Wireless Spiking Neural P Systems

David Orellana-Martín[1,2][0000−0002−2892−6775],
Francis George C. Cabarle[1,2,3][0000−0002−5006−6310],
Prithwineel Paul[4][0000−0001−8351−3407],
XiangXiang Zeng[5][0000−0001−6201−0114], and
Rudolf Freund[6][0000−0003−1255−1953]

[1] Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012, Seville, Spain

[2] SCORE lab, I3US
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012, Seville, Spain

[3] Department of Computer Science, University of the Philippines Diliman,
Quezon City, 1101, Philippines

[4] Department of Computer Science and Engineering,
Institute of Engineering and Management,
University of Engineering and Management,
New Town Rd., Kolkata, 700091, India

[5] Department of Computer Science, Hunan University, Changsha, China

[6] Faculty of Informatics, TU Wien,
Favoritenstraße 9–11, 1040 Wien, Austria

{dorellana, fcabarle}@us.es, fccabarle@up.edu.ph,
prithwineel.paul@iem.edu.in, xzeng@hnu.edu.cn, rudi@emcc.at

Abstract. Spiking neural P systems, SN P systems in short, are computing models based on the third generation of neuron models known as spiking neurons. Recent results in neuroscience highlight the importance of *extrasynaptic* activities of neurons, that is, features and functioning of neurons apart from their synapses. Previously it was thought that signals such as *neuropeptides* only assist neurons, but such signals have been given further importance more recently. Inspired by recent results, we introduce and define *wireless SN P systems*, or *WSN P systems* in short. In WSN P systems no synapses exist, and we associate regular expressions for each neuron to decide which spikes it receives. We provide two semantics of how to "interpret" the spikes released by neurons. A specific register machine is simulated to show the different programming style of WSN P systems compared to standard SN P systems and other variants. This style emphasizes a trade-off: WSN P systems can be more "flexible" since they are not limited by their synapses for sending spikes; however, loosing the useful and directed graph structure requires careful design of the rules and the expressions associated with each neuron. We use prime numbers in constructing the expressions and rules of the neurons to prove that WSN P systems are Turing complete in both spike semantics.

Keywords: natural computing · membrane computing · spiking neural P systems · extrasynaptic signaling · neuropeptides

1 Introduction

The present work in a formal way introduces a *variant* of spiking neural P systems, in short SN P systems. SN P systems introduced in [20] are inspired by spiking neurons and their network: the *processors* are *neurons* which are the *nodes* in a *directed graph*; the *edges* are *synapses* which allow for the communication between neurons using a single object a referred to as a *spike*; the neurons are spike processors which consume and produce spikes.

Some recent survey papers of SN P systems and variants include [24,12] and more recently [6]. Since their introduction, it is known that SN P systems are Turing-complete. SN P systems can also solve **NP**-complete problems, trading time for space [25].

In the past two decades many variants of SN P systems have been introduced depending on specific ingredients or features, mostly from biology, for instance, the introduction of autapses [41], synaptic plasticity [7], polarisations [46], synaptic schedules [5], neurogenesis [45]. Besides theoretical works, simulators of SN P systems and variants are used to support research or pedagogy, such as interactive and visual software in [14] with the main page in [48], and the recent tutorial in [23]. Solutions to hard problems are also implemented in parallel hardware such as in [17] which implements ideas from [28], with recent and some state-of-the-art results in [16].

Wireless SN P systems, or *WSN P systems* in short, are a SN P system variant defined in a formal way in the present work, previously introduced in an informal way in a recent report [32]. One general reference for the bio-inspiration of WSN P systems is from [26] with recent and detailed results from [38] and [39]. Briefly, such recent results emphasise the crucial and important role of neuronal activities outside of their synapses, hence, their *wireless* features and functions. Such recent works focus their attention on a specific animal known as *C. elegans*.

The worm *elegans* is a model organism, i.e., much is known about its biology including its nervous system due to its "simplicity" of several hundred neurons only. Despite the small size of this worm, its nervous system has interesting biochemical complexity with structural features shared by larger animals [39]. Due to better techniques and technology, more recently there are improved works to show how a *wireless network* (that is, without synaptic wiring) among nerve cells or neurons is able to operate [38,39]. These recent works challenge the idea that neurons communicate only or mainly through anatomical connections, that is, through their synapses [26]. Such recent works reveal new details of a *connectome* or wiring diagram among neurons, the *neuropeptidergic connectome*: a connectome which is equally important and perhaps more diverse than the synaptic connectome.

Furthermore, these recent works identify *neuropeptides*, the chemical messages released by neurons, as the basis for such wireless network among neurons.

Neurons in the *C. elegans* worms can release neuropeptides, or have receptors for such neuropeptides. The wireless network formed from these *pairs of releasing* and *receiving neurons* is dense and decentralised, compared to the less dense and more centralised network of synapses [39]. Such pairs are responsible for the existence of the wireless network, which means that neuropeptides are not random chemicals floating between neurons. Neuropeptides affect the neural system over larger scales of time and space, unlike synaptic signals restricted only to both sides of the synapse [39]

Previously it was thought that neuropeptides only assisted in synaptic communication. However, these recent works indicate the ubiquitous, important, and *direct role to neuron activation* of neuropeptides and the corresponding wireless network [26]. Neuropeptides are conserved and ancient chemicals in brains of many organisms, including human brains, suggesting the pioneering work with *C. elegans* can at least reveal useful structures or principles for brain function [38,39]. For instance, a recent technique allows for detecting neuropeptides, which can assist in better understanding of both wired and wireless networks of neurons including those of humans [44].

We use such recent results as inspirations for *extrasynaptic* functions of neurons, that is, functioning without or outside the usual synapses. Contributions of the present work include the formal introduction of wireless SN P systems and proofs for their Turing-completeness. No synapses are present in the neurons, while still using rules to consume and produce spikes. For each neuron we associate a regular expression to decide what "forms" of spikes the neuron can receive. We introduce two semantics for WSN P systems, based on the interpretation of the spikes released in each step by the neurons: (i) the *spike package semantic* considers the spikes as individual packages as released by each neuron; (ii) the *spike total semantic* considers the sum of spikes released by all neurons.

We show how to program a specific WSN P system through the simulation of specific register machine instructions. Such a simulation emphasises the rather different way how to program WSN P systems compared with SN P systems and their variants, due to the associated expression for each neuron and the lack of synapses. Thus, we note that the directed graph structure of SN P systems and variants is a very useful feature: some "flexibility" is gained in the sense that the neurons are not limited to sending spikes only to neurons where their synapses connect; however, losing the directed graph makes the programming of the system more "involved" in the sense that more effort can be required to design the rules and neurons.

The present work is organised as follows: In Section 2 we recall some preliminaries needed to understand WSN P systems and their computations; the model definition of WSN P systems is given in Section 3. An example of a WSN P system, considered under two semantics, is used to illustrate two kinds of computations in Section 4. In Section 5 we highlight the interesting way to program WSN P systems through the simulation of a small and specific register machine. This simulation also gives us an idea of the computing power and programming of WSN P systems in a general purpose way. The proofs for computational completeness are given in Section 6. Finally, conclusions and directions for future work are in Section 7.

2 Preliminaries

In this section we only briefly mention some notions required for our definitions and results. For more details on automata and language theory we refer to [29], for their applications to membrane computing in [36,37].

Given a finite and nonempty alphabet V, by V^* we denote the set of all finite strings over V; $V^+ = V^* \setminus \{\lambda\}$. The set of all multisets over V is denoted by V°. The family of regular string languages is denoted by REG, the corresponding family of regular multiset languages by $PsREG$ (as it contains the Parikh images of the regular string languages). $NREG(a)$ denotes the family of regular multiset languages over the one-letter alphabet $\{a\}$, as these directly correspond to linear sets of natural numbers.

Definition 1. *A* register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- m *is the number of registers,*
- P *is the set of instructions bijectively labeled by elements of* B*,*
- $l_0 \in B$ *is the initial label, and*
- $l_h \in B$ *is the final label.*

The instructions of M *can be of the following forms:*

- $p : (ADD(r), q(p), s(p))$; $p \in B \setminus \{l_h\}$, $q(p), s(p) \in B$, $1 \leq r \leq m$.
  Increase the value of register r *by one, and non-deterministically jump to instruction* q *or* s*.*
- $p : (SUB(r), q(p), s(p))$; $p \in B \setminus \{l_h\}$, $q(p), s(p) \in B$, $1 \leq r \leq m$.
  If the value of register r *is not zero then decrease the value of register* r *by one (*decrement *case) and jump to instruction* $q(p)$*, otherwise jump to instruction* $s(p)$ *(*zero-test *case).*
- $l_h : HALT$.
  Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.* M *is called* deterministic *if the ADD-instructions all are of the form* $p : (ADD(r), q)$*.*

Throughout the paper, $B_{ADD(r)}$ *denotes the set of labels of ADD-instructions* $p : (ADD(r), q(p), s(p))$ *of an arbitrary registers* r*, and* $B_{SUB(r)}$ *denotes the set of labels of all SUB-instructions* $p : (SUB(r), q(p), s(p))$ *of a decrementable register* r*. Moreover, for any* $p \in B \setminus \{l_h\}$*,* $Reg(p)$ *denotes the register affected by the ADD- or SUB-instruction labeled by* p*; for the sake of completeness, in addition* $Reg(l_h) = 1$ *is taken.*

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled by l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used, i.e., they are never decremented. The set of vectors of natural numbers generated by M is denoted by $Ps(M)$; if only sets of numbers are computed, we write $N(M)$.

It is known that register machines are Turing-complete, e.g., see [30], i.e., register machines characterise NRE ($PsRE$), the family recursively enumerable sets of (vectors of) natural numbers. Hence, register machines are a convenient model to be compared with models dealing with (sets of) numbers directly instead of strings.

3 Definition of WSN P Systems

In this section, we define both the syntax and semantics of WSN P systems. In fact, two different semantics can arise from the way the spikes are treated when they are fired from a neuron. Syntax and semantics of WSN P systems share similarities with SN P systems and their variants, for instance, we refer to [20,34,37] and more recently to [24] for further details.

3.1 Syntax

Definition 2. *A* WSN P system *of degree* $m \geq 1$ *is a construct*

$$\Pi = (O, \sigma_1, \ldots, \sigma_m)$$

where:

1. $O = \{a\}$ *is the singleton alphabet (*a *is called* spike*);*
2. $\sigma_i = (n_i, E_i, R_i), 1 \leq i \leq m$, *is a neuron such that:*
   (a) $n_i \in \mathbb{N}$ *is the* initial number *of spikes in neuron* σ_i*;*
   (b) $E_i \subseteq NFIN(a)$*, the* input filter *of neuron* σ_i*;*
   (c) R_i *is a finite set of* rules *of two possible forms:*
      i. $E/a^c \to a^s$ *where* $E \subseteq NREG(a)$ *is a regular set of numbers over* O *and* $c, s, d \in \mathbb{N}, c, s \geq 1$ *(*spiking *rules);*
      ii. $a^s \to \lambda$ *where* $s \in \mathbb{N}, s \geq 1$ *(*forgetting *rules);*

A WSN P system $\Pi = (O, \sigma_1, \ldots, \sigma_m)$ of degree $m \geq 1$ can be seen as a a set of m neurons labeled by $1, \ldots, m$ such that:

1. $n_1, \ldots, n_m$ represent the *initial multisets* of objects a (spikes) situated at the beginning in the m neurons of the system;
2. $E_1, \ldots, E_m$ are finite sets over O assigned to the m neurons of the system, working as *input filters* for the spike packages allowed to enter the neuron;

3. $R_1, \dots, R_m$ are finite sets of rules governing the dynamics of the system.

Remark 1. We mention that in this paper we refrain from considering delays, as they are not needed in the following and only make definitions much more complicated.

3.2 Applicability of Rules in a WSN P System

A *configuration* of a WSN P system Π at some moment of time t is described as

$$C_t = \langle (n_{1,t}), \dots, (n_{m,t}) \rangle$$

with the number of spikes $n_{i,t}$ in each neuron i. The initial configuration of Π is $C_0 = \langle (n_1), \dots, (n_m) \rangle$.

A *spiking rule* $E/a^c \to a^s \in R_i$ is applicable to a configuration C_t at a step t if in such a configuration, there exists a neuron labelled by i such that $a^{n_{i,t}} \in E$. The application of such a rule to that neuron i produces the following effects: c spikes are removed from such a neuron i and it produces (we also say *fires*) the s spikes in the environment.

A *forgetting rule* $a^s \to \lambda \in R_i$ is applicable to a configuration C_t at a moment t if in such a configuration, there exists a neuron labelled by i such that it has exactly s spikes. The application of such a rule to that neuron i removes the s spikes from such a neuron without generating any spike.

3.3 Semantics

Two possibilities arise regarding how the produced spikes of some neurons are received by *the same or other neurons*:

1. *spike packages semantics*: Each package of spikes is treated separately in the following way:
   Let $\{a^{c_1}, \dots, a^{c_k}\}$ be the multiset of packages of spikes produced by neurons that have applied a spiking rule in the current step. Thus, for each a^{c_j}, only all the neurons σ_i such that $a^{c_j} \in E_i$ receive c_j spikes.

2. *total spikes semantics*: we take the *sum* of all the spikes produced by the neurons of the system following way:
   Let $\{a^{c_1}, \dots, a^{c_k}\}$ be the multiset of packages of spikes produced by neurons that have applied a spiking rule in the current step, and let $c = \sum_{j=1}^{k} c_j$. Then only the neurons σ_i such that $a^c \in E_i$ receive c spikes.

These two semantics in the following will be abbreviated by *pac* and *tot*, respectively.

3.4 Computations in a WSN P System

At some instance t we say the configuration C_t of the WSN P system Π produces a configuration C_{t+1} in one *step* – we denote that by $C_t \Rightarrow_\Pi C_{t+1}$ – by executing the following two substeps:

- *all* neurons apply *one rule* (if possible);
- *each* neuron σ_i according to the underlying semantics $\alpha \in \{pac, tot\}$ takes the (packages of) spikes produced in the first substep from the environment if they can pass the input filter E_i of σ_i.

We assume a *global clock* to *synchronise* the computations in Π, that is, if a neuron can apply a rule then it must do so. In every step Π is *locally sequential* since at most one rule in each neuron can be applied, but *globally parallel* as more than one neuron can apply a rule. If more than one rule in a neuron can be applied, the rule to apply is chosen in a nondeterministic way.

A *computation* of a WSN P system Π is defined as a (finite or infinite) sequence of configurations $\mathcal{C} = (C_0, C_1, \dots, C_n, \dots)$, where C_0 is the initial configuration of Π and $C_t \Rightarrow_\Pi C_{t+1}$ for all t.

If after n steps no more rules as described above can be applied, we say that Π halts after n steps, and $\mathcal{C} = (C_0, C_1, \dots, C_n)$ is called a *halting computation*.

3.5 Output

Let

$$\Pi = (O, \sigma_1, \dots, \sigma_m)$$

be a WSN P system working in the semantics $\alpha \in \{pac, tot\}$.

There are several ways how at the end of a halting computation the output of the system can be obtained:

- The output consists of a k-vector of natural numbers given by the number of spikes in some designated output neurons $\sigma j_1, \dots, \sigma j_k$; in that case the whole WSN P system is given as

  $$\Pi = (O, \sigma_1, \dots, \sigma_m; j_1, \dots, j_k),$$

  and we may also distinguish the following subcases:
  - we write $Ps_{\alpha,k-out}(\Pi)$, if the numbers in the k-vector are directly given by the number of spikes contained in the output neurons;
  - if the numbers for the output vector are encoded in the number of spikes contained in the output neurons by a specific function f like an exponential function, we write $Ps_{\alpha,out_f}(\Pi)$; as a special case, we consider f to be a linear function, in which case we also write $Ps_{\alpha,k-out_l}(\Pi)$;
- The output is obtained from a designated output neuron σ_{i_0}, $1 \le i_0 \le m$; in that case the whole WSN P system is given as

  $$\Pi = (O, \sigma_1, \dots, \sigma_m; i_0),$$

  and we may also distinguish the following subcases:

- the output vector with k components is given by $k+1$ spikes sent to the environment by the output neuron σ_{i_0}, and we write $Ps_{\alpha,k}WSNP$; given the sequence of time instances $\langle t_1, \ldots, t_{k+1} \rangle$ when the $k+1$ spikes have been sent out by the output neuron σ_{i_0}, the k components of the output vector are obtained as the *time intervals* $\langle t_2 - t_1, \ldots, t_{k+1} - t_k \rangle$; in this case we write $Ps_{\alpha,k-int}(\Pi)$;
- the output vector with k components is given by k sequences of consecutive spikes sent to the environment by the output neuron σ_{i_0}, and we write $Ps_{\alpha,k-sequ}(\Pi)$.

In all the variants described above, we replace Ps by N, if only one natural number is to be obtained as output.

The families of sets of k-vector of natural numbers obtained by WSN P systems as described above are denoted by $Ps_{\alpha,k-out}WSNP$, $Ps_{\alpha,k-out_f}WSNP$, $Ps_{\alpha,k-int}WSNP$, and $Ps_{\alpha,k-sequ}WSNP$. If only sets of natural numbers are considered, we denote the corresponding families of sets of natural numbers by $N_{\alpha,out}WSNP$, $N_{\alpha,out_f}WSNP$, $N_{\alpha,int}WSNP$, and $N_{\alpha,sequ}WSNP$.

Remark 2. In the second case described above with the designated output neuron σ_{i_0} we can think of σ_{i_0} as the *interface* of Π to the environment. As a technical detail we mention that in contrast to SN P systems and other variants, the firing of σ_{i_0} should only send spikes to the environment, but none of the neurons in Π including σ_{i_0} itself should receive the spikes produced by σ_{i_0}. This may be accomplished by avoiding a to be contained in any of the input filters E_i, $1 \leq i \leq m$.

4 An Example with the Two Semantics

In this section, as an example we consider the WSN P system Π_1 shown in Figure 1. We use Π_1 to explain the definitions and the two semantics from Section 3. For short, Π_1 has three neurons, each labeled by a pair (i, E_i) for $1 \leq i \leq 3$. Each neuron has associated the finite input filter E_i to check which number(s) of spikes it can receive. For instance, neurons σ_1 and σ_2 have $E_1 = E_2 = \{a\}$, which means they can only receive spikes of the form $a^1 = a$ fired from other neurons or for σ_1 even including spikes sent from itself. We note that the rule set of σ_2 is empty, so the number of spikes inside can only either remain the same or increase.

4.1 Semantics 1: Spike Packages

We first consider semantics 1, which we refer to as *spike packages semantics*. It only considers spikes arriving in "packages" sent by neurons to the environment, not the total number of spikes in the environment. To help with clarifying the computation of Π_1 using the spike packages semantics we refer to the configuration tree in Figure 2.

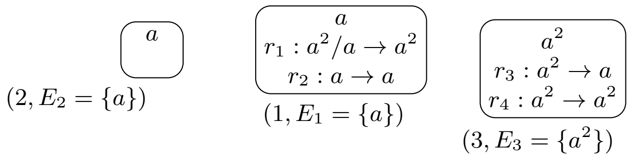

Fig. 1. Π_1 is an example of a wireless SN P system.

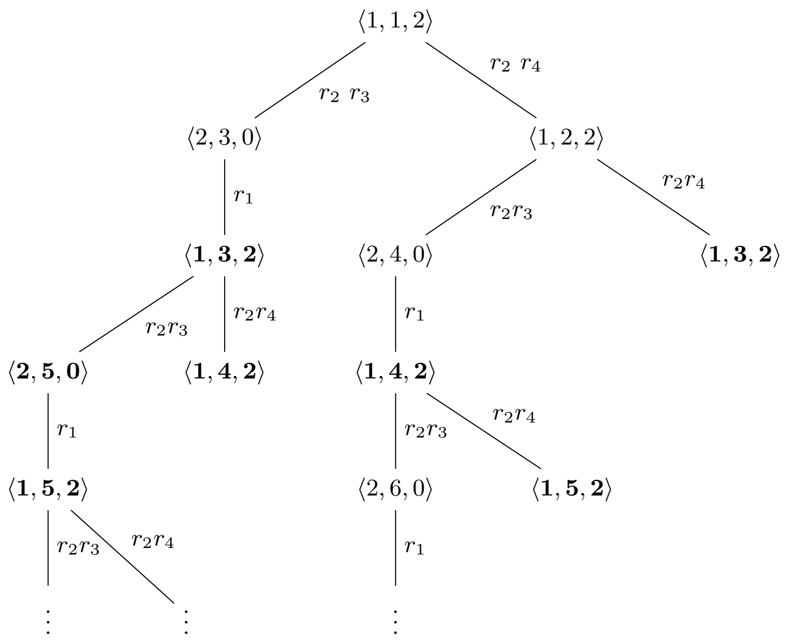
$\langle 1,1,2 \rangle$
$r_2\ r_3$
$r_2\ r_4$
$\langle 2,3,0 \rangle$
$\langle 1,2,2 \rangle$
r_1
$r_2 r_3$
$r_2 r_4$
$\langle \mathbf{1},\mathbf{3},\mathbf{2} \rangle$
$\langle 2,4,0 \rangle$
$\langle \mathbf{1},\mathbf{3},\mathbf{2} \rangle$
$r_2 r_3$
$r_2 r_4$
r_1
$\langle \mathbf{2},\mathbf{5},\mathbf{0} \rangle$
$\langle \mathbf{1},\mathbf{4},\mathbf{2} \rangle$
$\langle \mathbf{1},\mathbf{4},\mathbf{2} \rangle$
r_1
$r_2 r_3$
$r_2 r_4$
$\langle \mathbf{1},\mathbf{5},\mathbf{2} \rangle$
$\langle 2,6,0 \rangle$
$\langle \mathbf{1},\mathbf{5},\mathbf{2} \rangle$
$r_2 r_3$
$r_2 r_4$
r_1

Fig. 2. The tree of configurations of Π_1 in Figure 1 using semantics 1 (spike packages semantics). The initial configuration is $\langle 1,1,2 \rangle$. Except for $\langle 1,1,2 \rangle$, each *node* in the tree is a successor configuration obtained by having applied the rules labeling the connecting *edge*. Nodes (configurations) in bold are nodes repeated elsewhere in the portion of the tree depicted here.

The initial configuration of Π_1, assuming the (contents of the) neurons to be listed according to the total ordering $1,2,3$, is $C_0 = \langle 1,1,2 \rangle$, i.e., neurons 1, 2, and 3 contain 1, 1, and 2 spikes, respectively. Due to C_0 and the nondeterminism in Π_1 found only in neuron σ_3, there is a choice between applying rule r_2 as well as either rule r_3 or rule r_4.

If rule r_2 is applied, one spike is consumed in neuron σ_1 and sent to both neuron σ_1 and neuron σ_2 due to their input filters $E_1 = E_2 = \{a\}$.

Also applying rule r_3 means that σ_3 consumes two spikes but fires only one spike. Again this single spike from σ_3 arrives in σ_1 and σ_2 due to their input filters. Hence, in total we have got the transition $C_0 \overset{r_2 r_3}{\Longrightarrow} C_{1,0} = \langle 2, 3, 0\rangle$, i.e., by applying r_2 and r_3 we obtain configuration $C_{1,0}$ from configuration C_0.

Now we onsider the case when we apply rule r_4 together with rule r_2 instead. The effect of applying rule r_2 is still to return a spike to σ_1 and to increment the number of spikes in σ_2. The effect of r_4 is *reflexive*, i.e., in neuron σ_3 two spikes are consumed and then returned to itself, since $E_3 = \{a^2\}$. Hence, we have the transition $C_0 \overset{r_2 r_4}{\Longrightarrow} C_{1,1} = \langle 1, 2, 2\rangle$, i.e., by applying r_2 and r_4 we obtain configuration $C_{1,1}$ from configuration C_0.

Hence, in total, from the initial configuration C_0 we get the two successor configurations $C_{0,1} = \langle 2, 3, 0\rangle$ and $C_{0,2} = \langle 1, 2, 2\rangle$ as depicted in the tree of Figure 2.

As can be seen in the configuration tree in Figure 2, each branch of computations in Π_1 is non-halting, i.e., Π_1 always arrives at a configuration where some rule can still be applied. The number of spikes in neuron σ_2 continues to increase. More precisely, for all $b \geq 1$ we have the following transitions:

- $\langle 2, b, 0\rangle \overset{r_1}{\Longrightarrow} \langle 1, b, 2\rangle$,
- $\langle 1, b, 2\rangle \overset{r_2 r_3}{\Longrightarrow} \langle 2, b+2, 0\rangle$,
- $\langle 1, b, 2\rangle \overset{r_2 r_4}{\Longrightarrow} \langle 1, b+1, 2\rangle$.

4.2 Semantics 2: Total Spikes

We now consider the WSN P system Π_1 from Figure 1 together with the total spikes semantics. The corresponding configuration tree of Π_1 is now given by Figure 3.

From the same initial configuration $C_0 = \langle 1, 1, 2\rangle$ the computation proceeds in a different way:

The transition $C_0 \overset{r_2 r_4}{\Longrightarrow} C_{1,1} = \langle 0, 1, 0\rangle$ is a halting configuration, i.e., no more rules can be applied in Π_1. We note that the effect of applying rules r_2 and r_4 from C_0 is to release a total number of 3 spikes followed by the halting of Π_1, because the three spikes cannot enter any of the neurons, since none of them has a^3 in its input filter.

Only the subtree with transition $C_0 \overset{r_2 r_3}{\Longrightarrow} C_{1,0} = \langle 0, 1, 2\rangle$ continues to infinitely grow the number of spikes in neuron σ_2. In fact, for all $b \geq 1$, applying rule r_4 to the configuration $\langle 1, b, 0\rangle$ yields the configuration $\langle 1, b, 0\rangle$ again, whereas applying rule r_2 to the configuration $\langle 1, b, 0\rangle$ yields the configuration $\langle 1, b+1, 0\rangle$. In both cases we see that every branch leads to a non-halting computation.

5 Programming WSN P systems

In order to give an idea how to program WSN P systems, including their similarities and differences with SN P systems and their other variants, we consider a very small register machine M:

$$M = (m, B = \{1, 2, 3\}, l_0 = 2, l_h = 3, P)$$

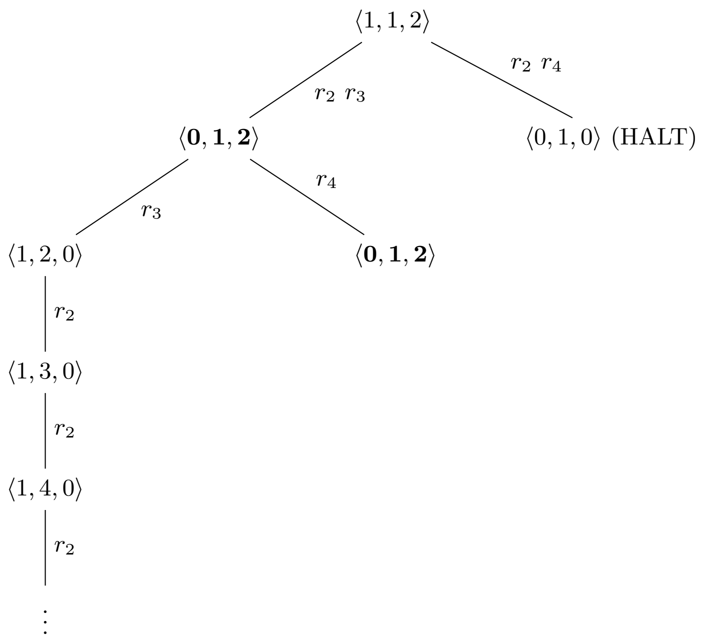

Fig. 3. Configuration tree for Π_1 in Figure 1 using semantics 2 (total spikes semantics). As in Figure 2, edges between nodes (configurations) are labeled by the rules applied from the source to destination nodes. Moreover, configurations in bold means they are repeated elsewhere in the tree.

with the following instructions in P:

$$1:(ADD(r_1),2,3), 2:(SUB(r_1),1,3), \text{ and } 3:HALT.$$

We just mention that this very simple register machine model toggles between incrementing and decrementing register r_1 using $1:(ADD(r_1),2,3)$ and $2:(SUB(r_1),1,3$ alternatingly, always choosing the first option in the *ADD*-instruction, before finally ending up with choosing the second option to halt, with the final contents of the register being 1. Observe that we never reach the zero-test branch of the *SUB*-instruction.

We simulate the instructions of M using a WSN P system Π_M defined as follows: We map prime numbers to elements of M and use the mapping as *addresses* in Π_M. The idea of addresses and their use in the simulation will become clear in a moment. In general, for elements of an arbitrary register machine we use the total order $\langle l_1, l_2, \ldots, r_1, r_2, \ldots\rangle$ and the correponding odd prime numbers $\langle P(l_1), P(l_2), \ldots, P(r_1), P(r_2), \ldots\rangle$, where all the prime numbers in this list have to be different.

Specific for M we use the following mapping of its elements to prime numbers:

$$P(1) = 11, P(2) = 13, P(2) = 17, P(r_1) = 19.$$

Starting from the first instruction labeled by 1 of M, we map it to the prime number 11, followed by mapping 2 and 3 to 13 and 17, respectively. After having mapped prime numbers to all the instructions of M, we map the next prime numbers to registers: there is only one register in M, i.e., r_1, which is mapped to 19. The reason why we start the mapping with the first prime number to be 11 will become clear in a moment: we use 3, 5, and 7 for different purposes. In general, if we need some of the first prime numbers for specific programming purposes, we may start the mapping with any odd prime number and continue with the next prime numbers in a consecutive way.

In general, the mapping we use for the contents n of register n is having $a^{2P(r_i)n}$ spikes in the neuron σ_{r_i}. Following the mapping of prime numbers above to elements of M: if n is the number stored in register r_1, the associated neuron σ_{r_1} carries $a^{P(r_1)n} = a^{2*19n}$ spikes.

The WSN P system Π for M now is defined as follows:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_{aux_1}, \sigma_{reg_1})$$

For every production labeled by $1, 2, 3$, Π contains the corresponding neuron
$\sigma_1 = (a^{11}, R_1 = \{a^{11} \to a^{11*19}\}, E_1 = \{a^{11}),$
$\sigma_2 = (\lambda, R_2 = \{a^{13} \to a^{13*19}\}, E_2 = \{a^{13}\}),$
$\sigma_3 = (\lambda, \emptyset, E_3 = \{a^{17}\}).$

Since 1 is the initial instruction we assume that at the beginning σ_1 contains 11 spikes, i.e., a^{11}, to allow for the application of its rule and thus to start the simulation.

In general, if instruction p is to be applied in M, then the corresponding simulation in Π starts by applying the rule in neuron σ_p.

If we chose the output strategy out_l we can stop the simulation of M as soon as the $HALT$-instruction 3 is reached, which simply is implemented by having no rules in the corresponding neuron σ_3, i.e., also the computation in Π halts.

At the beginning we assume register 1 to be empty, hence, the initial number of spikes in neuron 1 is empty (indicated by $a^0 = \lambda$).

$\sigma_{aux_1} = (\lambda, R_{aux_1,ADD} \cup R_{aux_1,SUB}, E_{aux_1} = \{a^{11*19}, a^{13*19}, a^{17*19}\};$
$\sigma_{reg_1} = (\lambda, R_{1,ADD} \cup R_{1,SUB}, E_{reg_1} = \{a^{2*19}, a^{3*19}\}).$

$R_{aux_1,ADD} = \{a^{11*19}/a^{11*18} \to a^{2*19}, a^{11} \to a^{13}, a^{11} \to a^{17}\};$

$R_{aux_1,SUB} = \{a^{13*19}/a^{13*18} \to a^{3*19}, a^{13+5*19} \to a^{11}, a^{13+7*19} \to a^{17}\};$
$R_{reg_r,SUB} = \{\{a^{(5+j)*19} \mid 0 \leq j\}/a^{5*19} \to a^{5*19}, a^{3*19} \to a^{7*19}\}.$

Simulating the *ADD*-Instruction

The simulation of the ADD-instruction $1 : (ADD(r_1), 2, 3)$ is started whenever neuron σ_1 contains a^{11}, for example, already in the initial configuration. So let us assume that at time t the whole configuration described by contents of the five neurons (described by natural numbers) is

$C(t) = \langle 11, 0, 0, 0, 2 * 19n \rangle$
for some $n \geq 0$.
Having 11 spikes in neuron σ_1 allows for the application of the rule
$a^{11} \to a^{11*19}$,
and the $11 * 19$ spikes according to the input filters can enter neuron σ_{aux_1}, i.e., we obtain
$C(t+1) = \langle 0, 0, 0, 11 * 19, 2 * 19n \rangle$.

Now in neuron σ_{aux_1} the rule
$a^{11*19}/a^{11*18} \to a^{2*19}$
is to be applied, leaving 11 spikes in neuron σ_{aux_1}, whereas $2 * 19$ spikes can enter the register neuron σ_{reg_1}, i.e., we obtain
$C(t+2) = \langle 0, 0, 0, 11, 2 * 19(n+1) \rangle$.

Finally, one of the two rules $a^{11} \to a^{13}$ and $a^{11} \to a^{17}$ is applied in neuron σ_{aux_1}, thus yielding one of the configurations
$C(t+3) = \langle 0, 13, 0, 0, 2 * 19(n+1) \rangle$ (activating neuron σ_2)
and
$C(t+3) = \langle 0, 0, 17, 0, 2 * 19(n+1) \rangle$ (activating neuron σ_3).

Simulating the *SUB* instruction

Let us assume that neuron σ_2 has been activated by having 13 spikes in this neuron, i.e., at time t the whole configuration described by contents of the five neurons (described by natural numbers) is
$C(t) = \langle 0, 13, 0, 0, 2 * 19n \rangle$
for some $n \geq 0$. Having 11 spikes in neuron σ_2 allows for the application of the rule $a^{13} \to a^{13*19}$, and the $13 * 19$ spikes according to the input filters can enter neuron σ_{aux_1}, i.e., we obtain
$C(t+1) = \langle 0, 0, 0, 13 * 19, 2 * 19n \rangle$.

The simulation of the *SUB*-instruction now proceeds with applying the rule $a^{13*19}/a^{13*18} \to a^{3*19}$ in neuron σ_{aux_1}. The $3 * 19$ spikes can only enter neuron σ_{reg_1}, i.e., we obtain the configuration
$C(t+2) = \langle 0, 0, 0, 13, 2 * 19n + 3 * 19 \rangle$.

In the third step of the simulation, the continuation depends on the value of n, i.e., on the contents of neuron σ_{reg_1}:

$n > 0$ With at least $5 * 19$ spikes in σ_{reg_1}, the rule
$\{a^{(5+j)*19} \mid 0 \leq j\}/a^{5*19} \to a^{5*19}$
is applied and the $5 * 19$ spikes are entering neuron σ_{aux_1}, which results in the configuration
$C(t+3) = \langle 0, 0, 0, 13 + 5 * 19, 2 * 19(n-1) \rangle$.

Finally, in the fourth step of the simulation, in σ_{aux_1} the rule $a^{13+5*19} \to a^{11}$ is applied, yielding 11 spikes in σ_1 and the configuration

$C(t+4) = \langle 11, 0, 0, 0, 2 * 19(n-1) \rangle$.

$n = 0$ With only $3 * 19$ spikes in σ_{reg_1}, the rule $a^{3*19} \to a^{7*19}$ has to be applied, and the $7*19$ spikes are entering neuron σ_{aux_1}, which results in the configuration
$C(t+3) = \langle 0, 0, 0, 13 + 7 * 19, 0 \rangle$.

Finally, in the fourth step of the simulation, in σ_{aux_1} the rule $a^{13+7*19} \to a^{17}$ is applied, yielding 17 spikes in σ_3 and the configuration
$C(t+4) = \langle 0, 0, 17, 0, 0 \rangle$.

According to the explanations given above, we see that the subtraction instruction $2 : (SUB(r_1), 1, 3)$ of M is correctly simulated: if register r_1 contains a non-zero value, then it is decremented, i.e., the number of spikes in neuron σ_{reg_1} is reduced by $2*19$, and the next instruction to be simulated is 1, otherwise the number in register r_1 and also the number of spikes in σ_{reg_1} finally remains zero and 3 is the next instruction.

Using prime numbers as "addresses" for each neuron and for choosing the correct rules allows for correct simulations. Such addressing we use not only in the input filters associated with each neuron, but also with the number of spikes released by the neurons.

6 Turing-completeness of WSN P Systems

In this section we generalise the ideas provided in Section 5 to prove that WSN P systems are Turing-complete by simulating an arbitrary register machine. In Theorem 1 we show that WSN P systems can generate any language in $PsRE$ using the spike packages semantics, and from the proof of this theorem immediately in Corollary 1 that the same can be achieved with WSN P systems using the the total spikes semantics.

Theorem 1. *$Ps_{pac,out_l}WSNP = PsRE$.*

Proof. We only prove one direction, i.e., the inclusion $PsRE \subseteq Ps_{pac,out_l}WSNP$.
Consider an arbitrary register machine

$$M = (m, B, l_0, l_h, P)$$

with $|B| = l = |P|$ and the last k of the m registers being the output registers, which according to Section 2 are never *never associated* with a *SUB*-instruction, i.e., the value in these registers only increases or remains unchanged.

We now construct a WSN P system Π to simulate M and its instructions. Without loss of generality we assume a total order starting with the instructions, followed by the registers, of M, i.e., we list instructions and registers as

$$\langle l_0, l_1, \ldots, l_h, r_1, \ldots, r_m \rangle = \langle 1, 2, \ldots, l_h = l, l+1, \ldots, l+m \rangle.$$

Next we assign a prime number $P(i)$ to each element i of this list, starting from an arbitrary odd prime number in such a way that the preceding ones can be used for other purposes as already explained for the example in the previous section. In the current proof, we may start with $P(l_0) = P(1) = 11$, i.e, the 4-th odd prime number, $P(2) = 13, \ldots$, $P(l_h) = P(l) = (4+l)$th odd prime number, followed by $P(r_1) = P(l+1) = (l+5)$th odd prime number, $\ldots, P(r_m) = P(l+m) = (4+l+m)$th odd prime number.

If a register r_i contains the number n then the corresponding neuron σ_{r_i} contains $a^{2P(r_i)n}$ spikes, i.e., the number n is encoded by the linear function $2P(r_i)n$. Specific neurons in Π are used to simulate the *ADD*- and *SUB*-instructions of M, respectively.

The WSN P system Π now is defined as follows:

$$\Pi = (\{a\}, \sigma_1, \ldots, \sigma_l, \sigma_{aux_1}, \ldots, \sigma_{aux_m}, \sigma_{reg_1}, \ldots, \sigma_{reg_m})$$

For every production (labeled by p), Π contains the neuron

$$\sigma_p = (\lambda, R_p, E_p = \{a^{P(p)}\})$$

with $R_p = \{a^{P(p)} \to a^{P(p)P(r_{Reg(p)})}\}$ for any $2 \leq p \leq l-1$.

Since $l_0 = 1$ is the initial instruction we assume that at the beginning σ_{l_0} contains $a^{P(l_0)}$, i.e., $a^{P(1)}$, spikes to allow for the application of its rule and thus to start the simulation, i.e.,

$$\sigma_1 = (a^{P(1)}, R_1, E_1 = \{a^{P(1)}\})$$

In general, if instruction p is to be applied in M, then the corresponding simulation in Π starts by applying the rule in neuron σ_p.

For the chosen output strategy out_l we can stop the simulation of M as soon as the $HALT$-instruction is reached, which simply can be implemented by having no rules in the corresponding neuron σ_{l_h}, i.e.:

$$\sigma_{l_h} = \sigma_l = (\lambda, \emptyset, E_{l_h} = E_l = \{a^{P(l_h)}\})$$

We already here mention that for other output strategies, neuron σ_{l_h} may have to start a rather complex output module to obtain the desired output.

All registers at the beginning are empty, hence, the initial number of spikes in the following neurons is empty (indicated by $a^0 = \lambda$).

As will become obvious later, for every register r, max_r can be defined as
$max_r := max\{P(p) \mid p \in B_{ADD(r)} \cup B_{SUB(r)}\}$.

For every decrementable register r, $1 \leq r \leq m-k$, we take:
$\sigma_{aux_r} = (\lambda, R_{aux_r,ADD} \cup R_{aux_r,SUB}, E_{aux_r} = \{a^{jP(reg_r)} \mid 1 \leq j \leq max_r\}$;
$\sigma_{reg_r} = (\lambda, R_{r,ADD} \cup R_{r,SUB}, E_{reg_r} = \{a^{2P(reg_r)}, a^{3P(reg_r)}\})$.

$R_{aux_r,ADD} = \{a^{P(p)P(reg_r)}/a^{P(p)(P(reg_r)-1)} \to a^{2P(reg_r)}, a^{P(p)} \to a^{P(q(p))}, a^{P(p)} \to a^{P(s(p))} \mid p \in B_{ADD(r)}\}$;

$R_{aux_r,SUB} = \{a^{P(p)P(reg_r)}/a^{P(p)(P(reg_r)-1)} \to a^{3P(reg_r)}, a^{P(p)+5P(reg_r)} \to a^{P(q(p))}, a^{P(p)+7P(reg_r)} \to a^{P(s(p))} \mid p \in B_{SUB(r)}\}$;

$R_{reg_r,ADD} = \emptyset$;

$R_{reg_r,SUB} = \{\{a^{(5+j)P(reg_r)} \mid 0 \leq j\}/a^{5P(reg_r)} \to a^{5P(reg_r)}, a^{3P(reg_r)} \to a^{7P(reg_r)}\}$.

For every non-decrementable output register r, $m-k \leq r \leq m$, we simply take $R_{aux_r,SUB} = R_{reg_r,SUB} = \emptyset$.

In the following two subsections we are going to explain how an *ADD*-instruction and a *SUB*-instruction are simulated by the neurons of Π as defined above.

Simulating an *ADD*-instruction

Consider an *ADD*-instruction $p : (ADD(r), q(p), s(p))$ with $p \in B_{ADD(r)}$. The simulation of this *ADD*-instruction in Π involves the neuron for the instruction p

$\sigma_p = (\lambda, R_p, E_p = \{a^{P(p)}\})$

as well as the neurons for the affected register $Reg(p) = r$, i.e., σ_{aux_r} and σ_{reg_r} with the only relevant rule set

$R_{aux_r,ADD} = \{a^{P(p)P(reg_r)}/a^{P(p)(P(reg_r)-1)} \to a^{2P(reg_r)}, a^{P(p)} \to a^{P(q(p))}, a^{P(p)} \to a^{P(s(p))} \mid p \in B_{ADD(r)}\}$.

Suppose that the simulation of instruction p starts at time t, i.e., at time t neuron σ_p has $P(p)$ spikes, and neuron σ_{reg_r} carries $2P(reg_r)n$ spikes encoding the number n; moreover, neuron σ_{aux_r} is empty. This situation can be captured by describing the subconfiguration $C'(t) = \langle P(p), 0, 2P(reg_r)n\rangle$ only showing the contents of the three neurons σ_p, σ_{aux_r},and σ_{reg_r}. The simulation now takes three steps:

Step 1 In the first step, neuron σ_p applies its only spiking rule, consumes all $P(p)$ spikes and firing $P(p)P(reg_r)$ spikes, which only neuron σ_{aux_r} can receive due to its input filter $E_{aux_r} = \{a^{(3+j)P(reg_r)} \mid 0 \leq j \leq max\}$; hence, we obtain
$C'(t+1) = \langle 0, P(p)P(reg_r), 2P(reg_r)n\rangle$.

Step 2 In the next step, the $P(p)P(reg_r)$ spikes in neuron σ_{aux_r} allow it to apply the rule (and only this one)
$a^{P(p)P(reg_r)}/a^{P(p)(P(reg_r)-1)} \to a^{2P(reg_r)}$,
which consumes $P(p)(P(reg_r)-1)$ spikes and fires $2P(reg_r)$, which number of spikes can only enter neuron σ_r, as its input filter is the only one containing $a^{2P(reg_r)}$ spikes. Neuron σ_r receiving $2P(reg_r)$ spikes corresponds to incrementing register r. Hence, we obtain
$C'(t+2) = \langle 0, P(p), 2P(reg_r)(n+1)\rangle$.

Step 3 Neuron σ_{aux_r} now has $P(p)$ spikes. Then two rules can be applied, to be chosen in a nondeterministic way: applying either rule $a^{P(p)} \to a^{P(q(p))}$ or rule $a^{P(p)} \to a^{P(s(p))}$, which corresponds to sending spikes either to neuron $\sigma_{q(p)}$ or neuron $\sigma_{s(p)}$. We finally have
$C'(t+3) = \langle 0, 0, 2P(reg_r)(n+1)\rangle$ (if neither $q(p)$ nor $s(p)$ are p again, otherwise, if p is activated again, we have
$C'(t+3) = \langle a^{P(p)}, 0, 2P(reg_r)(n+1)\rangle$.

In this way, the *ADD* instruction is correctly simulated: starting from the activation of neuron σ_p, the contents of σ_r is increased by $2(p_r)$ spikes, corresponding to an increment of register r. Afterwards, the computation continues either with instruction $q(p)$ or instruction $s(p)$, chosen in a nondeterministic way, which corresponds to activating either neuron $\sigma_{q(p)}$ or neuron $\sigma_{s(p)}$.

Simulating a *SUB*-instruction

Consider an instruction $p : (SUB(r) : q, s$ in $SUB(r)$. The simulation of this *SUB*-instruction in Π involves the neuron for the instruction p
$\sigma_p = (a^{P(p)}, R_p, E_p = \{a^{P(p)}\})$
as well as the neurons for the affected register $Reg(p) = r$, i.e., σ_{aux_r} and σ_{reg_r} with the main relevant rule sets

$R_{aux_r,SUB} = \{a^{P(p)P(reg_r)}/a^{P(p)(P(reg_r)-1)} \to a^{3P(reg_r)}, a^{P(p)+5P(reg_r)} \to a^{P(q(p))}, a^{P(p)+7P(reg_r)} \to a^{P(s(p))} \mid p \in B_{SUB(r)}\}$;

$R_{reg_r,SUB} = \{\{a^{(5+j)P(reg_r)} \mid 0 \leq j\}/a^{5P(reg_r)} \to a^{5P(reg_r)}, a^{3P(reg_r)} \to a^{7P(reg_r)}\}$.

Suppose that the simulation of instruction p starts at time t, i.e., at time t neuron σ_p has $P(p)$ spikes, and neuron σ_{reg_r} carries $2P(reg_r)n$ spikes encoding the number n; moreover, neuron σ_{aux_r} is empty. This situation can be captured by describing the subconfiguration $C'(t) = \langle P(p), 0, 2P(reg_r)n\rangle$ only showing the contents of the three neurons σ_p, σ_{aux_r},and σ_{reg_r}. The simulation now takes four steps:

Step 1 In the first step, neuron σ_p applies its only spiking rule, consumes all $P(p)$ spikes and fires $P(p)P(reg_r)$ spikes, which only neuron σ_{aux_r} can receive due to its input filter $E_{aux_r} = \{a^{(5+j)P(reg_r)} \mid 0 \leq j \leq max\}$; hence, we obtain
$C'(t+1) = \langle 0, P(p)P(reg_r), 2P(reg_r)n\rangle$.

Step 2 In the next step, the $P(p)P(reg_r)$ spikes in neuron σ_{aux_r} allow it to apply the rule (and only this one)
$a^{P(p)P(reg_r)}/a^{P(p)(P(reg_r)-1)} \to a^{3P(reg_r)}$,
which consumes $P(p)(P(reg_r)-1)$ spikes and fires $3P(reg_r)$ spikes, which number of spikes can only enter neuron σ_r, as its input filter is the only one containing $a^{3P(reg_r)}$ spikes. Hence, we obtain

$C'(t+2) = \langle 0, 3P(reg_r), 2P(reg_r)n + 3\rangle$.

We observe that the number of spikes in neuron σ_{reg_r} now is either at least $5P(reg_r)$ if $n > 0$ or else only $3P(reg_r)$.

$n > 0$: The number of spikes in neuron σ_{reg_r} now is at least $5P(reg_r)$:

Step 3 Hence, the rule $\{a^{(5+j)P(reg_r)} \mid 0 \leq j\}/a^{5P(reg_r)} \to a^{5P(reg_r)}$ can be applied, i.e., $5P(reg_r)$ spikes can enter neuron σ_{aux_r}, but no other neuron, i.e.,
$C'(t+3) = \langle 0, 5P(reg_r), 2P(reg_r)(n-1)\rangle$.

Step 4 σ_{aux_r} now has $5P(reg_r)$ spikes and therefore applies rule
$a^{P(p)+5P(reg_r)} \to a^{P(q(p))}$,
thus activating neuron $\sigma_{q(p)}$ for the next step, and we finish with
$C'(t+4) = \langle 0, 0, 2P(reg_r)(n-1)\rangle$ if $q(p) \neq p$, otherwise, σ_p is activated again, and we have
$C'(t+4) = \langle a^{P(q(p)),0,2P(reg_r)(n-1)}\rangle$.

$n = 0$: The number of spikes in neuron σ_{reg_r} now is exactly $3P(reg_r)$.

Step 3 Hence, the rule $a^{3P(reg_r)} \to a^{7P(reg_r)}$ has to be applied, i.e., $7P(reg_r)$ spikes can enter neuron σ_{aux_r}, but no other neuron, i.e.,
$C'(t+3) = \langle 0, 7P(reg_r), 0\rangle$.

Step 4 σ_{aux_r} now has $7P(reg_r)$ spikes and therefore applies rule
$a^{P(p)+7P(reg_r)} \to a^{P(s(p))}$,
thus activating neuron $\sigma_{s(p)}$ for the next step, and we finish with
$C'(t+4) = \langle 0, 0, 0\rangle$ if $s(p) \neq p$, otherwise, σ_p is activated again, and we have
$C'(t+4) = \langle a^{P(q(p))}, 0, 0\rangle$.

Output

As the output vector of dimension k $\langle n_1, \ldots, n_k\rangle$ is encoded in the k output neurons $\langle \sigma_{reg_{m-k+1}}, \ldots, \sigma_{reg_m}\rangle$ in linear functions $\langle 2P(m-k+1)n_1, \ldots, 2P(m-k+1)n_k\rangle$, finally we write the WSN P system as:

$$\Pi = (\{a\}, \sigma_1, \ldots, \sigma_l, \sigma_{aux_1}, \ldots, \sigma_{aux_m}, \sigma_{reg_1}, \ldots, \sigma_{reg_m}, l+m-k+1, \ldots, l+m)$$

Summing up, we have shown that $Ps(M) = Ps_{pac,k-out_l}(\Pi)$. □

Corollary 1. *$Ps_{tot,out_l}WSNP = PsRE$.*

Proof. All modules in the proof for Theorem 1 are *sequential*, i.e., in any step, at most one neuron fires. Hence, the modules also work together with the total spikes semantics. □

Obviously, the two preceding results immediately infer the following result for sets of natural numbers:

Corollary 2. *$N_{pac,out_l}WSNP = N_{tot,out_l}WSNP = NRE$.*

7 Discussions and Future Directions

In the present paper we have defined the syntax and semantics of wireless SN P systems (for short, *WSN P systems* and proved their Turing-completeness, based on the ideas started in the report [32]. Although the Turing-completeness of WSN P systems has been expected, it is very interesting to see the effects of losing the directed graph structure of SN P systems: in order to cope with this problem, we associated finite filters with each neuron, and thus could use prime numbers as "neuron addresses" when proving Turing-completeness.

The main novelty which can be found in WSN P systems is the idea of abandoning a static graph structure for the connection between the neurons being an inherent part SN P systems or in general common in neural systems or networks. This idea was motivated especially by recent discoveries in neuroscience, such as those mentioned in Section 1, i.e., our increasing knowledge of extrasynaptic signaling, of neuropeptides and their important influence in neuronal activities. We also introduced two bio-inspired semantics how the "floating" spikes are received by the neurons: the *spike packages semantics* and the *total spikes semantics*. As a main feature for the communication between neurons we associate a finite filter to each neuron thus allowing neurons to decide which (number of) spikes to be taken from the spikes present in the environment after the firing of the neurons. The the *spike packages semantics* also bears some resemblance to using packets of data among networks of computers that, for instance, connect wireless networks and the Internet.

The use of forgetting rules of the form $a^s \to \lambda$, used to remove spikes without producing spikes is common to SN P systems and many variants, but such rules are not used in our completeness proofs (Section 6).

Another way to avoid such rules is to use rules like $a^s \to a^x$ where x is not found in the input filter of any neuron. In this way we still remove the s number of spikes, which works perfectly for the but more care needs to be given using the *total spikes semantics*.

In many variants of SN P systems the *delay* feature is common: there can be a non-zero delay from releasing a spike and the spike arriving to another neuron. It is well known, e.g., see [27], that the delay feature is not required for obtaining universality; hence, we have not considered delays in this paper. On the other hand, using delays can be useful, for instance, in modelling [8]. Therefore it may be interesting to investigate the role of delays in WSN P systems.

Other common features of SN P systems and variants include the lack of reflexive synapses, or restricting the number of spikes produced by a neuron to be at most the number of consumed spikes. A variant known as SN P systems with autapses allows reflexive synapses, but this variant has a static directed graph as connection structure between the neurons, see [41]. For restricting the number of produced spikes to be less or equal to the number of consumed spikes, perhaps this can be achieved by using more time for the computation, using additional neurons to generate the required spikes.

Regarding the two semantics as defined in Section 4, it is interesting to see which types of problems or advantages computations in one semantics have over computations in the other one. As can be seen in the configuration trees in Figure 2 and Figure 3, for the same Π_1 the computations can be rather different.

Another interesting extension or semantics for WSN P systems is the idea of *decay* or *attenuation* of spikes: it is assumed that spikes (especially if delays are involved) can "float" without change for an arbitrary duration in time or distance in space between the neurons. It may be interesting to introduce such decay or attenuation in WSN P systems, similar to the decay of electromagnetic signals used in wireless networks of computers. Decaying spikes were already considered in SN P systems, e.g., see [13].

In programming Π_M, we could make it operating in a sequential way, i.e., in each step at most one neuron applies a rule. The sequential restriction has been applied to SN P systems as early as in [18], and more recently with variants having dynamic topologies in [7,5]. It may be interesting to investigate the influence of using more parallelism and to see what kinds of restrictions and computations can/cannot be obtained in terms of neurons, rules, etc.

Another interesting direction is to consider matrix representations of WSN P systems, as done for SN P systems in [47] and more recently in [2]. Such representations allow for faster simulations, such as parallel processors [16,1], web browsers [14,31], and their automatic design [15,42]. A related variant with matrix representation seems to be the model of SNPSP systems in [22]. SNPSP systems introduce plasticity to allow adding or removing of synapses, as introduced in [7].

Another variant known as SNP systems with scheduled synapses (in short, SSNP systems) has synapse dynamics, by assigning schedules or (range of) time steps when synapses exist or not. Besides SNPSP systems and SSNP systems, another related variant are extended SN P systems in [4] which also have no fixed and directed graph structure. We also may consider WSN P systems in the context of the formal framework [43] for spiking neural P systems.

A few other lines of investigation on the computing power to be considered in the future are the following ones:

- generating string languages with WSN P systems, for instance, see [9];
- providing "small" WSN P systems as in [33];
- realising or characterising *sub-Turing* computations of certain WSN P systems, especially to obtain *decidable* properties;
- studying normal forms, such as restricting the size of input filters, the complexity of the regular sets in the rules and the number of rules in each neuron, and so on; compare with some optimal results as described in [27] We mention that in the present work we already have provided a first normal form, as forgetting rules and delays were not required for obtaining Turing-completeness.
- In [35] they mention that such the regular applicability sets allow for an oracle, though a "simple" and regular one. Thus it is of interest to simplify such regular sets in WSN P systems, e.g., what is the power of such systems with very restricted forms of regular sets as $\{a\}^+$ and finite sets, in the sense of [18,27].

– Creating homogeneous systems, as for example, in [10,11] is also worth to be investigated: with respect to the form of the input filters associated with the neurons, or the number of produced spikes; these may need to be heterogeneous for some rules, unlike previous works on homogeneous SN P systems where each neuron has the same rule set.
– In PSN P systems in [3] it is shown that two *polarisations* are enough, recalling that PSN P systems have no regular applicability sets.

Besides computing power, computing efficiency is also interesting to be considered with WSN P systems,for instance, how to solve **NP**-complete problems in a (non)uniform way, e.g., see [25].

Another interesting extension is the feature allowing for the creation of new neurons as in [45] or using the idea of pre-computed resources, e.g., see [21].

Real world applications can perhaps benefit from WSN P systems with neurons having the ability to “distinguish signals” using their associated input filters. Applications may include improvements on intrusion detection [19] and skeletonising images [40], with more directions and open problems discussed in [24,6].

We end the present work by highlighting, based on the ideas presented here, that the directed graph structure of an SN P system seems to be powerful, at least useful in programming the system. But with using finite input filters instead the loss of the static connection structure given by a directed graph, WSN P systems could be shown to be a powerful computing model, too. For proving Turing-completeness, we used a linear prime number representation for representing the contents of a register when simulating register machines, which seems to be a rather unconventional way to simulate register machines in the area of membrane systems. These ideas have also shown that the programming of WSN P systems is quite different and interesting in comparison with standard SN P systems and their many variants.

Acknowledgements F.G.C. Cabarle is supported by the QUAL21 008 USE project, “Plan Andaluz de Investigación, Desarrollo e Innovación” (PAIDI) 2020 and “Fondo Europeo de Desarrollo Regional” (FEDER) of the European Union, 2014-2020 funds. X. Zeng was supported by National Natural Science Foundation of China (Grant Nos. 62122025, U22A2037, 62250028)

References

1. Aboy, B.C.D., Bariring, E.J.A., Carandang, J.P., Cabarle, F.G.C., De La Cruz, R.T., Adorna, H.N., Martínez-del Amor, M.Á.: Optimizations in CuSNP simulator for spiking neural P systems on CUDA gpus. In: 2019 International Conference on High Performance Computing & Simulation (HPCS). pp. 535–542. IEEE (2019)
2. Adorna, H.N.: Matrix representations of spiking neural P systems: Revisited. arXiv preprint arXiv:2211.15156 (2022)
3. Alhazov, A., Freund, R., Ivanov, S.: Spiking neural P systems with polarizations– two polarizations are sufficient for universality. In: Bulletin of the International Membrane Computing Society. pp. 97–103. No. 1 (6 2016)

4. Alhazov, A., Freund, R., Oswald, M., Slavkovik, M.: Extended spiking neural P systems. In: Hoogeboom, H.J., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. pp. 123–134. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
5. Bibi, A., Xu, F., Adorna, H.N., Cabarle, F.G.C., et al.: Sequential spiking neural P systems with local scheduled synapses without delay. Complexity **2019** (2019)
6. Cabarle, F.G.C.: Thinking About Spiking Neural P Systems: Some Theories, Tools, and Research Topics. Journal of Membrane Computing **0** (2024). https://doi.org/10.1007/s41965-024-00147-y
7. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking neural P systems with structural plasticity. Neural Computing and Applications **26**, 1905–1917 (2015)
8. Cabarle, F.G.C., Buño, K.C., Adorna, H.N.: Time after time: notes on delays in spiking neural P systems. In: Theory and Practice of Computation: 2nd Workshop on Computation: Theory and Practice. Manila, The Philippines, September 2012, Proceedings. pp. 82–92. Springer (2013)
9. Chen, H., Freund, R., Ionescu, M., Păun, Gh., Pérez-Jiménez, M.J.: On string languages generated by spiking neural P systems. Fundamenta Informaticae **75**(1-4), 141–162 (2007)
10. De la Cruz, R.T.A., Cabarle, F.G.C., Adorna, H.N.: Steps toward a homogenization procedure for spiking neural P systems. Theoretical Computer Science **981**, 114250 (2024). https://doi.org/https://doi.org/10.1016/j.tcs.2023.114250
11. De la Cruz, R.T.A., Cabarle, F.G.C., Macababayao, I.C.H., Adorna, H.N., Zeng, X.: Homogeneous spiking neural P systems with structural plasticity. Journal of Membrane Computing **3**, 10–21 (2021)
12. Fan, S., Paul, P., Wu, T., Rong, H., Zhang, G.: On applications of spiking neural P systems. Applied Sciences **10**(20), 7011 (2020)
13. Freund, R., Ionescu, M., Oswald, M.: Extended spiking neural P systems with decaying spikes and/or total spiking. International Journal of Foundations of Computer Science **19**(05), 1223–1234 (2008)
14. Gulapa, M., Luzada, J.S., Cabarle, F.G.C., Adorna, H.N., Buño, K., Ko, D.: WebSnapse reloaded: The next-generation spiking neural P system visual simulator using client-server architecture. In: Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023). pp. 434–461. Atlantis Press (2024). https://doi.org/10.2991/978-94-6463-388-7_26
15. Gungon, R.V., Hernandez, K.K.M., Cabarle, F.G.C., De la Cruz, R.T.A., Adorna, H.N., Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I.: GPU implementation of evolving spiking neural P systems. Neurocomputing **503**, 140–161 (2022)
16. Hernández-Tello, J., Martínez-del Amor, M.A., Orellana-Martín, D., Cabarle, F.G.C.: Sparse spiking neural-like membrane systems on graphics processing units. International Journal of Neural Systems **34**(07), 2450038 (2024). https://doi.org/10.1142/S0129065724500382, pMID: 38755115
17. Hernández-Tello, J., Martínez-Del-Amor, M.A., Orellana-Martín, D., Cabarle, F.G.: Sparse matrix representation of spiking neuralsystems on GPUs. In: International Conference on Membrane Computing. pp. 316–322. Chengdu, China and Debrecen, Hungary (August 2021)
18. Ibarra, O.H., Woodworth, S., Yu, F., Păun, A.: On spiking neural p systems and partially blind counter machines. In: Calude, C.S., Dinneen, M.J., Păun, Gh., Rozenberg, G., Stepney, S. (eds.) Unconventional Computation. pp. 113–129. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

19. Idowu, R.K., Chandren, R., Othman, Z.A.: Advocating the use of fuzzy reasoning spiking neural P system in intrusion detection. In: Asian Conference on Membrane Computing ACMC 2014. pp. 1–5 (2014). https://doi.org/10.1109/ACMC.2014.7065804
20. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta informaticae **71**(2, 3), 279–308 (2006)
21. Ishdorj, T.O., Leporati, A.: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. Natural Computing **7**, 519–534 (2008)
22. Jimenez, Z.B., Cabarle, F.G.C., De la Cruz, R.T.A., Buño, K.C., Adorna, H.N., Hernandez, N.H.S., Zeng, X.: Matrix representation and simulation algorithm of spiking neural P systems with structural plasticity. Journal of Membrane Computing **1**, 145–160 (2019)
23. Ko, D., Cabarle, F.G.C., De La Cruz, R.T.: WebSnapse Tutorial: A Hands-On Approach for Web and Visual Simulations of Spiking Neural P Systems. In: Bulletin of the International Membrane Computing Society. vol. 16, pp. 137–153 (December 2023)
24. Leporati, A., Mauri, G., Zandron, C.: Spiking neural P systems: main ideas and results. Natural Computing **21**(4), 629–649 (2022)
25. Leporati, A., Mauri, G., Zandron, C., Păun, Gh., Pérez-Jiménez, M.J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. Natural computing **8**(4), 681–702 (2009)
26. Lloreda, C.L.: Wi-Fi for neurons: first map of wireless nerve signals unveiled in worms. Nature **623**(7989), 894–895 (2023)
27. Macababayao, I.C.H., Cabarle, F.G.C., De la Cruz, R.T.A., Zeng, X.: Normal forms for spiking neural P systems and some of its variants. Information Sciences **595**, 344–363 (2022)
28. Martínez-Del-Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural P systems with sparse matrix-vector operations. Processes **9**(4) (2021). https://doi.org/10.3390/pr9040690
29. Mateescu, A., Salomaa, A.: Handbook of formal languages, volume 1: Word, language, grammar, chapter formal languages: an introduction and a synopsis (1997)
30. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., USA (1967)
31. Odasco, A.N.L., Rey, M.L.M., Cabarle, F.G.C.: Improving GPU web simulations of spiking neuralsystems. Journal of Membrane Computing pp. 1–16 (2023). https://doi.org/https://doi.org/10.1007/s41965-023-00128-7
32. Orellana-Martín, D., Cabarle, F.G.C., Paul, P., Zeng, X., Freund, R.: Neurons on Wi-Fi. 20th Brainstorming Week on Membrane Computing and First Workshop on Virus Machines, January 24–26, 2024, Sevilla, Spain ((to appear))
33. Păun, A., Păun, Gh.: Small universal spiking neural P systems. BioSystems **90**(1), 48–60 (2007)
34. Păun, Gh.: Spiking neural P systems: A tutorial. Bull. Eur. Assoc. Theor. Comput. Sci. **91**, 145–159 (Feb 2007)
35. Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems. recent results, research topics. In: Algorithmic bioprocesses, pp. 273–291. Springer (2009)
36. Păun, Gh.: Membrane computing: an introduction. Springer Science & Business Media (2002)
37. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford Univeristy Press (2010)
38. Randi, F., Sharma, A.K., Dvali, S., Leifer, A.M.: Neural signal propagation atlas of caenorhabditis elegans. Nature pp. 1–9 (2023)

39. Ripoll-Sánchez, L., Watteyne, J., Sun, H., Fernandez, R., Taylor, S.R., Weinreb, A., Bentley, B.L., Hammarlund, M., Miller, D.M., Hobert, O., Beets, I., Vértes, P.E., Schafer, W.R.: The neuropeptidergic connectome of C. elegans. Neuron **111**(22), 3570–3589.e5 (2023). https://doi.org/https://doi.org/10.1016/j.neuron.2023.09.043
40. Song, T., Pang, S., Hao, S., Rodríguez-Patón, A., Zheng, P.: A parallel image skeletonizing method using spiking neural P systems with weights. Neural Processing Letters **50**, 1485–1502 (2019)
41. Song, X., Valencia-Cabrera, L., Peng, H., Wang, J.: Spiking neural P systems with autapses. Information Sciences **570**, 383–402 (2021)
42. Uy, A.V.D., Wu, J.J.Q.C., Cabarle, F.G.C., De la Cruz, R.T.A., Adorna, H.N.: Evolving spiking neural P systems with rules on synapses on CUDA. Philippine Computing Journal **17**, 10–31 (2022)
43. Verlan, S., Freund, R., Alhazov, A., Ivanov, S., Pan, L.: A formal framework for spiking neural P systems. Journal of Membrane Computing **2**(4), 355–368 (2020)
44. Wang, H., Qian, T., Zhao, Y., Zhuo, Y., Wu, C., Osakada, T., Chen, P., Chen, Z., Ren, H., Yan, Y., Geng, L., Fu, S., Mei, L., Li, G., Wu, L., Jiang, Y., Qian, W., Zhang, L., Peng, W., Xu, M., Hu, J., Jiang, M., Chen, L., Tang, C., Zhu, Y., Lin, D., Zhou, J.N., Li, Y.: A tool kit of highly selective and sensitive genetically encoded neuropeptide sensors. Science **382**(6672), eabq8173 (2023). https://doi.org/10.1126/science.abq8173
45. Wang, J., Hoogeboom, H.J., Pan, L.: Spiking neural P systems with neuron division. In: Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24–27, 2010. Revised Selected Papers. pp. 361–376. Springer (2011)
46. Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking neural P systems with polarizations. IEEE transactions on neural networks and learning systems **29**(8), 3349–3360 (2017)
47. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural P systems. In: Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers. pp. 377–391. Springer (2011)
48. WebSnapse page (2023), `https://aclab.dcs.upd.edu.ph/productions/software/websnapse`.