

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 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 polynomial time and has only 18 neurons.

1 Introduction

Since their inception inside of the last decade P systems [12] have spawned a variety of hybrid systems. One such hybrid, that of spiking neural P system [3], results from a fusion with spiking neural networks. It has been shown that these systems are computationally universal. Here the time/space computational complexity of spiking neural P systems is examined. We begin by showing that counter machines simulate standard spiking neural 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 is no spiking neural P system that simulates Turing machines with less than exponential time and space overheads. These results are for spiking neural P systems that have a constant number of neurons independent of the input length.

Extended spiking neural P systems with exhaustive use of rules were proved computationally universal in [4]. However, the technique used to prove universality involved the simulation of counter machines and thus suffers from an exponential time overhead. In the second part of the paper we give an extended spiking neural P system with exhaustive use of rules that simulates Turing machines in polynomial time and has only 18 neurons. Previously, Păun and Păun [11] gave a small universal spiking neural P system with 84 neurons and another, that uses extended rules, with 49 neurons. Both of these spiking neural P systems

* The author would like to thank the anonymous reviewers for their careful reading and observations. The author is funded by Science Foundation Ireland Research Frontiers Programme grant number 07/RFP/CSMF641.

require exponential time and space to simulate Turing machines but do not have exhaustive use of rules.

Chen et al. [1] have shown that with exponential pre-computed resources SAT is solvable in constant time with spiking neural P systems. Leporati et al. [6] gave a semi-uniform family of extended spiking neural P systems that solve the SUBSET SUM problem in constant time. In later work, Leporati et al. [7] gave a uniform family of maximally parallel spiking neural P systems with more general rules that solve the SUBSET SUM problem in polynomial time. All the above solutions to NP-hard problems rely families of spiking neural P systems. Specifically, the size of the problem instance determines the number of neurons in the spiking neural 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 Dragos [5] have shown that spiking neural P systems simulate circuits in linear time.

In the next two sections we give definitions for spiking neural P systems and counter machines and explain the operation of both. Following this, in Section 4, we prove that counter machines simulate spiking neural P systems in linear time. Thus proving that there exists no universal spiking neural P systems that simulate Turing machines in less than exponential time. In Section 5 we present our universal spiking neural P systems that simulates Turing machine in polynomial time and has only 18 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 is a tuple $\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out)$, where:*

1. $O = \{s\}$ is the unary alphabet (s is known as a spike),
2. $\sigma_1, \sigma_2, \dots, \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
 - (a) $n_i \geq 0$ 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 , $b \geq 1$ and $d \geq 1$,
 - ii. $s^e \rightarrow \lambda; 0$ where λ is the empty word, $e \geq 1$, 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. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ are the set of synapses between neurons, where $i \neq j$ for all $(i, j) \in syn$,
4. $in, out \in \{\sigma_1, \sigma_2, \dots, \sigma_m\}$ are the input and output neurons respectively.

In the same manner as in [11], 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. 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 $j \geq b$ 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 $t + d - 1$ 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 $t + d - 1$ and no other rule may execute in σ_i until time $t + d$. We note here that in 2b(i) it is standard to have a $d \geq 0$. However, we have $d \geq 1$ as it simplifies explanations throughout the paper. This does not effect the operation as the neuron fires at time $t + d - 1$ instead of $t + d$. A forgetting rule $r' = s^e \rightarrow \lambda; 0$ 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 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 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. The output is the time between the first and second spike in the output neuron σ_m .

An extended spiking neural P system [11] has more general rules of the form $E/s^b \rightarrow s^p; d$, where $b \geq p \geq 0$. Note if $p = 0$ then $E/s^b \rightarrow s^p; d$ is a forgetting rule. An extended spiking neural P system with exhaustive use of rules [4] applies its rules as follows. If a neuron σ_i contains k spikes and the rule $E/s^b \rightarrow s^p; d$ is applicable, 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, a synapse in a spiking neural P system with exhaustive use of rules may transmit an arbitrary number of spikes in a single timestep. In the sequel we allow the input neuron of a system with exhaustive use of rules to receive an arbitrary number of spikes in a single timestep. This is a generalisation on the input allowed by Ionescu et al. [4].

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

3 Counter machines

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, $Q = \{q_0, q_1, \dots, q_h\}$ 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

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

where $g(i)$ is a binary valued function and $0 \leq i \leq z$, Y and N control the movement of the input read head, and INC , DEC , 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 .

- $f(\alpha, q_j, g(i)) = (Y, q_k, INC(h))$ 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.
- $f(\alpha, q_j, g(i)) = (N, q_k, DEC(h))$ 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 by 1.

Our definition for counter machine, given above, is more restricted than the definition given by Fischer [2]. In Fischer's definition INC and DEC 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 non-deterministic counter machines as they use the same unary storage method.

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

Theorem 2. *Let Π be a spiking neural P system with m neurons that completes its computation in time T and space S . Then there is a non-deterministic counter machine C_Π that simulates the operation of Π in time $O(T(x_r)^2m + Tm^2)$ and space $O(S)$ where x_r is a constant dependant on the rules of Π .*

Proof idea Before we give the proof of Theorem 2 we give the main idea behind the proof. Each neuron σ_i from the spiking neural 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 $\{\sigma_{i_1}, \dots, \sigma_{i_v}\}$ then for each open neuron σ_{i_j} in $\{\sigma_{i_1}, \dots, \sigma_{i_v}\}$ 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 $\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out)$ be a spiking neural P system where $in = \sigma_1$ and $out = \sigma_2$. We explain the operation of a non-deterministic counter machine C_Π that simulates the operation of Π in time $O(T(x_r)^2 m + Tm^2)$ and space $O(S)$.

There are $m + 1$ counters $c_1, c_2, c_3, \dots, c_m, c_{m+1}$ 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 . The states of the counter machine are used to control which neural rules are simulated in each counter and also to synchronise the operations of the simulated neurons (counters).

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 terminal 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 terminal at each simulated timestep. The counter c_1 (the simulated input neuron) is incremented only on timesteps when a 1 (a simulated spike) is read. As such at each simulated timestep t , a simulated spike is received by c_1 if and only if a spike is received by the input neuron σ_1 . 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.

Storing neural rules in the counter machine states Recall from Definition 1 that the applicability of a rule in a neuron is dependant on a regular expression over a unary alphabet. Let $r = E/s^b \rightarrow s; d$ be a rule in neuron σ_i . Then there is a finite state machine G that accepts language $L(E)$ and thus decides if the number of spikes in σ_i permits the application of r in σ_i at a given time in the computation. G is given in Figure 1. If g_j is an accept state in G then $j > b$. This ensures that there is enough spikes to execute r . We also place the restriction on G that $x > b$. During a computation we may use G to decide if r is applicable in σ_i by passing an s to G each time a spike enters σ_i . However, G may not give the correct result if spikes leave the neuron as it does not record

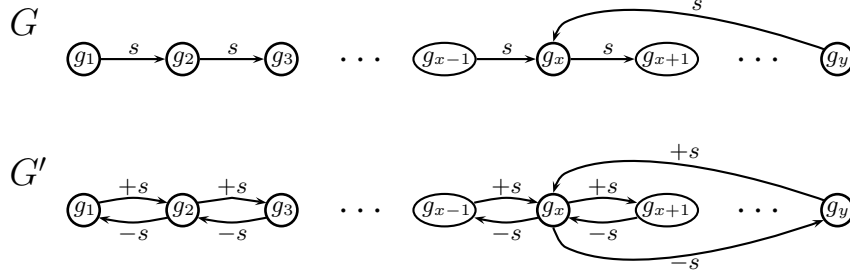


Fig. 1. Finite state machine G decides if a particular rule is applicable in a neuron given the number of spikes in the neuron *at a given time* in the computation. Each s represents a spike in the neuron. Machine G' keeps track of the movement of spikes into and out of the neuron and decides whether or not a particular 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.

spikes leaving σ_i . 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_i to g_j in G' that records the removal of a spike from G' .

By recording the dynamic movement of spikes, G' is able to decide if the number of spikes in σ_i permits the application of r in σ_i at each timestep during the computation. G' is also given in Figure 1. Note that forgetting rules $s^e \rightarrow \lambda; 0$ are dependent on simpler regular expressions thus we will not give a machine G' for forgetting rules here.

Let neuron σ_i have the greatest number l of rules of any neuron in Π . Thus the applicability of rules r_1, r_2, \dots, r_l in σ_i is decided by the automata G'_1, G'_2, \dots, G'_l . We record if a rule may be simulated in a neuron at any given timestep during the computation by recording the current state of its G' automaton (Figure 1) in the states of the counter machine. 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.

Recall that in each rule of the form $r = E/s^b \rightarrow s; d$ that d specifies the number of timesteps between the removal of b spikes from the neuron and the spiking of the neuron. The number of timesteps $< d$ remaining until a neuron will spike is recorded in the states of the C_Π . Each state in our counter machine remembers at most m different values $< d$.

Algorithm overview Next we explain the operation of C_{II} 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 y of simulated spikes are removed by decrementing the counter y times ($y = b$ or $y = e$ in 2b 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 synchronises each neuron with the global clock and increments the output counter if necessary. If the entire word w has not been read from the input tape the next symbol is read.

Stage 1. Identify rules to be simulated and remove spikes from neurons

Recall that $d = 0$ indicates a neuron is open and the value of d in each neuron is recorded in the states of the counter machine. Thus our algorithm begins by determining which rule to simulate in counter c_{i_1} where $i_1 = \min\{i \mid d = 0 \text{ for } \sigma_i\}$ and the current state of the counter machine encodes an accept state for one or more of the G' automata for the rules in σ_{i_1} at time t . If there is more than one rule applicable the counter machine non-deterministically chooses which rule to simulate. Let $r = E/s^b \rightarrow s; d$ be the rule that is to be simulated. Using the $DEC(i_1)$ instruction, counter c_{i_1} is decremented b times. With each decrement of c_{i_1} the new current state of each automaton G'_1, G'_2, \dots, G'_l is recorded in the counter machine's current state. After b decrements of c_i the simulation of the removal of b spikes from neuron σ_{i_1} is complete. Note that the value of d from rule r is recorded in the counter machine state.

There is a case not covered by the above paragraph. To see this note that in G' in Figure 1 there is a single non-deterministic choice to be made. This choice is at state g_x if a spike is being removed ($-s$). Thus, if one of the automata is in such a state g_x our counter machine resolves this by decrementing the counter x times using the DEC instruction. If $c_{i_1} = 0$ after the counter has been decremented x times then the counter machine simulates state g_{x-1} otherwise state g_y is simulated. Immediately after this the counter is incremented $x - 1$ times to restore it to the correct value.

When the simulation of the removal of b spikes from neuron σ_{i_1} is complete, the above process is repeated with counter c_{i_2} where $i_2 = \min\{i \mid i_2 > i_1, d = 0 \text{ for } \sigma_i\}$ and the current state of the counter machine encodes an accept state for one or more of the G' automata for the rules in σ_{i_2} at time t . 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 $i_1 = \min\{i \mid d = 1 \text{ for } \sigma_i\}$ (if $d = 1$ the neuron is firing). Let $\{(i_1, j_1), (i_1, j_2), \dots, (i_1, j_k)\}$ be

the set of synapses leaving σ_i where $j_u < j_{u+1}$ and $d \leq 1$ in σ_{j_u} at time t (if $d \leq 1$ the neuron is open and may receive spikes). Then the following sequence of instructions are executed $\text{INC}(j_1), \text{INC}(j_2), \dots, \text{INC}(j_k)$, thus incrementing any counter (simulated neuron) that receives a simulated spike.

The above process is repeated for synapses of the form (i_2, j) where $i_2 = \min\{i \mid i_2 > i_1, d = 1 \text{ for } \sigma_i\}$. 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 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 being read in by the input neuron. In this stage the state of the counter machine changes to record the fact that each $k \leq d$ that records the number of timesteps until a currently closed neuron will fire is decremented to $k - 1$. If the 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 output counter c_{m+1} . If during the simulation of timestep t counter c_m has simulated a spike 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 time between the first two spikes in σ_m .

Space analysis The input word on the binary tape of C_{II} is identical to the length of the binary sequence read in by the input neuron of II . Counters c_1 to c_m uses 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 II . Thus C_{II} simulates II in space of $O(S)$.

Time analysis The simulation involves 3 stages. Recall that $x > b$. Let x_r be the maximum value for x of any G' automaton thus x_r is greater than the maximum number of spikes deleted in a neuron.

Stage 1. In order to simulate the deletion of a single spike in the worst case the counter will have to be decremented x_r times and incremented $x_r - 1$ times as in the special case. This is repeated a maximum of $b < x_r$ times (where b is the number of spikes removed). Thus a single iteration of Stage 1 take $O(x_r^2)$ time. Stage 1 is iterated a maximum of m times per simulated timestep giving $O(x_r^2 m)$ time.

Stage 2. The maximum number of synapses leaving a neuron i is m . 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.

Thus a single timestep of II is simulated by C_{II} in $O((x_r)^2 m + m^2)$ time and T timesteps of II are simulated in linear time $O(T(x_r)^2 m + Tm^2)$ by C_{II} . \square

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

Corollary 1. *There exist no universal spiking neural P system that simulates Turing machines with less than exponential time and space overheads.*

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

In this section we construct a universal spiking neural P system that allows exhaustive use of rules, has only 18 neurons, and simulates Turing machines in polynomial time. The system constructed efficiently simulates the computation of an existing small universal Turing machine [9]. This universal machine has 6 states and 4 symbols and is called $U_{6,4}$. The following theorem gives the time/space simulation overheads for $U_{6,4}$.

Theorem 3 ([9]). *Let M be a single tape Turing machine that runs in time T . Then $U_{6,4}$ simulates the computation of M in time $O(T^6)$ and space $O(T^3)$.*

This result is used in the proof of our main theorem which is as follows.

Theorem 4. *Let M be a single tape Turing machine that runs in time T . Then there is a universal spiking neural P system $\Pi_{U_{6,4}}$ with exhaustive use of rules that simulates the computation of M in time $O(T^6)$ and space $O(32T^3)$ and has only 18 neurons.*

If the reader would like to get a quick idea of how our spiking neural P system with 18 neurons operates they should skip to the algorithm overview subsection in the proof below.

Proof. We give a spiking neural P system $\Pi_{U_{6,4}}$ that simulates the universal Turing machine $U_{6,4}$ in linear time and exponential space. The algorithm given for $\Pi_{U_{6,4}}$ is deterministic and is mainly concerned with the simulation of an arbitrary transition rule for any Turing machine with the same state-symbol product as $U_{6,4}$, providing it has the same halting condition. Thus it is not necessary to give a detailed explanation of the operation of $U_{6,4}$. Any details about $U_{6,4}$ will be given where necessary.

Encoding a configuration of universal Turing machine $U_{6,4}$ Each unique configuration of $U_{6,4}$ is encoded as three natural numbers using a well known technique. A configuration of $U_{6,4}$ is given by the following equation

$$C_k = \mathbf{u}_r, \dots ccc a_{-x} \dots a_{-3} a_{-2} a_{-1} \underline{a_0} a_1 a_2 a_3 \dots a_y ccc \dots \quad (1)$$

where u_r is the current state, c is the blank symbol, each a_i is a tape cell of $U_{6,4}$ and the tape head of $U_{6,4}$, given by an underline, is over a_0 . Also, tape cells a_{-x} and a_y both contain c , and the cells between a_{-x} and a_y include all of

the cells on $U_{6,4}$'s tape that have either been visited by the tape head prior to configuration C_k or contain part of the input to $U_{6,4}$.

The tape symbols of $U_{6,4}$ are $c, \delta, b,$ and g and are encoded as $\langle c \rangle = 1, \langle \delta \rangle = 2, \langle b \rangle = 3,$ and $\langle g \rangle = 4,$ where the encoding of object x is given by $\langle x \rangle$. 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 $U_{6,4}$. We encode the tape contents in Equation (1) to the left and right of the tape head as the numbers $X = \sum_{i=1}^x 32^i \langle a_i \rangle$ and $Y = \sum_{j=1}^y 32^j \langle a_j \rangle,$ respectively. The states of $U_{6,4}$ are $u_1, u_2, u_3, u_4, u_5,$ and u_6 and are encoded as $\langle u_1 \rangle = 5, \langle u_2 \rangle = 9, \langle u_3 \rangle = 13, \langle u_4 \rangle = 17, \langle u_5 \rangle = 21$ and $\langle u_6 \rangle = 25.$ Thus the entire configuration C_k is encoded as three natural numbers via the equation

$$\langle C_k \rangle = (X, Y, \langle u_r \rangle + \langle \alpha_1 \rangle) \quad (2)$$

where $\langle C_k \rangle$ is the encoding of C_k from Equation (1) and α_1 is the symbol being read by the tape head in cell a_0 .

A transition rule $u_r, \alpha_1, \alpha_2, D, u_s$ of $U_{6,4}$ is executed on C_k as follows. If the current state is u_r and the tape head is reading the symbol α_1 in cell a_0, α_2 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 u_s becomes the new current state. A simulation of transition rule $u_r, \alpha_1, \alpha_2, D, u_s$ on the encoded configuration $\langle C_k \rangle$ from Equation (2) is given by the equation

$$\langle C_{k+1} \rangle = \begin{cases} (\frac{X}{32} - (\frac{X}{32} \bmod 32), 32Y + 32\langle \alpha_2 \rangle, (\frac{X}{32} \bmod 32) + \langle u_s \rangle) \\ (32X + 32\langle \alpha_2 \rangle, \frac{Y}{32} - (\frac{Y}{32} \bmod 32), (\frac{Y}{32} \bmod 32) + \langle u_s \rangle) \end{cases} \quad (3)$$

where configuration C_{k+1} results from executing a single transition rule on configuration $C_k,$ and $(b \bmod c) = d$ 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 sequence. Cell a_0 is overwritten with the write symbol α_2 and thus we compute $32Y + 32\langle \alpha_2 \rangle$ 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 $\frac{X}{32} - (\frac{X}{32} \bmod 32).$ The rightmost cell of the left sequence a_{-1} is the *new* tape head location and the tape symbol it contains is encoded as $(\frac{X}{32} \bmod 32).$ Thus the value $(\frac{X}{32} \bmod 32)$ is added to the new encoded current state $\langle u_s \rangle.$ For the bottom case, a right move, the sequence to the right gets shorter which is simulated by $\frac{Y}{32} - (\frac{Y}{32} \bmod 32)$ and the sequence to the left gets longer which is simulated by $32X + 32\langle \alpha_2 \rangle.$ The leftmost cell of the right sequence a_1 is the new tape head location and the tape symbol it contains is encoded as $(\frac{Y}{32} \bmod 32).$

Input to $\Pi_{U_{6,4}}$ Here we give an explanation of how the input is read into $\Pi_{U_{6,4}}.$ We also give an rough outline of how the input to $\Pi_{U_{6,4}}$ is encoded in linear time.

A configuration C_k given by Equation (2) is read into $\Pi_{U_{6,4}}$ as follows. All the neurons of the system initially have no spikes with the exception of σ_3 , which has 30 spikes. The input neuron σ_1 receives X spikes at the first timestep t_1 , Y spikes at time t_2 , and $\langle \alpha_1 \rangle + \langle u_r \rangle$ spikes at time t_3 . Using the rule $s^*/s \rightarrow s; 1$ the neuron σ_1 sends all the spikes it receives during timestep t_i to σ_6 at timestep t_{i+1} . Thus using the rules $(s^{64}(s^{32})^*/s \rightarrow s; 1)$ and $(s^{\langle \alpha_1 \rangle + \langle u_r \rangle}/s \rightarrow s; 1)$ in σ_6 , the rule $(s^{64}(s^{32})^*/s \rightarrow s; 2)$ in σ_5 , the rule $(s^{64}(s^{32})^*/s \rightarrow s; 1)$ in σ_7 , and the rule $s^{30}/s^{30} \rightarrow \lambda; 5$ in σ_3 , the spiking neural P system has X spikes in σ_2 , Y spikes in σ_3 , and $\langle \alpha_1 \rangle + \langle u \rangle$ spikes in σ_5 and σ_7 at time t_6 . Note that the rule $s^{30}/s^{30} \rightarrow \lambda; 5$ in σ_3 prevents the first X spikes from entering σ_3 and the rule $(s^{64}(s^{32})^*/s \rightarrow s; 2)$ in σ_5 prevents the spikes encoding Y from entering σ_2 . Forgetting rules $(s^{64}(s^{32})^*/s \rightarrow \lambda; 0)$ and $(s^{\langle \alpha_1 \rangle + \langle u_r \rangle}/s \rightarrow \lambda; 0)$ are applied in σ_8 , σ_9 , σ_{10} , and σ_{11} to get rid of superfluous spikes.

Given a configuration of $U_{6,4}$ the input to our spiking neural P system in Figure 2 is computed in linear time. This is done as follows; A configuration of $U_{6,4}$ is encoded as three binary sequences w_1 , w_2 , and w_3 . Each of these sequences encode a numbers from Equation 2. We then use a spiking neural P system Π_{input} with exhaustive use of rules that takes each sequence and converts it into a number of spikes that is used as input by our system in Figure 2. We give a rough idea of how Π_{input} operates. The input neuron of Π_{input} receives the binary sequence w as a sequence of spikes and no-spikes. If a 1 is read at a given timestep a single spike is sent into Π_{input} . As each bit of the binary sequence is read the total number of spikes in the system is multiplied by 2 (this is a simplification of what actually happens). Thus, Π_{input} completes its computation in time that is linear in the length of the tape contents of $U_{6,4}$. Also, w_1 , w_2 , and w_3 are computed in time that is linear in length of the tape contents of $U_{6,4}$.

Algorithm overview To help simplify the explanation, some of the rules given here in the overview differ slightly from those in the more detailed simulation below. The numbers from Equation (2), encoding a Turing machine configuration, are stored in the neurons of our systems as X , Y and $\langle \alpha_1 \rangle + \langle u \rangle$ spikes. Equation (3) is implemented in Figure 2 to give a spiking neural P system $\Pi_{U_{6,4}}$ that simulates the transition rules of $U_{6,4}$. The two values X and Y are stored in neurons σ_2 and σ_3 , respectively. If X or Y is to be multiplied the spikes that encode X or Y move down through the network of neurons from either σ_2 or σ_3 respectively, until they reach σ_{18} . Note in Figure 2 that there are synapses from σ_6 to $\sigma_8, \sigma_9, \sigma_{10}$ and σ_{11} , thus the number N of spikes in σ_6 becomes $4N$ when it fires as it sends N spikes to each neuron $\sigma_8, \sigma_9, \sigma_{10}$ and σ_{11} . If $32Y$ is to be computed we calculate $4Y$ by firing σ_6 , then $16Y$ by firing $\sigma_8, \sigma_9, \sigma_{10}$, and σ_{11} , and finally $32Y$ by firing $\sigma_{12}, \sigma_{13}, \sigma_{14}$, and σ_{15} . $32X$ is computed using the same technique.

We give the general idea of how the neurons compute $\frac{X}{32} - (\frac{X}{32} \bmod 32)$ and $(\frac{X}{32} \bmod 32)$ from Equation (3) (a slightly different strategy is used in the simulation). We begin with X spikes in σ_2 . The rule $(s^{32})^*/s^{32} \rightarrow s; 1$ is applied

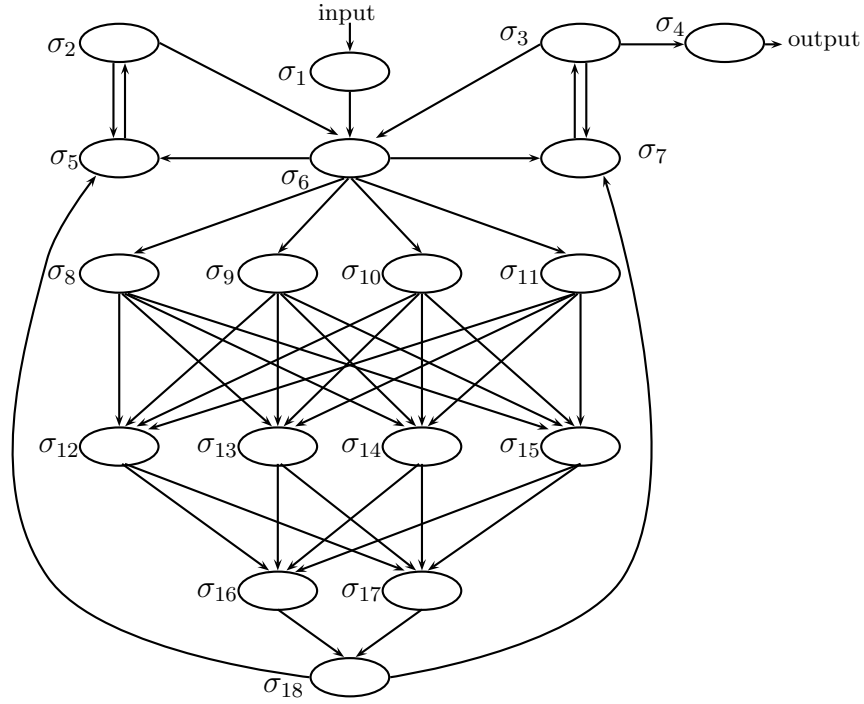


Fig. 2. Universal spiking neural P system $\Pi_{U_{6,4}}$. Each oval shape is a neuron and each arrow represents the direction spikes move along a synapse between a pair of neurons.

in σ_2 sending $\frac{X}{32}$ spikes to σ_5 . Following this $(s^{32})^* s^{(\frac{X}{32} \bmod 32)} / s^{32} \rightarrow s^{32}; 1$ is applied in σ_5 which sends $\frac{X}{32} - (\frac{X}{32} \bmod 32)$ to σ_2 leaving $(\frac{X}{32} \bmod 32)$ spikes in σ_5 . The values $\frac{Y}{32} - (\frac{Y}{32} \bmod 32)$ and $(\frac{Y}{32} \bmod 32)$ are computed in a similar manner.

Finally, using the encoded current state $\langle u_r \rangle$ and the encoded read symbol $\langle \alpha_1 \rangle$ the values $32\langle \alpha_2 \rangle$ and $\langle u_s \rangle$ are computed. Using the technique outlined in the first paragraph of the algorithm overview the value $32(\langle u_r \rangle + \langle \alpha_1 \rangle)$ is computed by sending $\langle u_r \rangle + \langle \alpha_1 \rangle$ spikes from σ_6 to σ_{18} in Figure 2. Then the rule $(s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle)}) / s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle) - \langle u_s \rangle} \rightarrow s^{32\langle \alpha_2 \rangle}; 1$ is applied in σ_{18} which sends $32\langle \alpha_2 \rangle$ spikes out to neurons σ_5 and σ_7 . This rule uses $32(\langle u_r \rangle + \langle \alpha_1 \rangle) - \langle u_s \rangle$ spikes thus leaving $\langle u_s \rangle$ spikes remaining in σ_{18} and $32\langle \alpha_2 \rangle$ spikes in both σ_5 and σ_7 . This completes our sketch of how $\Pi_{U_{6,4}}$ in Figure 2 computes the values in Equation (3) to simulate a transition rule. A more detailed simulation of a transition rule follows.

Simulation of $u_r, \alpha_1, \alpha_2, L, u_s$ (top case of Equation (3)) The simulation of the transition rule begins at time t_i with X spikes in σ_2 , Y spikes in σ_3 , and $\langle \alpha_1 \rangle + \langle u \rangle$ spikes in σ_5 and σ_7 . 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 . For example at time t_i we have

$$\begin{aligned}
t_i : \sigma_2 &= X, \\
\sigma_3 &= Y, \\
\sigma_5 &= \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow s; 1, \\
\sigma_7 &= \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow s; 1.
\end{aligned}$$

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 2 when we apply the rule $s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s \rightarrow s; 1$ in neurons σ_5 and σ_7 at time t_i we get

$$\begin{aligned}
t_{i+1} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 9, \\
\sigma_3 &= Y + \langle u_r \rangle + \langle \alpha_1 \rangle, & (s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow s; 1.
\end{aligned}$$

$$\begin{aligned}
t_{i+2} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 8, \\
\sigma_4 &= Y + \langle u_r \rangle + \langle \alpha_1 \rangle, \\
&\text{if } \langle u_r \rangle + \langle \alpha_1 \rangle = \langle u_6 \rangle + \langle c \rangle & (s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s^{32}; 1, \\
&\text{if } \langle u_r \rangle + \langle \alpha_1 \rangle \neq \langle u_6 \rangle + \langle c \rangle & (s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow \lambda; 0, \\
\sigma_6 &= Y + \langle u_r \rangle + \langle \alpha_1 \rangle, & (s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow s; 1, \\
\sigma_7 &= Y + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{32}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow \lambda; 0.
\end{aligned}$$

$$\begin{aligned}
t_{i+3} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 7, \\
\sigma_5, \sigma_7 &= Y + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{32}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow \lambda; 0, \\
\sigma_8, \sigma_9, \sigma_{10}, \sigma_{11} &= Y + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{32}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow s; 1.
\end{aligned}$$

In timestep t_{i+2} above σ_4 the output neuron fires if and only if the encoded current state $\langle u_r \rangle = \langle u_6 \rangle$ and the encoded read symbol $\langle \alpha_1 \rangle = \langle c \rangle$. The universal Turing machine $U_{6,4}$ halts if and only if it encounters the state-symbol pair (u_6, c) . Also, when $U_{6,4}$ 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 $U_{6,4}$.

Note that at timestep t_{i+3} each of the neurons $\sigma_{12}, \sigma_{13}, \sigma_{14}$, and σ_{15} receive $Y + \langle u_r \rangle + \langle \alpha_1 \rangle$ spikes from each of the four neurons $\sigma_8, \sigma_9, \sigma_{10}$, and σ_{11} . Thus at timestep t_{i+4} each of the neurons $\sigma_{12}, \sigma_{13}, \sigma_{14}$, and σ_{15} contain $4(Y + \langle u_r \rangle + \langle \alpha_1 \rangle)$ spikes. Neurons $\sigma_{12}, \sigma_{13}, \sigma_{14}$, and σ_{15} are fired at time t_{i+4} to give $16(Y + \langle u_r \rangle + \langle \alpha_1 \rangle)$ spikes in each of the neurons σ_{16} and σ_{17} at timestep t_{i+5} . Firing neurons σ_{16} and σ_{17} at timestep t_{i+5} gives $32(Y + \langle u_r \rangle + \langle \alpha_1 \rangle)$ spikes in

σ_{18} at timestep t_{i+6} .

$$\begin{aligned}
t_{i+4} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 6, \\
\sigma_{12}, \sigma_{13}, \sigma_{14}, \sigma_{15} &= 4(Y + \langle u_r \rangle + \langle \alpha_1 \rangle), & (s^{128})^* s^{4(\langle u_r \rangle + \langle \alpha_1 \rangle)} / s &\rightarrow s; 1. \\
\\
t_{i+5} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 5, \\
\sigma_{16}, \sigma_{17} &= 16(Y + \langle u_r \rangle + \langle \alpha_1 \rangle), & (s^{512})^* s^{16(\langle u_r \rangle + \langle \alpha_1 \rangle)} / s &\rightarrow s; 1. \\
\\
t_{i+6} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 4, \\
\sigma_{18}, &= 32(Y + \langle u_r \rangle + \langle \alpha_1 \rangle), & (s^{32^2})^* s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle)} / s^{32^2} &\rightarrow (s^{32^2}); 1.
\end{aligned}$$

Note that $(32Y \bmod 32^2) = 0$ and also that $32(\langle u_r \rangle + \langle \alpha_1 \rangle) < 32^2$. Thus in neuron σ_{18} at time t_{i+6} the rule $(s^{32^2})^* s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle)} / s^{32^2} \rightarrow s^{32^2}; 1$ separates the encoding of the right side of the tape s^{32Y} and the encoding of the current state and read symbol $s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle)}$. To see this note the number of spikes in neurons σ_7 and σ_{18} at time t_{i+7} .

The rule $s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle) - \langle u_s \rangle} \rightarrow s^{32\langle \alpha_2 \rangle}; 1$, applied in σ_{18} at timestep t_{i+7} , computes the new encoded current state $\langle u_s \rangle$ and the write symbol $32\langle \alpha_2 \rangle$. To see this note the number of spikes in neurons σ_7 and σ_{18} at time t_{i+8} . The reason the value $32\langle \alpha_2 \rangle$ appears in σ_7 instead of $\langle \alpha_2 \rangle$ is that the cell containing α_2 becomes part of the sequence on the right and is added to $32Y$ (as in Equation (3)) at timestep t_{i+9} . Note that $d > 1$ in σ_2 at timesteps t_{i+7} and t_{i+8} indicating σ_2 is closed. Thus the spikes sent out from σ_5 at these times do not enter σ_2 .

$$\begin{aligned}
t_{i+7} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 3, \\
\sigma_5 &= 32Y, & (s^{32})^* / s^{32} &\rightarrow s; 1, \\
\sigma_7 &= 32Y, & (s^{32})^* / s^{32} &\rightarrow s; 1, \\
\sigma_{18}, &= 32(\langle u_r \rangle + \langle \alpha_1 \rangle), & s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle) - \langle u_s \rangle} / s^{32(\langle u_r \rangle + \langle \alpha_1 \rangle) - \langle u_s \rangle} &\rightarrow s^{32\langle \alpha_2 \rangle}; 1. \\
\\
t_{i+8} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 2, \\
\sigma_3 &= 32Y, & & \\
\sigma_5 &= 32\langle \alpha_2 \rangle, & (s^{32})^* / s^{32} &\rightarrow s; 1, \\
\sigma_7 &= 32\langle \alpha_2 \rangle, & (s^{32})^* / s^{32} &\rightarrow s; 1, \\
\sigma_{18}, &= \langle u_s \rangle, & s^{\langle u_s \rangle} / s &\rightarrow s; 4. \\
\\
t_{i+9} : \sigma_2 &= X + \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{64}(s^{32})^* s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s^{32} &\rightarrow s; 1, \\
\sigma_3 &= 32Y + 32\langle \alpha_2 \rangle, & & \\
\sigma_{18}, &= \langle u_s \rangle, & s^{\langle u_s \rangle} / s &\rightarrow s; 3.
\end{aligned}$$

At time t_{i+10} in neuron σ_5 the rule $(s^{32}) * s(\frac{X}{32} \bmod 32) / s^{32} \rightarrow s^{32}; 1$ is applied sending $\frac{X}{32} - (\frac{X}{32} \bmod 32)$ spikes to σ_2 and leaving $(\frac{X}{32} \bmod 32)$ spikes in σ_5 . At the same time in neuron σ_6 the rule $(s^{32}) * s(\frac{X}{32} \bmod 32) / s^{32} \rightarrow \lambda; 0$ is applied leaving only $(\frac{X}{32} \bmod 32)$ spikes in σ_6 . Note that from Equation (1) and the value of X that $(\frac{X}{32} \bmod 32) = \langle \alpha_j \rangle$ where α_j is the symbol in cell a_{-1} at the new tape head location.

$$\begin{aligned}
t_{i+10} : \sigma_2 &= \langle u_r \rangle + \langle \alpha_1 \rangle, & s^{\langle u_r \rangle + \langle \alpha_1 \rangle} / s &\rightarrow \lambda; 0, \\
\sigma_3 &= 32Y + 32\langle \alpha_2 \rangle \\
\sigma_5 &= \frac{X}{32}, & (s^{32}) * s(\frac{X}{32} \bmod 32) / s^{32} &\rightarrow s^{32}; 1, \\
\sigma_6 &= \frac{X}{32}, & (s^{32}) * s(\frac{X}{32} \bmod 32) / s^{32} &\rightarrow \lambda; 0, \\
\sigma_{18} &= \langle u_s \rangle, & s^{\langle u_s \rangle} / s &\rightarrow s; 2.
\end{aligned}$$

$$\begin{aligned}
t_{i+11} : \sigma_2 &= \frac{X}{32} - (\frac{X}{32} \bmod 32) \\
\sigma_3 &= 32Y + 32\langle \alpha_2 \rangle \\
\sigma_5 &= \frac{X}{32} \bmod 32 & s^{\frac{X}{32} \bmod 32} / s^{\frac{X}{32} \bmod 32} &\rightarrow \lambda; 0 \\
\sigma_6 &= \frac{X}{32} \bmod 32 & s^{\frac{X}{32} \bmod 32} / s^{\frac{X}{32} \bmod 32} &\rightarrow s; 1 \\
\sigma_{18} &= \langle u_s \rangle, & s^{\langle u_s \rangle} / s &\rightarrow s; 1.
\end{aligned}$$

$$\begin{aligned}
t_{i+12} : \sigma_2 &= \frac{X}{32} - (\frac{X}{32} \bmod 32) \\
\sigma_3 &= 32Y + 32\langle \alpha_2 \rangle \\
\sigma_5 &= (\frac{X}{32} \bmod 32) + \langle u_s \rangle & s^{(\frac{X}{32} \bmod 32) + \langle u_s \rangle} / s &\rightarrow s; 1 \\
\sigma_7 &= (\frac{X}{32} \bmod 32) + \langle u_s \rangle & s^{(\frac{X}{32} \bmod 32) + \langle u_s \rangle} / s &\rightarrow s; 1, \\
\sigma_8, \sigma_9, \sigma_{10}, \sigma_{11} &= \frac{X}{32} \bmod 32 & s^{\frac{X}{32} \bmod 32} / s^{\frac{X}{32} \bmod 32} &\rightarrow \lambda; 0.
\end{aligned}$$

The simulation of the left moving transition rule is now complete. Note that the number of spikes in σ_2 , σ_3 , σ_5 , and σ_7 at timestep t_{i+12} are the values given by the top case of Equation (3) and encode the configuration after the left move transition rule.

The cases 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 $X = 32$ (recall a_{-x} contains c). If the tape head moves to the left from Equation (3) we get $X = 0$. Therefore the

length of X is increased to simulate the infinite blank symbols (c symbols) to the left as follows. The rule $s^{32+(\alpha_1)+\langle u_r \rangle} / s^{32} \rightarrow s^{32}; 1$ is applied in σ_2 at time t_{i+9} . Then at time t_{i+10} the rule $s^{32} \rightarrow s^{32}; 1$ is applied in σ_5 and the rule $s^{32} \rightarrow s; 1$ is applied in σ_6 . Thus at time t_{i+10} there are 32 spikes in σ_2 which simulates another c symbol to the left. Also at time t_{i+10} , there is 1 spike in σ_5 and σ_7 to simulate the current read symbol c .

We have shown how to simulate an arbitrary left moving transition rule of $U_{6,4}$. Right moving transition rules are also simulated in 12 timesteps in a manner similar to that of left moving transition rules. Thus a single transition rule of $U_{6,4}$ is simulated by $\Pi_{U_{6,4}}$ in 12 timesteps and from Theorem 3 the entire computation of M is simulated in $0(T^6)$ timesteps. From Theorem 3 and Equation (2) M is simulated in $0(32T^3)$ space. \square

It was mentioned at the end of Section 2 that we generalised the previous definition of spiking neural P systems with exhaustive use of rules to allow the input neuron to receive an arbitrary number of spikes in a single timestep. If the synapses of the system can transmit an arbitrary number of spikes in a single timestep, then it does not seem unreasonable to allow an arbitrary number of spikes enter the input neuron in a single timestep. This generalisation can be removed from our system. This is done by modifying the spiking neural P system Π_{input} mentioned in the subsection ‘‘Input to $\Pi_{U_{6,4}}$ ’’, and attaching its output neuron to the input neuron of $\Pi_{U_{6,4}}$ in Figure 2. The input neuron of this new system is the input neuron of Π_{input} and receives no more than a single spike at each timestep. This new universal spiking neural P system would be larger than the one in Figure 2, but there would be less work done in encoding the input.

While the small universal spiking neural P system in Figure 2 simulates Turing machines with a polynomial time overhead it *requires* an exponential space overhead. This *requirement* may be shown by proving it is simulated by a counter machine using the same space. However, it is not unreasonable to expect efficiency from simple universal systems as many of the simplest computationally universal models have polynomial time and space overheads [8, 13, 10].

A more time efficient simulation of Turing machines may be given by spiking neural P system with exhaustive rules. Using similar techniques it can be shown that for each multi-tape Turing machine M' there is a spiking neural P system with exhaustive rules that simulates M' in linear time.

$\Pi_{U_{6,4}}$ from Figure 2 is easily altered to simulate other small universal Turing machines (i.e. to simulate them directly and not via $U_{6,4}$). Using the same basic algorithm the number of neurons grows at a rate that is a log in the state-symbol product of the Turing machine it simulates. One approach to find spiking neural P systems smaller than that in Figure 2 is to simulate the universal Turing machines in [10]. These machines are weakly universal, which means that they have an infinitely repeated word to the left of their input and another to the right. The smallest of these machines has a state-symbol product of 8 and so perhaps the above algorithm could be altered to give a system with fewer neurons.

References

1. H. Chen, M. Ionescu, and T. Ishdorj. On the efficiency of spiking neural P systems. In M.A. Gutiérrez-Naranjo et al., editor, *Proceedings of Fourth Brainstorming Week on Membrane Computing*, pages 195–206, Sevilla, 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 George Eleftherakis et al., editor, *Proceedings of the Eighth Workshop on Membrane Computing*, pages 383–394, Thessaloniki, June 2007.
6. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. On the computational power of spiking neural P systems. In M.A. Gutiérrez-Naranjo et al., editor, *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, pages 227–245, Sevilla, Jan. 2007.
7. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. Solving numerical NP-complete problems with spiking neural P systems. In George Eleftherakis et al., editor, *Proceedings of the Eighth Workshop on Membrane Computing*, pages 405–423, Thessaloniki, June 2007.
8. T. Neary and D. Woods. P-completeness of cellular automaton Rule 110. In Michele Bugliesi et al., editor, *International Colloquium on Automata Languages and Programming 2006, (ICALP) Part I*, volume 4051 of *LNCS*, pages 132–143, Venice, July 2006. Springer.
9. T. Neary and D. Woods. Four small universal Turing machines. In J. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality (MCU)*, volume 4664 of *LNCS*, pages 242–254, Orléans, France, Sept. 2007. Springer.
10. T. Neary and D. Woods. Small weakly universal Turing machines. Technical Report arXiv:0707.4489v1, arXiv online report, July 2007.
11. A. Păun and G. Păun. Small universal spiking neural P systems. *BioSystems*, 90(1):48–60, 2007.
12. G. Păun. *Membrane Computing: An Introduction*. Springer, 2002.
13. D. Woods and T. Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *4th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 439–448, Berkeley, California, Oct. 2006. IEEE.