different strategy is used in the simulation). We begin with X spikes in σ_1. The rule $(s^z)^*/s^z \to s; 0$ is applied in σ_1 sending $\frac{X}{z}$ spikes to σ_4. Following this, $(s^z)^* s^{(\frac{X}{z} \mod z)}/s^z \to s^z; 0$ is applied

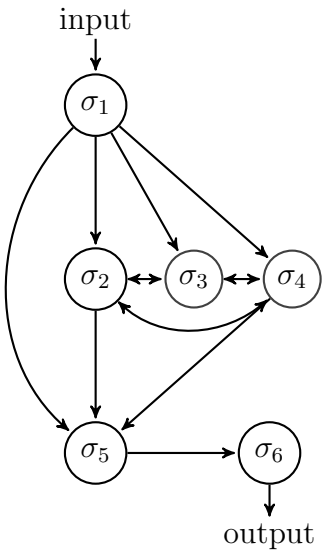

Fig. 2. Spiking neural P system Π_{input}. Each circle is a neuron and each arrow represents the direction spikes move along a synapse between a pair of neurons. The rules for Π_{input} are to be found in Table 4.

in σ_4 which sends $\frac{X}{z} - (\frac{X}{z} \mod z)$ to σ_1 leaving $(\frac{X}{z} \mod z)$ spikes in σ_4. The values $\frac{Y}{z} - (\frac{Y}{z} \mod z)$ and $(\frac{Y}{z} \mod z)$ are computed in a similar manner.

Finally, using the encoded current state $\langle q_r \rangle$ and the encoded read symbol $\langle \alpha_i \rangle$ the values $z\langle \alpha_j \rangle$ and $\langle q_u \rangle$ from Equation (3) are computed. Using the technique outlined in the first paragraph of the algorithm overview the value $z(\langle q_r \rangle + \langle \alpha_i \rangle)$ is computed by sending $\langle q_r \rangle + \langle \alpha_i \rangle$ spikes from σ_5 to σ_{10} in Figure 3. Then the rule $s^{z(\langle q_r \rangle + \langle \alpha_i \rangle)} / s^{z(\langle q_r \rangle + \langle \alpha_i \rangle) - \langle q_u \rangle} \rightarrow s^{z\langle \alpha_j \rangle}; 0$ is applied in σ_{10} which sends $z\langle \alpha_j \rangle$ spikes out to neurons σ_4 and σ_6. This rule uses $z(\langle q_r \rangle + \langle \alpha_i \rangle) - \langle q_u \rangle$ spikes thus leaving $\langle q_u \rangle$ spikes remaining in σ_{10}. This completes our sketch of how Π_M in Figure 3 computes the values in Equation (3) to simulate a transition rule. A more detailed simulation of a transition rule follows.

Before beginning this more detailed simulation it may help the reader to note the following: The encoding of the tape contents X and Y are multiples of z, whereas the combined encoding $\langle q_r \rangle + \langle \alpha_i \rangle$ of the current state and read symbol is strictly less than z. This fact allows us to combine the encoding $\langle q_r \rangle + \langle \alpha_i \rangle$ with the encoding for the left side or right side of the tape, while at the same time allowing us to distinguish the value of the encoded current state and read symbol. For example, given $X + \langle q_r \rangle + \langle \alpha_i \rangle$ or $Y + \langle q_r \rangle + \langle \alpha_i \rangle$ the applicability of a rule of the form $(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle} / s^b \rightarrow s^p; d$ is dependent only on the value of the current state and read symbol. Finally, it may also be helpful to notice that it is a fairly straightforward matter to determine from Tables 2 and 3 that it is

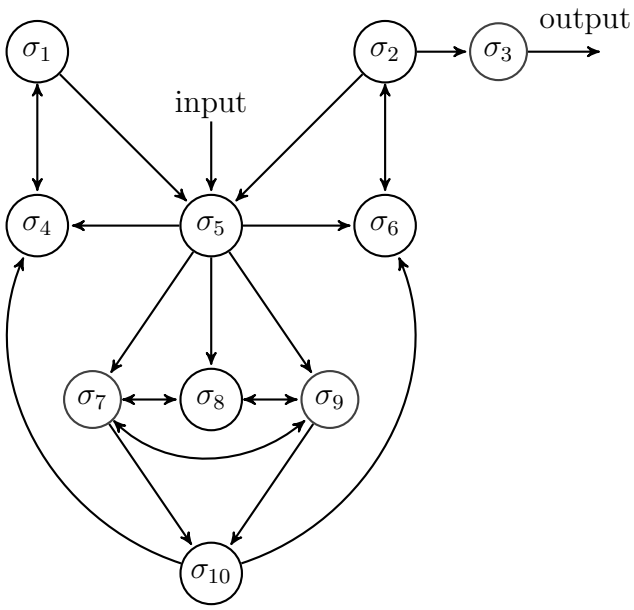

Fig. 3. Universal SN P system Π_M. Each circle is a neuron and each arrow represents the direction spikes move along a synapse between a pair of neurons. The rules for Π_M are to be found in Tables 2 and 3.

only possible to have one rule applicable in a neuron at a given timestep, thus ensuring that the operation of our system is deterministic.

5.4 Simulation of $q_r, \alpha_i, \alpha_j, L, q_u$ (top case of Equation (3))

The simulation of the transition rule begins at time t_k with X spikes in σ_1, Y spikes in σ_2, $\langle q_r \rangle + \langle \alpha_i \rangle$ spikes in σ_4 and σ_6, and 1 spike in σ_{10}. As before we explain the simulation by giving the number of spikes in each neuron and the rule that is to be applied in each neuron at time t. So at time t_k we have

$$\begin{aligned} t_k : \sigma_1 &= X, \\ \sigma_2 &= Y, \\ \sigma_4, \sigma_6 &= \langle q_r \rangle + \langle \alpha_i \rangle, & s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s \to s; 0, \\ \sigma_{10} &= 1, & s/s \to s; \log_2(z) + 2. \end{aligned}$$

Thus from Figure 3 when we apply the rule $s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s \to s; 0$ in neurons σ_4 and σ_6 at time t_k we get

$$\begin{aligned} t_{k+1} : \sigma_1 &= X + \langle q_r \rangle + \langle \alpha_i \rangle, & s^{2z}(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z \to s; \log_2(z) + 5, \\ \sigma_2 &= Y + \langle q_r \rangle + \langle \alpha_i \rangle, & (s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s \to s; 0, \\ \sigma_{10} &= 1, & s/s \to s; \log_2(z) + 1. \end{aligned}$$

$$
\begin{aligned}
t_{k+2}: \sigma_1 &= X + \langle q_r \rangle + \langle \alpha_i \rangle, & s^{2z}(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z &\to s; \log_2(z) + 4,\\
\sigma_3 &= Y + \langle q_r \rangle + \langle \alpha_i \rangle, & & \\
&\quad \text{if } \langle q_r \rangle = \langle q_{|Q|} \rangle & (s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z &\to s^z; 0,\\
&\quad \text{if } \langle q_r \rangle \neq \langle q_{|Q|} \rangle & (s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to \lambda,\\
\sigma_5 &= Y + \langle q_r \rangle + \langle \alpha_i \rangle, & (s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to s; 0,\\
\sigma_6 &= Y + \langle q_r \rangle + \langle \alpha_i \rangle, & s^z(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to \lambda,\\
\sigma_{10} &= 1, & s/s &\to s; \log_2(z).
\end{aligned}
$$

$$
\begin{aligned}
t_{k+3}: \sigma_1 &= X + \langle q_r \rangle + \langle \alpha_i \rangle, & s^{2z}(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z &\to s; \log_2(z) + 3,\\
\sigma_4, \sigma_6 &= Y + \langle q_r \rangle + \langle \alpha_i \rangle, & s^z(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to \lambda,\\
\sigma_7, \sigma_8, \sigma_9 &= Y + \langle q_r \rangle + \langle \alpha_i \rangle, & s^z(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s &\to s; 0,\\
\sigma_{10} &= 1, & s/s &\to s; \log_2(z) - 1.
\end{aligned}
$$

In timestep t_{k+2} above σ_3 (the output neuron) fires iff the current state q_r is the halt state $q_{|Q|}$. Recall that when M halts the entire tape contents are to the right of the tape head, thus only Y (the encoding of the right sequence) is sent out of the system. Thus the unary output is a number of spikes that encodes the tape contents of M.

Note that at timestep t_{k+3} neuron σ_7 receives $Y + \langle q_r \rangle + \langle \alpha_i \rangle$ spikes from each of the two neurons σ_8 and σ_9. Thus at time t_{k+4} neuron σ_7 contains $2(Y + \langle q_r \rangle + \langle \alpha_i \rangle)$ spikes. In a similar manner σ_8 and σ_9 also receive $2(Y + \langle q_r \rangle + \langle \alpha_i \rangle)$ spikes at timestep t_{k+3}. The number of spikes in each of the neurons σ_7, σ_8 and σ_9 doubles at each timestep between t_{k+3} and $t_{k+\log_2(z)+2}$.

$$
\begin{aligned}
t_{k+4}: \sigma_1 &= X + \langle q_r \rangle + \langle \alpha_i \rangle, & s^{2z}(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z &\to s; \log_2(z) + 2,\\
\sigma_7, \sigma_8, \sigma_9 &= 2(Y + \langle q_r \rangle + \langle \alpha_i \rangle), & s^z(s^z)^* s^{2(\langle q_r \rangle + \langle \alpha_i \rangle)}/s &\to s; 0,\\
\sigma_{10} &= 1, & s/s &\to s; \log_2(z) - 2.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+5}: \sigma_1 &= X + \langle q_r \rangle + \langle \alpha_i \rangle, & s^{2z}(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z &\to s; \log_2(z) + 1,\\
\sigma_7, \sigma_8, \sigma_9 &= 4(Y + \langle q_r \rangle + \langle \alpha_i \rangle), & s^z(s^z)^* s^{4(\langle q_r \rangle + \langle \alpha_i \rangle)}/s &\to s; 0,\\
\sigma_{10} &= 1, & s/s &\to s; \log_2(z) - 3.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+6}: \sigma_1 &= X + \langle q_r \rangle + \langle \alpha_i \rangle, & s^{2z}(s^z)^* s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s^z &\to s; \log_2(z),\\
\sigma_7, \sigma_8, \sigma_9 &= 8(Y + \langle q_r \rangle + \langle \alpha_i \rangle), & s^z(s^z)^* s^{8(\langle q_r \rangle + \langle \alpha_i \rangle)}/s &\to s; 0,\\
\sigma_{10} &= 1, & s/s &\to s; \log_2(z) - 4.
\end{aligned}
$$

The number of spikes in neurons σ_7, σ_8, and σ_9 continues to double until timestep $t_{k+\log_2(z)+2}$. When neurons σ_7 and σ_9 fire at timestep $t_{k+\log_2(z)+2}$ they each send

$\frac{z}{2}(Y + \langle q_r\rangle + \langle \alpha_i\rangle)$ spikes to neuron σ_{10}, which has opened at time $t_{k+\log_2(z)+2}$ (for the first time in the transition rule simulation). Thus at time $t_{k+\log_2(z)+3}$ neuron σ_{10} contains $z(Y + \langle q_r\rangle + \langle \alpha_i\rangle)$ spikes.

$$
\begin{aligned}
t_{k+\log_2(z)+2} : \sigma_1 &= X + \langle q_r\rangle + \langle \alpha_i\rangle, && s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s^z \to s; 4,\\
\sigma_7, \sigma_8, \sigma_9 &= \frac{z}{2}(Y + \langle q_r\rangle + \langle \alpha_i\rangle), && s^z(s^z)^* s^{\frac{z}{2}(\langle q_r\rangle+\langle \alpha_i\rangle)}/s \to s; 0,\\
\sigma_{10} &= 1, && s/s \to s; 0.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+\log_2(z)+3} : \sigma_1 &= X + \langle q_r\rangle + \langle \alpha_i\rangle, && s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s^z \to s; 3,\\
\sigma_4, \sigma_6 &= 1, && s/s \to \lambda,\\
\sigma_7, \sigma_8, \sigma_9 &= z(Y + \langle q_r\rangle + \langle \alpha_i\rangle), && (s^z)^*/s \to \lambda,\\
\sigma_{10} &= z(Y + \langle q_r\rangle + \langle \alpha_i\rangle), && (s^{z^2})^* s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}/s^{z^2} \to s^{z^2}; 0.
\end{aligned}
$$

Note that $(zY \mod z^2) = 0$ and also that $z(\langle q_r\rangle + \langle \alpha_i\rangle) < z^2$. Thus in neuron σ_{10} at time $t_{k+\log_2(z)+3}$ the rule $(s^{z^2})^* s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}/s^{z^2} \to s^{z^2}; 0$ separates the encoding of the right side of the tape s^{zY} and the encoding of the current state and read symbol $s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}$. To see this, note that the number of spikes in neurons σ_6 and σ_{10} at time $t_{k+\log_2(z)+4}$.

The rule $s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}/s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)-\langle q_u\rangle-1} \to s^{z\langle \alpha_j\rangle}; 0$, applied in neuron σ_{10} at timestep $t_{k+\log_2(z)+4}$, computes the new encoded current state $\langle q_u\rangle$ and the encoded write symbol $z\langle \alpha_j\rangle$. To see this note the number of spikes in neurons σ_6 and σ_{10} at time $t_{k+\log_2(z)+5}$. Note that neuron σ_1 is preparing to execute the rule $s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s^z \to s; 0$ at timestep $t_{k+\log_2(z)+6}$, and so at timesteps $t_{k+\log_2(z)+4}$ and $t_{k+\log_2(z)+5}$ neuron σ_1 remains closed. Thus the spikes sent out from σ_4 at these times do not enter σ_1.

$$
\begin{aligned}
t_{k+\log_2(z)+4} : \sigma_1 &= X + \langle q_r\rangle + \langle \alpha_i\rangle, && s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s^z \to s; 2,\\
\sigma_4, \sigma_6 &= zY, && (s^z)^*/s \to s; 0,\\
\sigma_{10} &= z(\langle q_r\rangle + \langle \alpha_i\rangle), && s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}/s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)-\langle q_u\rangle-1} \to s^{z\langle \alpha_j\rangle}; 0.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+\log_2(z)+5} : \sigma_1 &= X + \langle q_r\rangle + \langle \alpha_i\rangle, && s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s^z \to s; 1,\\
\sigma_2 &= zY, &&\\
\sigma_4, \sigma_6 &= z\langle \alpha_j\rangle, && (s^z)^*/s \to s; 0,\\
\sigma_{10} &= \langle q_u\rangle + 1, && s^{\langle q_u\rangle+1}/s^{\langle q_u\rangle} \to s^{\langle q_u\rangle}; 3.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+\log_2(z)+6} : \sigma_1 &= X + \langle q_r\rangle + \langle \alpha_i\rangle, && s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s^z \to s; 0,\\
\sigma_2 &= zY + z\langle \alpha_j\rangle, &&\\
\sigma_{10} &= \langle q_u\rangle + 1, && s^{\langle q_u\rangle+1}/s^{\langle q_u\rangle} \to s^{\langle q_u\rangle}; 2.
\end{aligned}
$$

At time $t_{k+\log_2(z)+7}$ in neuron σ_4 the rule $s^z(s^z)^*s^{(\frac{X}{z} \mod z)}/s^z \to s^z; 0$ is applied sending $\frac{X}{z} - (\frac{X}{z} \mod z)$ spikes to σ_1 and leaving $(\frac{X}{z} \mod z)$ spikes in σ_4. At the same time in neuron σ_5 the rule $s^z(s^z)^*s^{(\frac{X}{z} \mod z)}/s^z \to \lambda$ is applied leaving only $(\frac{X}{z} \mod z)$ spikes in σ_5.

$$
\begin{aligned}
t_{k+\log_2(z)+7} : \sigma_1 &= \langle q_r \rangle + \langle \alpha_i \rangle, & s^{\langle q_r \rangle + \langle \alpha_i \rangle}/s \to \lambda,\\
\sigma_2 &= zY + z\langle \alpha_j \rangle, & \\
\sigma_4 &= \frac{X}{z}, & s^z(s^z)^*s^{(\frac{X}{z} \mod z)}/s^z \to s^z; 0,\\
\sigma_5 &= \frac{X}{z}, & s^z(s^z)^*s^{(\frac{X}{z} \mod z)}/s^z \to \lambda,\\
\sigma_{10} &= \langle q_u \rangle + 1, & s^{\langle q_u \rangle + 1}/s^{\langle q_u \rangle} \to s^{\langle q_u \rangle}; 1.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+\log_2(z)+8} : \sigma_1 &= \frac{X}{z} - (\frac{X}{z} \mod z), & \\
\sigma_2 &= zY + z\langle \alpha_j \rangle, & \\
\sigma_4 &= \frac{X}{z} \mod z, & s^{(\frac{X}{z} \mod z)}/s \to \lambda,\\
\sigma_5 &= \frac{X}{z} \mod z, & s^{(\frac{X}{z} \mod z)}/s \to s; 0,\\
\sigma_{10} &= \langle q_u \rangle + 1, & s^{\langle q_u \rangle + 1}/s^{\langle q_u \rangle} \to s^{\langle q_u \rangle}; 0.
\end{aligned}
$$

$$
\begin{aligned}
t_{k+\log_2(z)+9} : \sigma_1 &= \frac{X}{z} - (\frac{X}{z} \mod z), & \\
\sigma_2 &= zY + z\langle \alpha_j \rangle, & \\
\sigma_4 &= \langle q_u \rangle + (\frac{X}{z} \mod z), & s^{\langle q_u \rangle + (\frac{X}{z} \mod z)}/s \to s; 0,\\
\sigma_6 &= \langle q_u \rangle + (\frac{X}{z} \mod z), & s^{\langle q_u \rangle + (\frac{X}{z} \mod z)}/s \to s; 0,\\
\sigma_7, \sigma_8, \sigma_9 &= \frac{X}{z} \mod z, & s^{(\frac{X}{z} \mod z)}/s^{(\frac{X}{z} \mod z)} \to \lambda,\\
\sigma_{10} &= 1, & s/s \to s; \log_2(z) + 2.
\end{aligned}
$$

The simulation of the left moving transition rule is now complete. Note that the number of spikes in σ_1, σ_2, σ_4, and σ_6 at timestep $t_{k+\log_2(z)+9}$ are the values given by the top case of Equation (3) and encode the configuration after the left move transition rule.

The case of when the tape head moves onto a part of the tape that is to the left of a_{-x+1} in Equation (1) is not covered by the simulation. For example when the tape head is over cell a_{-x+1}, then $X = z$ (recall a_{-x} contains α_1). If the tape head moves to the left, then from the top case of Equation (3) the new value for the left sequence is $X = 0$. However, we must maintain the value of $X = z$ to simulate the infinite blank symbols (α_1 symbols) to the left of the tape. This

is achieved as follows: The rule $s^{z+\langle q_r \rangle + \langle \alpha_i \rangle}/s^z \to s^z; 0$ is applied in σ_1 at time $t_{k+\log_2(z)+6}$. Then at time $t_{k+\log_2(z)+7}$ the rule $(s^z)^*/s \to s; 0$ is applied in σ_4 and the rule $s^z/s^{z-1} \to \lambda$ is applied in σ_5. Thus at time $t_{k+\log_2(z)+8}$ there are z spikes in σ_1 which simulates another α_1 symbol to the left, and there is 1 spike in σ_5 to simulate the current read symbol α_1.

We have shown how to simulate an arbitrary left moving transition rule of M. Right moving transition rules are also simulated in $\log_2(z) + 9$ timesteps in a manner similar to that of left moving transition rules. Thus a single transition rule of M is simulated by Π_M in $\log_2(z) + 9$ timesteps. Recall from Section 5.1 $z = 2^{\log_2 \lceil 2|Q||A|+2|A| \rceil}$ and note that $2^{\log_2 \lceil 2|Q||A|+2|A| \rceil} \leqslant 8|Q||A|$, thus the entire computation of M is simulated in $O(T \log(|Q||A|))$ time. From Section 5.1 M is simulated in $O([|Q||A|]^T)$ space. □

6 Conclusions

While the small universal SN P system in Figure 3 simulates Turing machines with a linear time overhead it *requires* an exponential space overhead. This is because of the unary encoding used by SN P systems.

It was mentioned in Section 2 that we generalised the previous definition of SN P system with exhaustive use of rules; no bound is placed on the number of spikes that can be received from the environment by the input neuron in a single timestep. We can give a universal SN P system with exhaustive use of rules that simulates Turing machines in linear time and receives at most one spike from the environment at each timestep. The input to such a system would be a spike train $\{0, 1\}^*$ that gives a binary encoding of the Turing machine tape contents. The system C_Π given in Figure 3 can be modified to take such input. To convert a binary input encoding to a unary encoding suitable for use by the system in Figure 3 would involve repeated multiplication of the input by 2 as it is read in (similar to the technique used by the system in Figure 2). C_Π already has a set of neurons (σ_7, σ_8, σ_9) that implements multiplication by 2, and thus our system in Figure 3 would only need a small increase in the number of neurons and some additional rules to take binary input. Note that if we impose the further restriction of allowing only a constant number of spikes to enter the input neuron from the environment, as is the case with many other universal SN P systems, then the system will remain exponentially slow. This is because such input sequences necessitate unary encodings that result in exponentially slow communication with the environment. By restricting our system to receive only a single spike from the environment at each timestep means that the only efficient way to send input into the system is with a binary input encoding. This results in passing binary input into a system that in essence works on unary encodings; the initial part of the computation would be concerned with converting the binary input data into a from that the system can manipulate in an effective manner. If there is no bound placed on the number of spikes that can be transmitted by a synapses in a single timestep, then placing no bound on the number of spikes that may enter the input neuron from the environment in

a single timestep seems like a natural generalisation. This allows a unary input encoding to be read into the system in an efficient manner. Note that without this generalisation communication between the environment is more restricted than communications between neurons within the system. This is not the case for the original SN P system model where rules are applied in a non-exhaustive way.

Another issue when giving a simulation with SN P systems with exhaustive use of rules, is the choice of model. SN P systems with exhaustive use of rules can multiply any natural number by a positive rational constant in a single timestep. Addition and subtraction, on the other hand, would appear to be more cumbersome to simulate with exhaustive use of rules. The first universal SN P systems with exhaustive use of rules simulated counter machines [4, 18]. These systems used encodings of the form 3^n to encoded the value n stored in a counter so that addition and subtraction could be simulated using multiplication by 3 and $\frac{1}{3}$, respectively. Such an encoding uses space that is doubly exponential when compared with a binary encoding of n. Recall that moving left or right on a Turing machine tape may be simulated by division or multiplication (for example see Equation 3). Thus, when one wishes to simulate a sequential model using SN P systems with exhaustive use of rules, the Turing machine seems like a good choice.

References

1. H. Chen, M. Ionescu, and T. Ishdorj. On the efficiency of spiking neural P systems. In M. A. Gutiérrez-Naranjo, G. Păun, A. Riscos-Núñez, and F. J. Romero-Campero, editors, *Proceedings of Fourth Brainstorming Week on Membrane Computing*, pages 195–206, Sevilla, Spain, Feb. 2006.
2. P. C. Fischer, A. Meyer, and A. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2(3):265–283, 1968.
3. M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.
4. M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems with exhaustive use of rules. *International Journal of Unconventional Computing*, 3(2):135–153, 2007.
5. M. Ionescu and D. Sburlan. Some applications of spiking neural P systems. In G. Eleftherakis, P. Kefalas, and G. Păun, editors, *Proceedings of the Eighth Workshop on Membrane Computing*, pages 383–394, Thessaloniki, Greece, June 2007.
6. I. Korec. Small universal register machines. *Theoretical Computer Science*, 168(2):267–301, 1996.
7. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. Solving numerical NP-complete problems with spiking neural P systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *8th International Workshop, WMC 2007*, volume 4860 of *LNCS*, pages 336–352, Thessaloniki, Greece, June 2007.
8. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, 5(5):459–473, 2009.

9. T. Neary. Presentation at The International Workshop on Computing with Biomolecules (CBM 2008). Available at http://www.emcc.at/UC2008/Presentations/CBM5.pdf .
10. T. Neary. On the computational complexity of spiking neural P systems. In C. S. Calude, J. Félix, Costa, R. Freund, M. Oswald, and G. Rozenberg, editors, *Unconventional Computation, 7th International Conference, UC 2008*, volume 5204 of *LNCS*, pages 189–205, Vienna, Austria, Aug. 2008. Springer.
11. T. Neary. A small universal spiking neural P system. In E. Csuhaj-Varjú, R. Freund, M. Oswald, and K. Salomaa, editors, *International Workshop on Computing with Biomolecules*, pages 65–74, Vienna, Austria, Aug. 2008. Austrian Computer Society.
12. T. Neary. A boundary between universality and non-universality in spiking neural P systems. *arXiv:0912.0741v3 [cs.CC]*, Dec. 2009.
13. T. Neary. A boundary between universality and non-universality in spiking neural P systems. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *4th International Conference on Language and Automata Theory and Applications, LATA 2010*, volume 6031 of *LNCS*, pages 475–487, Trier, Germany, May 2010. Springer.
14. T. Neary. A universal spiking neural P system with 11 neurons. In *Eleventh International Conference on Membrane Computing (CMC11)*, Jena, Germany, Aug. 2010.
15. A. Păun and G. Păun. Small universal spiking neural P systems. *BioSystems*, 90(1):48–60, 2007.
16. G. Păun. *Membrane Computing: An Introduction*. Springer, 2002.
17. R. Schroeppel. A two counter machine cannot calculate 2^n. Technical Report AIM-257, A.I. memo 257, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1972.
18. X. Zhang, Y. Jiang, and L. Pan. Small universal spiking neural P systems with exhaustive use of rules. In D. Kearney, V. Nguyen, G. Gioiosa, and T. Hendtlass, editors, *3rd International Conference on Bio-Inspired Computing: Theories and Applications(BICTA 2008)*, pages 117–128, Adelaide, Australia, Oct. 2008. IEEE.
19. X. Zhang, X. Zeng, and L. Pan. Smaller universal spiking neural P systems. *Fundamenta Informaticae*, 87(1):117–136, 2008.

neuron	rules
σ_1	$(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to s;0$ if D=R
	$s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s^z \to s;\log_2(z)+5$ if D=L
	$s^{z+\langle q_r\rangle+\langle\alpha_i\rangle}/s^z \to s^z;\log_2(z)+5$ if D=L
	$s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to \lambda$ if D=L
σ_2	$(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to s;0$ if D=L or $\langle q_r\rangle = \langle q_{\lvert Q\rvert}\rangle$
	$s^{2z}(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s^z \to s;\log_2(z)+5$ if D=R
	$s^{z+\langle q_r\rangle+\langle\alpha_i\rangle}/s^z \to s^z;\log_2(z)+5$ if D=R
	$s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to \lambda$ if D=R
σ_3	$(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s^z \to s^z;0,$ if $\langle q_r\rangle = \langle q_{\lvert Q\rvert}\rangle$
	$(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to \lambda,$ if $\langle q_r\rangle \neq \langle q_{\lvert Q\rvert}\rangle$
	$(s^z)^* s^{(\frac{X}{z} \mod z)}/s \to \lambda$
	$s^z/s \to \lambda$
σ_4	$s^2(s^z)^*/s^z \to s^z;1$
	$(s^z)^*/s \to s;0$
	$s^2/s^2 \to \lambda$
	$s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to s;0$
	$s^z(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to \lambda$
	$s/s \to \lambda$
	$s^z(s^z)^* s^{(\frac{X}{z} \mod z)}/s^z \to s^z;0$
	$s^{(\frac{X}{z} \mod z)}/s \to \lambda$
σ_5	$s^2(s^z)^*/s \to s;0$
	$s^{2z}(s^z)^*/s \to s;0$
	$(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to s;0$
	$s^z(s^z)^* s^{(\frac{X}{z} \mod z)}/s^z \to \lambda$
	$s^z/s^{z-1} \to \lambda$
	$s^{(\frac{X}{z} \mod z)}/s \to s;0$
σ_6	$s^2(s^z)^*/s \to \lambda$
	$(s^z)^*/s \to s;0$
	$s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to s;0$
	$s^z(s^z)^* s^{\langle q_r\rangle+\langle\alpha_i\rangle}/s \to \lambda$
	$s/s \to \lambda$
	$s^z(s^z)^* s^{(\frac{Y}{z} \mod z)}/s^z \to s^z;0$
	$s^{(\frac{Y}{z} \mod z)}/s \to \lambda$

Table 2. This table gives the rules in each of the neurons σ_1 to σ_6 of Π_M. In the rules above q_r is the current state, α_i is the read symbol, α_j is the write symbol, D is the move direction, and q_u is the next state of some transition rule $q_r, \alpha_i, \alpha_j, D, q_u$ of M. Note that $(\frac{X}{z} \mod z), (\frac{Y}{z} \mod z) \in \langle A\rangle$ the set of encodings for the symbols of M (see Section 5.1).

neuron	rules
$\sigma_7, \sigma_8, \sigma_9$	$s^2(s^z)^*/s \to \lambda$
	$(s^z)^*/s \to \lambda$
	$s^{\langle q_r\rangle+\langle \alpha_i\rangle}/s \to \lambda$
	$s^z(s^z)^* s^{\frac{z}{m}(\langle q_r\rangle+\langle \alpha_i\rangle)}/s \to s;0$ for all $m = 2^k$, $2 \leqslant m \leqslant z$ and $k \in \mathbb{N}$
	$s^{(\frac{X}{z} \mod z)}/s^{(\frac{X}{z} \mod z)} \to \lambda$
σ_{10}	$s^{31}/s^{16} \to \lambda$
	$s^{15}/s^8 \to \lambda$
	$s^7/s^4 \to \lambda$
	$s^3/s^2 \to \lambda$
	$s/s \to s; \log_2(z) + 2$
	$(s^{z^2})^* s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}/s^{z^2} \to s^{z^2};0$
	$s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)}/s^{z(\langle q_r\rangle+\langle \alpha_i\rangle)-\langle q_u\rangle-1} \to s^{z\langle \alpha_j\rangle};0$
	$s^{\langle q_u\rangle+1}/s^{\langle q_u\rangle} \to s^{\langle q_u\rangle};3$

Table 3. This table gives the rules in each of the neurons σ_7 to σ_{10} of Π_M. See caption of Table 2 for further explanation.

neuron	rules
σ_1	$s^*/s \to s;0$
$\sigma_2, \sigma_3, \sigma_4$	$s^*/s \to s;0$
σ_5	$(s^z)^* s^{\langle \alpha\rangle}/s \to s; \log_2(z) - 1$
	$(s^z)^* s^2/s \to s;0$
σ_6	$(s^z)^* s^{\langle \alpha\rangle}/s \to \lambda$
	$(s^z)^* s^2/s^z \to s^z;0$

Table 4. This table gives the rules in each of the neurons of Π_{input}.