Spiking Neural P Systems with Weights

Jun Wang
junwangjf@gmail.com
Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China, and Leiden Institute of Advanced Computer Science, Universiteit Leiden, 2333 CA Leiden, Netherlands

Hendrik Jan Hoogeboom
hoogeboom@liacs.nl
Leiden Institute of Advanced Computer Science, Universiteit Leiden, 2333 CA Leiden, Netherlands

Linqiang Pan[*]
lqpan@us.es, lqpan@mail.hust.edu.cn
Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, Hubei, People's Republic of China, and Department of Computer Science and Artificial Intelligence, University of Sevilla, 41012, Sevilla, Spain

Gheorghe Păun
gpaun@us.es
Department of Computer Science and Artificial Intelligence, University of Sevilla, 41012, Sevilla, Spain, and Institute of Mathematics of the Romanian Academy, 014700 Bucureşti, Romania

Mario J. Pérez-Jiménez
marper@us.es
Department of Computer Science and Artificial Intelligence, University of Sevilla, 41012, Sevilla, Spain

A variant of spiking neural P systems with positive or negative weights on synapses is introduced, where the rules of a neuron fire when the potential of that neuron equals a given value. The involved values—weights, firing thresholds, potential consumed by each rule—can be real (computable) numbers, rational numbers, integers, and natural numbers. The power of the obtained systems is investigated. For instance, it is proved that integers (very restricted: 1, −1 for weights, 1 and 2 for firing thresholds, and as parameters in the rules) suffice for computing all

[*]Corresponding author

Turing computable sets of numbers in both the generative and the accepting modes. When only natural numbers are used, a characterization of the family of semilinear sets of numbers is obtained. It is shown that spiking neural P systems with weights can efficiently solve computationally hard problems in a nondeterministic way. Some open problems and suggestions for further research are formulated.

1 Introduction

Membrane computing is one of the recent branches of natural computing; it was initiated in Păun (2000) and has developed rapidly (as early as 2003, the Institute for Scientific Information considered membrane computing a "fast emerging research area in computer science"; see http://esi-topics.com). The aim is to abstract computing ideas (data structures, operations with data, ways to control operations, computing models) from the structure and the functioning of a single cell and from complexes of cells, such as tissues and organs, including the brain. The models are distributed and parallel computing devices, usually called *P systems*. There are three main classes of P systems: cell-like P systems (Păun, 2000), tissue-like P systems (Martin-Vide, Pazos, Păun, & Rodriguez-Patón, 2003), and neural-like P systems. Many variants of these systems have been considered. An overview of the field can be found in Păun (2002) and Păun, Rozenberg, and Salomaa (2010), with up-to-date information available at the membrane computing Web site (http://ppage.psystems.eu). For an introduction to membrane computing, one may consult Păun and Rozenberg (2002, 2010). This letter deals with a class of neural-like P systems called *spiking neural P systems* (SN P systems, for short), introduced in Ionescu, Păun, and Yokomori (2006).

SN P systems are a class of distributed and parallel computing models inspired by spiking neurons. As we know, spiking neurons are currently much investigated in neural computing (Gerstner & Kistler, 2002; Maass, 2002; Maass & Bishop, 1999). An SN P system consists of a set of neurons placed in the nodes of a directed graph, where neurons send signals (spikes, denoted by the symbol a in what follows) along synapses (arcs of the graph). Thus, the architecture of an SN P system is that of a tissue-like P system, with only one kind of object present in the cells. The objects evolve by means of spiking rules, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. In other words, a neuron containing k spikes, such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied. There are also forgetting rules, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are forgotten if the neuron contains exactly s spikes. The system works in a synchronized manner: in each time unit, the rule to be applied in each neuron is nondeterministically chosen, a chosen rule must be applied for each neuron with applicable rules, and the work of the system is sequential in each neuron: only (at most) one rule is applied in each neuron. One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, and the other moments are marked with 0. This binary sequence is called the spike train of the system; it might be infinite if the computation does not stop. Various numbers can be associated with a spike train, which can be considered as computed (or generated) by an SN P system. In this work, the result of a computation is encoded by the time span elapsed between the first two consecutive spikes sent into the environment by the (output neuron of the) system.

In SN P systems, the applicability of each rule is determined by checking the number of spikes in the neuron against a regular set associated with the rule. Considerable computational power is hidden in the implicit mechanism that SN P systems use to decide whether a given rule can be applied. For instance, it is proved that deciding whether a rule can be applied is at least NP-hard (Leporati, Zandron, Ferretti, and Mauri, 2007).

In this work, a variant of SN P systems is presented, with the goal of identifying an easy way to determine the applicability of rules. To this aim, we do not count spikes, as in usual SN P systems; instead, we consider that each neuron contains a potential, which generally can be expressed by a real number (to avoid any complication, for example, in building a hypercomputation device by using arbitrary real numbers; in what follows we always use computable real numbers). Each neuron fires when its potential is equal to a given firing value (we call it a threshold, because it reminds us of the firing threshold in neurobiology, although the two notions are not identical, as we will briefly note below); at that time, part of the potential is consumed, and a unit potential (a spike) is produced. This unit potential is passed to neighboring neurons and multiplied by the weights of synapses. The weights can also be real numbers—hence, both positive and negative. In this way, we can define computations and the result of computations as is usual in SN P systems (the result is associated with the spike train of the computation; here we consider the number of steps elapsed between the first two spikes, which leave the output neuron; SN P systems working in the accepting mode are also considered).

An important convention is assumed: when the potential of a neuron is higher than its firing threshold, the potential remains unchanged (it can be changed, that is, increased or decreased, by adding new potential from other neurons; the value of the potential can be positive or negative, depending on the weights of the synapses). But when the potential of a neuron is smaller than the firing threshold, this potential vanishes (i.e., the potential of the neuron is set to zero). These assumptions are essentially used in the proofs in this letter.

As we will see, SN P systems with integer values for weights and potentials are computationally universal, and the proofs are simpler than for usual SN P systems. Also, they use very small numbers—only $1, -1$ as weights and $1, 2$ as parameters in the rules.

Considering SN P systems with weights and firing thresholds also has a biological motivation. Like most of the other cells in the body, the plasma membrane of excitable cells exhibits a membrane potential (an electrical voltage difference across the membrane), called a *resting membrane potential*, and its typical value is −70 mV. Moreover, each neuron has its own threshold potential, which is the membrane potential. A membrane must be depolarized to initiate an action potential. If the membrane potential of a neuron reaches or exceeds its threshold potential, the neuron will fire, sending out an action potential (signal), and its membrane potential will return to the resting membrane potential. If the membrane potential is smaller than the threshold potential, no signal is emitted and the membrane potential will also return to the resting membrane potential. (For more details, see Gerstner & Kistler, 2002, and Maass & Bishop, 1999.)

We stress here that this work does not intend to propose a platform for modeling biological processes. Actually the initial goal of membrane computing was to bring something useful to (theoretical) computer science from biology, although many applications of P systems were reported later, especially in modeling biological and biomedical processes. Spiking neural P systems are currently a subject of investigation for theoretical computer science, without any claim concerning their possible relevance for biological modeling. In particular, the notion of a threshold used here is borrowed from the real neurons, but this notion is used in a slightly different way: as in biology, a neuron fires when its membrane potential is equal to the threshold; potentials lower than the threshold vanish. However, unlike the biological fact, a neuron does not fire in an SN P system when its membrane potential exceeds the threshold.

The letter organized as follows. In section 2, some necessary prerequisites are introduced. The computation model investigated in the letter, spiking neural P systems with weights, is defined in section 3. An example spiking neural P system with weights is given in section 4. Some preliminary results are given in section 5. In section 6, we prove that spiking neural P systems with integers as weights are universal. In section 7, the family of semilinear sets is characterized by spiking neural P systems with natural numbers as weights. The efficiency of spiking neural P systems with weights is investigated in section 8. Final remarks are presented in section 9.

2 Prerequisites

It is useful for readers to have some familiarity with basic elements of language theory (e.g., from Rozenberg & Salomaa, 1997), as well as basic membrane computing (Păun, 2002). A quick introduction to membrane computing can be found in Păun and Rozenberg (2002). (For more updated information about membrane computing, refer to http://ppage.psystems.eu.) We introduce here most of the necessary prerequisites; the definitions in the subsequent sections are self-contained.

By $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}_c$ we denote the sets of natural, integer, rational, and computable real numbers, respectively. For an alphabet, V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, we write simply a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$.

A regular expression over an alphabet V is defined as follows: λ and each $a \in V$ is a regular expression; if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V; and nothing else is a regular expression over V. With each regular expression E, we associate a language $L(E)$, defined in the following way: $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, and $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Unnecessary parentheses can be omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ can also be written as E^*.

By *SLIN, NRE* we denote the families of semilinear and Turing computable sets of numbers. (*SLIN* is the family of length sets of regular languages, that is, languages characterized by regular expressions, and *NRE* is the family of length sets of recursively enumerable languages, that is, those recognized by Turing machines.)

In the university proofs, we use the notion of a register machine, which is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction `HALT`), and R is the set of instructions. Each label from H labels only one instruction from R, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k).
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is nonzero, then subtract 1 from it, and go to the instruction with label l_j; otherwise, go to the instruction with label l_k).
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes (generates) a number n in the following way. We start with all registers empty (i.e., storing the number 0), apply the instruction with label l_0, and proceed to apply instructions as indicated by labels (and, in the case of SUB instructions, by the content of registers). If we reach the halt instruction, the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers that are Turing computable, hence they characterize *NRE* (see, e.g., Minsky, 1967).

Without loss of generality, we may assume that l_0 labels an ADD instruction and that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation (we only add to its content).

We can also use a register machine in the accepting mode. A number is stored in the first register (all other registers are empty). If the computation starting in this configuration eventually halts, the number is accepted. Again, all sets of numbers in *NRE* can be obtained, even using deterministic register machines, that is, with the ADD instructions of the form $l_i : (\mathtt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\mathtt{ADD}(r), l_j)$).

We use the following convention. When comparing the power of two number generating or accepting devices, D_1, D_2, the number 0 is ignored, that is, we write $N(D_1) = N(D_2)$ if and only if $N(D_1) - \{0\} = N(D_2) - \{0\}$ (this corresponds to the usual practice of ignoring the empty string in language and automata theory).

3 Spiking Neural P Systems with Weights

In this section, we introduce the type of SN P systems investigated in this letter. The definition is complete, but familiarity with the basic elements of classic SN P systems (e.g., from Păun & Pérez-Jiménez, 2008) is helpful.

An SN P system with weights (from now on, we deal only with such systems and call them *SN P systems* for brevity; when it is necessary to stress the new type of systems, we use *WSN P systems*), of degree $m \geq 1$, is a construct of the form

$$\Pi = (\sigma_1, \ldots, \sigma_m, syn, in, out),$$

where:

- $\sigma_1, \ldots, \sigma_m$ are neurons, of the form $\sigma_i = (p_i, R_i), 1 \leq i \leq m,$ where (1) $p_i \in \mathbb{R}_c$ is the initial potential in neuron σ_i and (2) R_i is a finite set of spiking rules of the from $T_i/d_s \to 1$, $s = 1, 2, \ldots, n_i$ for some $n_i \geq 1$, where $T_i \in \mathbb{R}_c$, $T_i \geq 1$, is the firing threshold potential of neuron σ_i, and $d_s \in \mathbb{R}_c$ with the restriction $0 < d_s \leq T_i$.
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \times \mathbb{R}_c$ are synapses between neurons, where $i \neq j$, $r \neq 0$ for each $(i, j, r) \in syn$, and for each $(i, j) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ there is at most one synapse (i, j, r) in syn.
- $in, out \in \{1, 2, \ldots, m\}$ indicate the input and output neurons, respectively.

The spiking rules are applied as follows. Assume that at a given moment, neuron σ_i has a potential equal to p. If $p = T_i$, then any rule $T_i/d_s \to 1 \in R_i$ can be applied. The execution of this rule consumes an amount of d_s of the potential (thus, the potential becomes $T_i - d_s$) and produces one unit potential (we also say a spike) to be delivered to all the neurons σ_j such that $(i, j, r) \in syn$. Specifically, each of these neurons σ_j receives a quantity of potential equal to r, which is added to the existing potential in σ_j. Note that

r can be positive or negative; hence, the potential of the receiving neuron is increased or decreased. The potential emitted by a neuron σ_i passes immediately to all neurons σ_j such that $(i, j, r) \in syn$, that is, the transition of potential takes no time. If a neuron σ_i spikes and it has no outgoing synapse, then the potential emitted by neuron σ_i is lost.

We stress that (1) each neuron σ_i has only one fixed threshold potential T_i; (2) if a neuron has the potential equal to its threshold potential, then all rules associated with this neuron are enabled, and only one of them is nondeterministically chosen to be applied; and (3) when a neuron spikes, there is always only one unit potential emitted.

If neuron σ_i has a potential p such that $p < T_i$, then the neuron σ_i returns to the resting potential 0. If neuron σ_i has a potential p such that $p > T_i$, the potential p remains unchanged.

To sum up, if neuron σ_i has potential p and receives potential k at step t, then at step $t + 1$, it has the potential p', where

$$p' = \begin{cases} k, & \text{if } p < T_i; \\ p - d_s + k, & \text{if } p = T_i \text{ and rule } T_i/d_s \to 1 \text{ is applied;} \\ p + k, & \text{if } p > T_i. \end{cases}$$

As usual in membrane computing, a global clock is assumed, marking the time for the whole system; hence, the functioning of the system is synchronized. Each neuron uses at most one rule in each step, nondeterministically chosen among its rules, provided that its potential equals the firing threshold, but all neurons that have applicable rules must choose and apply a rule.

The configuration of the system is described by the distribution of potentials in neurons. The initial configuration of the system is the tuple $\langle p_1, \dots, p_m \rangle$. Applying the rules as suggested above, we can define transitions among configurations. Any sequence of transitions starting from the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be used. With any computation, halting or not, we associate a spike train, the binary sequence with occurrences of 1 indicating time instances when the output neuron sends one unit potential (a spike) out of the system (we also say that the system itself spikes at that time).

The result of a computation can be defined in several ways. In this letter, with any spike train containing at least two spikes, the first two being emitted at step t_1, t_2, one associates a result in the form of the number $t_2 - t_1$; we say that this number is computed by Π. The set of all numbers computed in this way by Π is denoted by $N_2(\Pi)$ (the subscript indicates that we consider only the distance between the first two spikes of any computation; note that 0 cannot be computed, which is why we disregard this number when investigating the computing power of any device).

SN P systems can also work in the accepting mode. We start the computation from the initial configuration and introduce in the input neuron two spikes, in steps t_1 and t_2 (hence, we introduce in σ_{in} one unit of potential in each step t_1 and t_2); the number $t_2 - t_1$ is accepted by the system if the computation eventually halts. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π.

When dealing with decidability problems, as in section 8, various values of potential are input into σ_{in}, at precise time units, instead of single spikes; such a sequence of potentials is used as an encoding of the (instance of the) problem to solve.

In the generative case, the neuron with label in is ignored. In the accepting mode, the neuron with label out is ignored. For simplification, occasionally, "neuron i" is used to denote "neuron σ_i"; that is, when we say "neuron i," we mean a reference to "neuron σ_i."

We denote by $N_\alpha W_X SNP_m$ the families of all sets $N_\alpha(\Pi)$, $\alpha \in \{2, acc\}$, computed by WSN P systems with at most $m \geq 1$ neurons, using weights, thresholds, and amounts of consumed potentials in the rules taken from the set X, for $X \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}_c\}$. When the number of neurons is not bounded, the subscript m is replaced with $*$.

Usually, in the area of SN P systems, one takes into account several other parameters describing the size of the used systems, such as the maximal number of rules in a neuron and the maximal number of spikes consumed by a rule. Here we can also consider the maximal firing threshold, the maximal positive weight, and the minimal negative weight of a synapse. However, as we will see in the following sections, these parameters will have very small values in all results we obtain, so we prefer to simplify the notation and ignore these parameters.

4 An Example of WSN P Systems

In order to show the function of WSN P systems, we give an SN P system with rational numbers as weights.

Consider the SN P system Π shown in Figure 1, which consists of three neurons denoted by rounded rectangles with the initial potential and spiking rules inside. Arrows between these rounded rectangles represent the synapses; these arrows are marked with numbers indicating the weights. The input neuron has an incoming arrow, and the output neuron has an outgoing arrow, suggesting their communication with the environment. When the weight on a synapse is 1, it is omitted.

At step 1, only output neuron σ_{out} spikes, while the other two neurons σ_1, σ_2 maintain their potentials, because their potentials are greater than their corresponding firing thresholds. Neurons σ_1 and σ_2 receive potentials -1.5 and -1, respectively, from neuron σ_{out}. At step 2, neurons σ_1 and σ_2 have potentials 1.5 and 1, respectively, which equal their corresponding firing thresholds; hence, both neurons σ_1 and σ_2 spike.

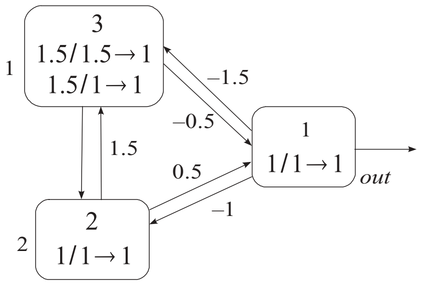

Figure 1: Example of a WSN P system Π.

When neuron σ_2 spikes, it consumes one unit of potential and at the same time receives one unit of potential from neuron σ_1; hence, at the next step, it still has potential 1 and spikes again. Neuron σ_1 has two rules, $1.5/1.5 \to 1$ and $1.5/1 \to 1$, and one of them is nondeterministically chosen. If rule $1.5/1.5 \to 1$ is applied, then by consuming potential 1.5 and receiving potential 1.5 from neuron σ_2, the potential of neuron σ_1 is still 1.5; hence, it will spike again. In this way, neurons σ_1 and σ_2 can spike as long as rule $1.5/1.5 \to 1$ is chosen to be applied. At each step during this process, neuron σ_{out} receives potential -0.5 from σ_1 and potential 0.5 from σ_2, which means that neuron σ_{out} has potential 0 and does not spike.

If at step $t \geq 2$, rule $1.5/1 \to 1$ is chosen to be applied, then at step $t+1$, neuron σ_1 has the potential $1.5 - 1 + 1.5 = 2$, which is greater than its threshold, and it will not spike. At step $t+1$, neuron σ_2 has potential 1 and spikes; neuron σ_{out} receives potential 1 from neuron σ_2 at step $t+1$ and spikes at step $t+2$. At step $t+2$, neuron σ_1 receives potential -1.5 from neuron σ_{out}; its potential changes to $2 - 1.5 = 0.5$, which is less than its threshold potential 1.5; and the neuron returns to resting potential 0 at step $t+3$. At step $t+2$, neuron σ_2 receives potential -1 from neuron σ_{out}; its potential changes to $0 - 1 = -1$, which is less than its threshold potential 1; and it returns to resting potential 0 at step $t+3$. So the system halts after step $t+3$.

The number generated by system Π is $(t+2)+1 = t+3$, where $t \geq 2$, and the value of t depends on the nondeterministic choice of rules $1.5/1.5 \to 1$ or $1.5/1 \to 1$ in neuron σ_1. Thus, $N_2(\Pi) = \mathbb{N} - \{1, 2\}$ (recall that the number 0 is ignored when investigating the computational power of devices).

5 Preliminary Results

We start by noting some immediate relations, following from the definitions:

Lemma 1. *(i)* $N_\alpha W_{\mathbb{N}} SNP_m \subseteq N_\alpha W_{\mathbb{Z}} SNP_m \subseteq N_\alpha W_{\mathbb{Q}} SNP_m \subseteq N_\alpha W_{\mathbb{R}_c} SNP_m \subseteq NRE$, *for all* $\alpha \in \{2, acc\}$ *and* $m \geq 1$ *or* $m = *$. *(ii)* $N_\alpha W_X SNP_m \subseteq$

$N_\alpha W_X SNP_{m'} \subseteq N_\alpha W_X SNP_*$, for all $\alpha \in \{2, acc\}$, $m' \geq m \geq 1$, and $X \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}_c\}$.

All relations are obvious, with the inclusion of $N_\alpha W_{\mathbb{R}_c} SNP_m \subseteq NRE$ being a consequence of the fact that we use only computable real numbers and the Turing-Church thesis.

For a given WSN P system $\Pi = (\sigma_1, \ldots, \sigma_m, syn, in, out)$ and a constant $k \in \mathbb{R}_c$, let us denote by $k\Pi$ the system obtained by multiplying by k all weights, thresholds, and potentials from Π. (If a rule of Π is of the form $T_i/d_s \to 1$, then in $k\Pi$, we use the rule $kT_i/kd_s \to 1$; a synapse (i, j, r) will become (i, j, kr) in $k\Pi$, hence transporting a potential equal to kr when neuron σ_i produces one spike.)

Lemma 2. *For any WSN P system Π and constant $k \in \mathbb{R}_c$, with $k\Pi$ constructed as above, we have $N_\alpha(\Pi) = N_\alpha(k\Pi)$, for all $\alpha \in \{2, acc\}$.*

The assertion directly follows from the way $k\Pi$ is defined and has the next interesting consequence:

Corollary 1. *$N_\alpha W_{\mathbb{Z}} SNP_m = N_\alpha W_{\mathbb{Q}} SNP_m$ for all $\alpha \in \{2, acc\}$ and $m \geq 1$ or $m = *$.*

One inclusion is pointed out in lemma 1, and the opposite one follows from lemma 2: take an arbitrary WSN P system with all constants in $\mathbb{Q}$, let k be the least common multiple of all denominators of rational numbers in Π (weights, thresholds, and potentials), and consider $k\Pi$. The system $k\Pi$ has all integers used, so it is equivalent to Π.

6 Universality of WSN P Systems with Integers

In this section, WSN P systems with integers are proved to be universal in both the generative and the accepting case. However, these results cannot be obtained as particular cases of universality results known for usual SN P systems: the weights bring additional possibilities to "program" the computation of an SN P system, but instead we are very much restricted by the fact that all the rules of a neuron are enabled at the same time, when the firing threshold is reached. This corresponds to typical SN P systems with a finite number of spikes in each neuron (and only one regular expression—identifying a singleton), and such systems are known to compute only semilinear sets of numbers (Ionescu et al., 2006).

6.1 The Generative Case. Consider first the case of sets $N_2(\Pi)$. We have the following result:

Theorem 1. *$N_2 W_{\mathbb{Z}} SNP_* = NRE$.*

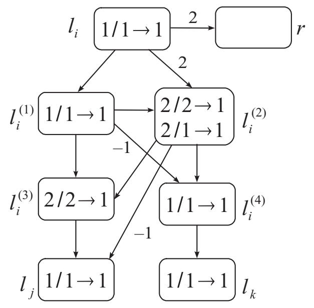

Figure 2: Module ADD (simulating $l_i : (\mathtt{ADD}(r), l_j, l_k)$).

Proof. We have only to prove the inclusion $NRE \subseteq N_2W_{\mathbb{Z}}SNP_*$. To this aim, we use the characterization of NRE by means of register machines used in the generating mode.

Let us consider a register machine $M = (m, H, l_0, l_h, R)$ as introduced in section 2. It is assumed that in the halting configuration, all registers are empty except the first register and that output register is never decremented during the computation. We construct a WSN P system Π to simulate M as follows. We construct modules ADD and SUB to simulate the instructions of M, as well as an output module FIN, which provides the result (in the form of a suitable spike train). Each register r of M will have a neuron σ_r in Π, and if the register contains the number n, the associated neuron will have the potential $2n + 2$. A neuron σ_{l_i} is associated with each label $l_i \in H$, and some auxiliary neurons $\sigma_{l_i^{(j)}}$, $j = 1, 2, 3, \ldots$, will also be considered (each $l_i \in H$ is associated with a unique instruction of M; hence all neurons σ_{l_i}, $\sigma_{l_i^{(j)}}$ are precisely associated with a unique instruction of M).

The modules will be given in graphic form. In the initial configuration, all neurons have the potential 0, except that the neuron associated with the label l_0 of M has potential 1 and the neurons associated with the registers have potential 2. In general, when a neuron σ_{l_i}, where $l_i \in H$, has potential 1, the neuron becomes active, and the module associated with the respective instruction of M starts to work, simulating the instruction.

Module ADD: Simulating an ADD Instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$. Module ADD, shown in Figure 2, is composed of eight neurons: neuron σ_r for register r; neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ for instructions with labels l_i, l_j, l_k; and four auxiliary neurons.

The initial instruction of M, the one with label l_0, is an ADD instruction. Let us assume that at step t, we have to simulate an instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$, with neuron σ_{l_i} having potential 1 and other neurons having resting potential 0, except those neurons associated with registers. Having potential 1, neuron σ_{l_i} fires by rule $1/1 \to 1$. Simultaneously, neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r receive potentials 1, 2, 2, respectively. In this way, the potential of neuron σ_r increased by two, thus simulating the increase of the number stored in register r by one.

At the next step, the computation of M passes nondeterministically to one of the instructions with labels l_j and l_k; that is, we have to ensure the firing of neurons σ_{l_j} or σ_{l_k} in system Π, nondeterministically choosing one of them. To this aim, we use the nondeterministic choice of rules $2/2 \to 1$ and $2/1 \to 1$ in $\sigma_{l_i^{(2)}}$. Because neuron $\sigma_{l_i^{(2)}}$ has potential 2 (received from neuron σ_{l_i} at the last step), it has to choose nondeterministically one of these rules. We have two cases:

1. If rule $2/2 \to 1$ is applied at step $t+1$, then neuron $\sigma_{l_i^{(2)}}$ consumes its potential for spiking. Which receiving potential 1 from neuron $\sigma_{l_i^{(1)}}$ at step $t+1$, neuron $\sigma_{l_i^{(2)}}$ has potential 1 at step $t+2$, which is less than its threshold potential; hence, the neuron returns to the resting potential 0. At step $t+1$, neuron σ_{l_j} receives potential -1 from neuron $\sigma_{l_i^{(2)}}$, which is less than its firing threshold, and it returns to the resting potential 0 at step $t+2$. At step $t+1$, neuron $\sigma_{l_i^{(4)}}$ receives potential -1 from neuron $\sigma_{l_i^{(1)}}$ and potential 1 from neuron $\sigma_{l_i^{(2)}}$; hence, its potential is still 0. At step $t+1$, neuron $\sigma_{l_i^{(3)}}$ receives potential 2 from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$, and at step $t+2$ it spikes by rule $2/2 \to 1$. Receiving potential 1 from neuron $\sigma_{l_i^{(3)}}$, neuron σ_{l_j} becomes active and starts to simulate the instruction l_j of M.
2. If rule $2/1 \to 1$ is applied at step $t+1$, then neuron $\sigma_{l_i^{(2)}}$ consumes one unit of its potential for spiking. When receiving potential 1 from neuron $\sigma_{l_i^{(1)}}$ at step $t+1$, neuron $\sigma_{l_i^{(2)}}$ still has potential 2 at step $t+2$ and spikes again. At step $t+1$, neuron σ_{l_j} receives potential -1 from neuron $\sigma_{l_i^{(2)}}$, which is less than its threshold potential, and returns to the resting potential 0 at step $t+2$. At step $t+2$, neuron σ_{l_j} receives potential -1 from neuron $\sigma_{l_i^{(2)}}$ and potential 1 from neuron $\sigma_{l_i^{(3)}}$, so its potential is 0. At step $t+1$, neuron $\sigma_{l_i^{(4)}}$ receives potential -1 from neuron $\sigma_{l_i^{(1)}}$ and potential 1 from neuron $\sigma_{l_i^{(2)}}$, so its potential remains 0. At step $t+2$, neuron $\sigma_{l_i^{(4)}}$ receives one unit of potential from neuron $\sigma_{l_i^{(2)}}$ and it spikes at step $t+3$. Receiving potential 1 from neuron $\sigma_{l_i^{(4)}}$ at step $t+3$, neuron σ_{l_k} becomes active and starts to simulate the instruction l_k of M.

Therefore, from firing neuron σ_{l_i}, the system nondeterministically fires one of neurons σ_{l_j}, σ_{l_k}, which correctly simulates the ADD instruction l_i : $(\mathtt{ADD}(r), l_j, l_k)$.

Module SUB: Simulating a SUB Instruction $l_i : (\mathit{SUB}(r), l_j, l_k)$. Module SUB, shown in Figure 3, is composed of seven neurons: neuron σ_r for register r; neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ for instructions with labels l_i, l_j, l_k; and three auxiliary neurons $\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}, \sigma_{l_i^{(3)}}$.

Instruction l_i is simulated in Π in the following way. Initially, neuron σ_{l_i} has potential 1, and other neurons have potential 0, except neurons

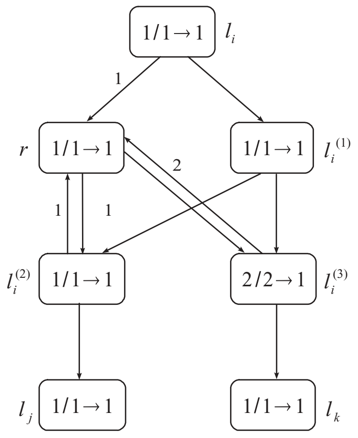

Figure 3: Module SUB (simulating instruction l_i : ($\texttt{SUB}(r), l_j, l_k$).

associated with registers. Let t be the moment when neuron σ_{l_i} fires. At step t, neurons $\sigma_{l_i^{(1)}}$ and σ_r receive potentials 1 and -1, respectively. At step $t+1$, neuron $\sigma_{l_i^{(1)}}$ fires, and neurons $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ receive potential 1 from neuron $\sigma_{l_i^{(1)}}$. For neuron σ_r, there are the following two cases:

1. The potential of neuron σ_r is 2 at step t (i.e., the number stored in register r is 0). Then, at step $t+1$, neuron σ_r has potential 1 (it has received potential -1 from neuron σ_{l_i} at the previous step), and it spikes by rule $1/1 \to 1$. At step $t+1$, neuron $\sigma_{l_i^{(2)}}$ receives potential -1 from neuron σ_r and potential 1 from neuron $\sigma_{l_i^{(1)}}$, so it has potential 0. At step $t+1$, neuron $\sigma_{l_i^{(3)}}$ receives potential 2 (one unit of potential from neuron σ_r and another one from neuron $\sigma_{l_i^{(1)}}$), and it spikes at step $t+2$. Receiving potential 1 from neuron $\sigma_{l_i^{(3)}}$, neuron σ_{l_k} becomes active and starts to simulate instruction l_k of M. Note that at step $t+2$, neuron σ_r receives potential 2 from neuron $\sigma_{l_i^{(3)}}$, and in this way, it correctly ends with potential 2, which corresponds to the fact that the number stored in register r is 0.
2. The potential of neuron σ_r is $2n+2$ ($n > 0$) at step t. At step $t+1$, neuron σ_r has potential $2n+1$, which is greater than its threshold, and will remain unchanged. At step $t+1$, neuron $\sigma_{l_i^{(3)}}$ receives potential 1 from neuron $\sigma_{l_i^{(1)}}$, which is less than its threshold potential; hence, it will not spike and has potential 0 at step $t+2$. At step $t+1$, neuron $\sigma_{l_i^{(2)}}$ receives potential 1 from neuron $\sigma_{l_i^{(1)}}$, and it spikes at step $t+2$. Receiving potential -1 from neuron $\sigma_{l_i^{(2)}}$ at step $t+2$, the potential of neuron σ_r changes to $2n$, and in this way, it simulates that the number stored in register r is decreased by one. Receiving potential 1 from neuron $\sigma_{l_i^{(2)}}$ at step $t+2$, neuron σ_{l_j} becomes active, and starts to simulate the instruction l_j of M.

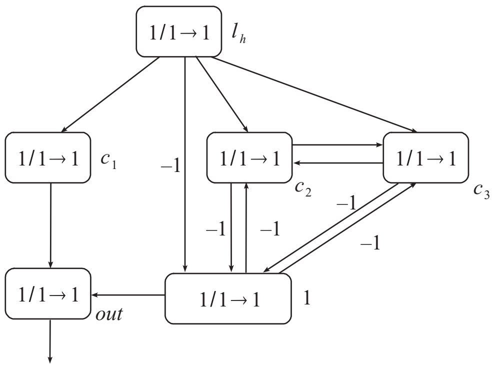

Figure 4: Module FIN: Outputting the result of computation.

The simulation of SUB instruction is correct: we started from σ_{l_i} and ended in σ_{l_j} (if the register r is not empty and decreased by one) or in σ_{l_k} (if the register is empty).

Note that there is no interference between neurons used in the ADD and the SUB modules other than correctly firing the neurons σ_{l_j} or σ_{l_k}, which may label instructions of the other kind. However, it is possible to have interference between neurons in two SUB modules. Specifically, if there are several SUB instructions l_t that act on register r, then neurons $\sigma_{l_t^{(2)}}$ and $\sigma_{l_t^{(3)}}$ receive potentials -1 and 1 from neuron σ_r, respectively, while simulating the instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$. After receiving these potentials, neurons $\sigma_{l_t^{(2)}}$ and $\sigma_{l_t^{(3)}}$ have potentials that are less than their corresponding firing thresholds, so both return to resting potential 0 at the next step. Consequently, the interference among SUB modules will not cause undesired steps in Π (i.e., steps that do not correspond to correct simulations of instruction of M).

Module FIN: Outputting the Result of the Computation. Module FIN is shown in Figure 4. Assume that the computation in M halts, which means that the halting instruction is reached. This means that neuron σ_{l_h} in Π has potential 1 and fires by rule $1/1 \to 1$. At that moment, neuron σ_1 has potential $2n+2$, where $n \geq 1$ is the number stored in register 1 of M. When σ_{l_h} fires, each neuron $\sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}$ receives potential 1; neuron σ_1 receives potential -1, changing its potential to $2n+1$. Suppose that this is step t. At step $t+1$, neuron σ_{c_1} spikes; neuron σ_{out} receives potential 1 from neuron neuron σ_{c_1} and spikes at step $t+2$ (this is the first spike sent out by system Π).

From step $t+1$ on, consuming one unit potential, neurons σ_{c_2} and σ_{c_3} send potential 1 to each other, and this process continues until they receive potential -1 from neuron σ_1. During this process, at each step, neuron σ_1 receives potential -1 from neuron σ_{c_2} and -1 from σ_{c_3}, which corresponds to decreasing by 1 the value of the register 1. At step $t+n+1$, neuron σ_1 has potential 1 and spikes; neurons σ_{c_2} and σ_{c_3} have potential 0 after receiving potential -1 from neuron σ_1. Receiving potential 1 from σ_1 at step $t+n+1$, neuron σ_{out} spikes again at step $t+n+2$, and the system sends the second spike to the environment. The interval between these two spikes sent out by the system is $(t+n+2)-(t+2)=n$, which is exactly the number stored in register 1 of M at the moment when the computation of M halts.

Note that after system Π sends out the second spike, all neurons in Π have potential 0 except that neurons σ_i, $2 \leq i \leq m$, have potential 2. For mathematical elegance, we can return the potentials of neurons σ_i, $2 \leq i \leq m$) to 0 when the computation of Π halts. To this end, we need to add synapses $(out, i, -2)$ $(i = 2, 3, \dots, m)$ in system Π: when σ_{out} spikes for the first time, the potential it sends to each σ_i, $2 \leq i \leq m$, is equal to -2; hence, it cancels the potential of the target neuron.

If the number stored in register 1 is 0 when register machine M halts, then at step $t+1$, neuron σ_{out} has potential 2, which is greater than its threshold potential 1. In this case, neuron σ_{out} is blocked, and system Π sends no spike to the environment. Remember that 0 is ignored when we investigate the power of our systems

From this description of the modules and their work, it is clear that the register machine M is correctly simulated by the system Π. Therefore, $N_2(\Pi) = N(M)$, and this completes the proof.

Let us now examine the weights used in the previous proof. In the ADD module, we have two synapses with weight 2. This value can be avoided, so that the module uses only weights 1 and -1 in the following way. Consider two new neurons, say σ_a and σ_b, intermediate between σ_{l_i} and its neighboring neurons (specifically, with synapses $(l_i, a, 1), (l_i, b, 1), (a, r, 1), (b, r, 1), (a, l_i^{(1)}, 1), (a, l_i^{(2)}, 1), (b, l_i^{(2)}, 1)$. Each of the new neurons holds the rule $1/1 \to 1$. In this way, both σ_r and $\sigma_{l_i^{(2)}}$ get two potential units, two steps after activating σ_{l_i}, simultaneously with one potential unit coming to $\sigma_{l_i^{(1)}}$. From now on, the work of the module continues as described above. The same trick can be used in the SUB module in order to remove the single synapse with weight 2, noting that no synchronization problem appears; hence, the synapse is removed, and two intermediate neurons are introduced, similar to σ_a, σ_b above, between $\sigma_{l_i^{(3)}}$ and σ_r. Module FIN contains only synapses with weights 1 and -1.

Consequently, the universality is obtained with WSN P systems of a rather restricted form. This observation may be be formulated as a normal form result:

Corollary 2. *The universality of WSN P systems is preserved if we use only (i) weights 1 and -1 for synapses, (ii) at most two rules per neuron, and (iii) all rules are of one of the following three forms: $1/1 \to 1$, $2/2 \to 1$, and $2/1 \to 1$.*

The use of two rules in at least one neuron cannot be avoided because in the generative case, the system should be nondeterministic. If the system is

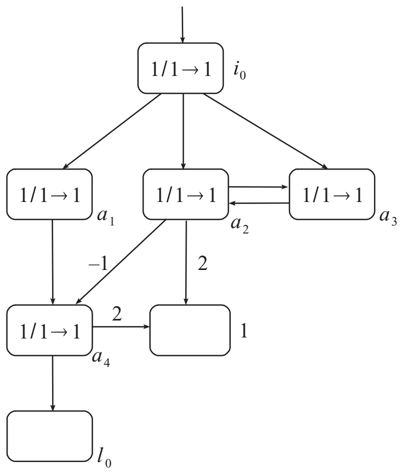

Figure 5: Module INPUT: Initializing the computation.

deterministic, the system generates nothing or a singleton. Nondeterminism originates from choosing between rules; hence, at least two rules are needed in the same neuron.

6.2 The Accepting Case. The number of rules per neuron can be decreased to one in this case due to the fact that the ADD instructions of a register machine used in the accepting mode can be assumed to be deterministic. The construction follows the same ideas as the one from the proof of theorem 1.

The number to be computed is introduced in the system as the distance between the first two unit potentials that the input neuron receives. This is done by means of a module INPUT, as indicated in Figure 5. The first unit potential entering the neuron σ_{i_0} of the module passes to neurons a_1, a_2, a_3. The last two neurons will feed each other until one further unit potential is received from σ_{i_0} (the system stops at this moment). In the time span when the two spikes enter the system, neuron σ_1 receives two potential units in each time unit. Note that neuron σ_{a_4} does not spike this time: either it receives potential from both σ_{a_1} and σ_{a_2} and they cancel each other, or only from σ_{a_2}, which is under the threshold of σ_{a_4}, and it is reset to 0. However, when the second spike enters the system, because σ_{a_2} does not spike, σ_{a_4} receives only one potential unit, from σ_{a_1}; hence it fires, adding two potential units to σ_1 and activating σ_{l_0} by sending one potential unit to it. Note that σ_1 contains $2n + 2$ units of potential, exactly what is needed in view of the construction in the proof of theorem 1. If the system halts, the number n is accepted, so we do not need a module for outputting the result of computation. For a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$, we use the

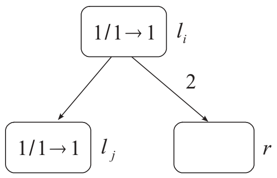

Figure 6: Module ADD in the deterministic case.

same module as in Figure 3. For a deterministic ADD instruction of the form $l_i : (\mathtt{ADD}(r), l_j)$, we consider the simple module given in Figure 6. Initially all neurons in all these modules have the potential 0, with the exception of neurons $\sigma_i, 2 \leq i \leq m$ (associated with the registers), which contain two potential units. The way the modules work can be checked in a similar way as in the proof of theorem 1.

We conclude with the following counterpart of theorem 1 (actually, the assertions in corollary 2 also hold true, with point ii stating that exactly one rule is used in each neuron; removal of synapses with weight 2 is done in the same way as described above):

Theorem 2. $N_{acc}W_{\mathbb{Z}}SNP_* = NRE$.

The previous results can be summarized as follows:

Corollary 3. $N_\alpha W_{\mathbb{N}}SNP_m \subseteq N_\alpha W_{\mathbb{Z}}SNP_m = N_\alpha W_{\mathbb{Q}}SNP_m = N_\alpha W_{\mathbb{R}_c}SNP_m =$ *NRE, for all* $\alpha \in \{2, acc\}$ *and* $m \geq 1$ *or* $m = *$.

7 Systems with Natural Numbers as Weights

The only case that has remained unsettled concerns systems with natural numbers as weights, thresholds, and potentials. Somewhat surprisingly at first sight, such systems characterize the family of semilinear sets. For the proof of this assertion, we use a series of lemmas.

Lemma 3. *Every finite set of natural numbers is in the family* $N_2W_{\mathbb{N}}SNP_*$.

Proof. Let us take a finite set of natural numbers, $U = \{n_1, n_2, \ldots, n_k\}$, all of them different from 0. We construct a WSN P system as suggested in Figure 7. Specifically, for each number n_i we have a "subsystem" composed of neurons $\sigma_{(i,a)}, \sigma_{(i,b)}, \sigma_{(i,c)}, \sigma_{(i,0)}, \sigma_{(i,1)}, \ldots, \sigma_{(i,n_i+1)}$, with synapses, rules, and initial potentials as indicated in Figure 7. A synapse exists from neuron $\sigma_{(i,n_i+1)}$ to the output neuron σ_{out}. There also exists one further neuron, σ_0, for which only two synapses exist: $((1,0),0,1)$ and $(0, out, 1)$. Figure 7 shows

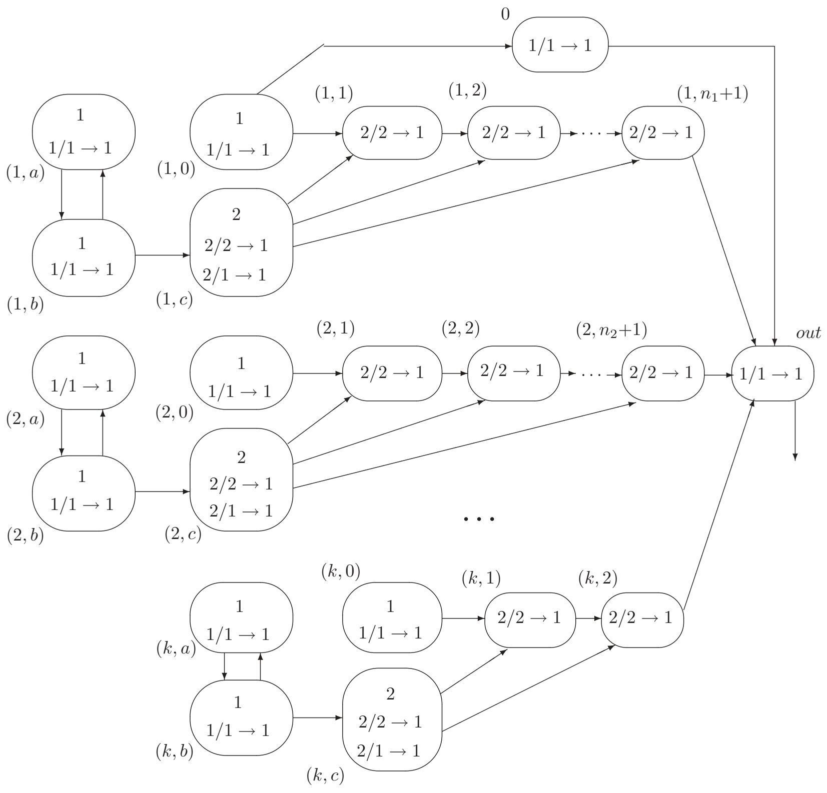

Figure 7: A WSN P system generating a finite set of numbers.

only two generic subsystems and the subsystem that helps in generating number $n_k = 1$.

This system works as follows. All neurons behave deterministically, except $\sigma_{(i,c)}$, for each $1 \leq i \leq k$. As long as such a neuron uses its second rule, $2/1 \to 1$, it can spike again in the next step. One potential unit remains inside, and a further one is received from $\sigma_{(i,b)}$; hence, the initial amount, equal to the firing threshold, is restored. In turn, as long as $\sigma_{(i,c)}$ spikes, the sequence of neurons $\sigma_{(i,1)}, \dots, \sigma_{(i,n_i+1)}$ continues to work, moving to the right toward σ_{out}, the neuron that can fire. A neuron in this sequence must receive two spikes in order to fire: one from the preceding neuron (initially $\sigma_{(i,0)}$ fires, because it has inside one potential unit) and one from $\sigma_{(i,c)}$. Note that each time unit, $\sigma_{(i,c)}$ is fed with one potential unit by $\sigma_{(i,b)}$, which works perpetually in cooperation with $\sigma_{(i,a)}$. (They can be stopped, for example, when σ_{out} spikes, by sending them a further spike, but this detail is not important for our result. It can be important when the output is defined differently, for instance, if halting is relevant.) If all neurons in the sequence

$\sigma_{(i,1)}, \ldots, \sigma_{(i,n_i+1)}$ work, then in step $n_i + 2$, a spike is set to σ_{out}, and in step $n_i + 3$, a spike is sent to the environment. Note, however, that the first spike is sent out of the system in step 3, on the path $\sigma_{(1,0)}, \sigma_0, \sigma_{out}$. Consequently, the distance between these spikes is $(n_i + 3) - 3 = n_i$.

However, any of these processes of sending a spike toward σ_{out} along the path $\sigma_{(i,1)}, \ldots, \sigma_{(i,n_i+1)}$ can be stopped at any step after the first one by applying rule $2/2 \to 1$ in neuron $\sigma_{(i,c)}$. This rule consumes both potential units of $\sigma_{(i,c)}$; only one unit is received from $\sigma_{(i,b)}$, which is removed; and $\sigma_{(i,c)}$ remains idle. Therefore, nondeterministically, we can stop all but one sequence $\sigma_{(i,1)}, \ldots, \sigma_{(i,n_i+1)}$ of neurons, $1 \leq i \leq k$, so that the output neuron receives only two spikes: the one in step 3, along the path $\sigma_{(1,0)}, \sigma_0, \sigma_{out}$, and the one along the path $\sigma_{(i,1)}, \ldots, \sigma_{(i,n_i+1)}$, which remain unblocked. In conclusion, each number in the set $\{n_1, n_2, \ldots, n_k\}$ can be generated, that is, $N_2(\Pi) = U$; hence $FIN \subseteq N_2WT_{\mathbb{N}}SNP_*$.

The number of neurons depends on the sum of numbers in the set U, but some neurons in the previous construction can be saved (a unique pair $\sigma_{(i,a)}, \sigma_{(i,b)}$ can feed up all neurons $\sigma_{(i,c)}$), but this aspect is not relevant for us.

Lemma 4. *Any arithmetical progression $P_{k,l} = \{kn + l \mid n \geq 1\}$ with $k \geq 2$, $l \geq 2$ is in the family $N_2W_{\mathbb{N}}SNP_*$.*

Proof. Let us consider system Π in Figure 8. It generates the set $N_2(\Pi) = \{2n + 2 \mid n \geq 1\}$.

The output neuron spikes in the first step and then only after receiving a spike from neuron σ_4. In turn, this neuron spikes only after receiving a spike from each neuron σ_2 and σ_3 (if we have only one spike in σ_4, it is removed). Then neuron σ_2 can send a spike to σ_4 simultaneously with σ_3 only if it, after receiving two spikes from σ_0 (note that the synapse $(0, 2, 2)$ is the only one with weight 2), first uses rule $2/1 \to 1$, so that one spike remains inside, making the firing possible in the next step. If neuron σ_2 uses rule $2/2 \to 1$, its spike will reinitiate the work of neuron σ_0, and the spikes from σ_2 (received from σ_1) and, after one step from σ_4, are removed. Thus, the output neuron fires for the second time after a number of passings through the cycle $\sigma_0, \sigma_1, \sigma_0$ (which means two steps), then ending the computation, which needs two further steps. The precise checking of the functioning of the system in Figure 8 is left to the reader.

Thus, we can generate the arithmetical progression $P_{2,2}$. If we want to generate a progression $P_{k,l}$ with $k = 2 + i$ and $l = 2 + j$, we add i neurons between σ_2 and σ_0, (instead of the synapse $(2, 0, 1)$) and j neurons between σ_4 and σ_{out}, arranged in a sequence, with rule $1/1 \to 1$ in each of them. These neurons will lengthen the cycle $\sigma_0, \sigma_2, \ldots, \sigma_0$ with further i steps and the path from σ_4 to σ_{out} with j steps. We denote by $\Pi_{i,j}$ the resulting system. We have $N_2(\Pi_{i,j}) = \{(2 + i)n + (2 + j) \mid n \geq 1\} = P_{k,l}$.

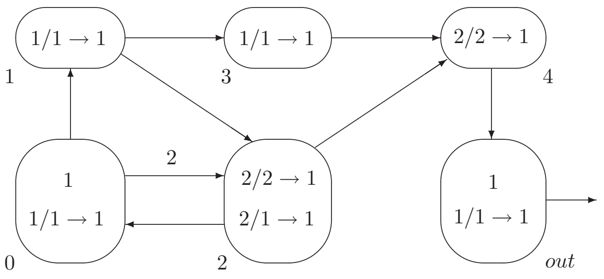

Figure 8: A WSN P system generating an infinite set of numbers.

Lemma 5. *If* $\Pi_1, \ldots, \Pi_n$ *are WSN P systems with natural numbers as weights and potentials, and for each* $1 \le i \le n$*, there is* $T_j \ge 1$ *such that all computations in* Π_i *spike for the first time at the same step* T_i*, then* $\bigcup_{i=1}^{n} N_2(\Pi_i) \in N_2 W_{\mathbb{N}} SNP_*$.

Proof. Let us take separately neurons $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_{out}$ from Figure 8, with two spikes present in σ_2 and one in σ_1 from the beginning; also change the label of σ_{out}, for instance, to σ_5, without having here any spike in the initial configuration. This system behaves like a trigger: it sends, or not, a spike out of σ_5, depending on the nondeterministic behavior of σ_2.

Consider now a finite set of WSN P systems $\Pi_1, \ldots, \Pi_n$ as in the statement of the lemma. Let T be the maximum of all $T_i, 1 \le i \le n$. From the output neuron of each Π_i we consider a chain of additional $T - T_i$ neurons with the unique rule $1/1 \to 1$, ending with a new output neuron. Regardless of the length of this chain, the set of numbers generated by Π_i remains the same, as the first two spikes leaving the system remain at the same distance in time; they leave the system only later. Moreover, in this way, all modified systems spike for the first time at step T.

Take a trigger as above for each of the modified systems Π_i (we continue to denote them by Π_i). Assume that a neuron σ_s from some Π_i contains $r_s \ge 1$ spikes in the initial configuration of Π_i. We remove these spikes from σ_s and establish a synapse $(5, s, r_s)$. In this way, when the neuron σ_5 of the trigger spikes, the system Π is loaded with exactly as many spikes as it contained initially.

In this way, nondeterministically, the triggers will load one or more of the systems $\Pi_i, 1 \le i \le n$. Now take an additional neuron that will be considered the output neuron of the whole system—let us call it σ_f—and connect all output neurons of systems Π_i to it. With a delay of one step, the spike train of each $\Pi_i, 1 \le i \le n$, is produced by the new system. It is important that all systems Π_i spike for the first time at the same moment. Only one of the triggers has to activate a system Π_i. All other systems Π_j, $j \ne i$, should remain idle, without any spike inside; otherwise the system produces no output. Indeed, if two spikes arrive at the same time in neuron σ_f (this is the case if two systems Π_i, Π_j were activated), then σ_f is blocked forever, since its potential is higher than its firing threshold.

Note that it is crucial here that a trigger behaves nondeterministically; it sends, or not, a spike out of his neuron σ_5.

Consequently, the system whose construction was suggested above generates the union of sets $N_2(\Pi_i)$, $1 \le i \le n$.

Theorem 3. $N_2 W_{\mathbb{N}} SNP_* = SLIN$.

Proof. (i) In order to obtain the inclusion $SLIN \subseteq N_2 W_{\mathbb{N}} SNP_*$ we use the known fact that any semilinear set is a finite union of a finite set with a finite number of arithmetical progressions. From the previous lemmas, we know that finite sets and arithmetical progressions of the form $P_{k,l}$ with $k \ge 2$ and $l \ge 2$ are in $N_2 W_{\mathbb{N}} SNP_*$. Let us note that the systems constructed in the proofs of lemmas 3 and 4 have the property in the statement of lemma 5 to have all computations spiking for the first time at the same step. What remains to show is that arithmetical progressions that are not of the form $P_{k,l}$ with $k \ge 2$ and $l \ge 2$ are also in $N_2 W_{\mathbb{N}} SNP_*$.

Such progressions are $P_{2,1}$, $P_{2,0}$, and $P_{1,l}$ for all $l \ge 0$. However, we have

$$P_{2,1} = \{3\} \cup P_{2,3}, \quad P_{2,0} = \{2\} \cup P_{2,2},$$

Consequently, with lemmas 4 and 5, they belong to $N_2 W_{\mathbb{N}} SNP_*$. Moreover,

$$P_{1,l} = (P_{1,l} \cap \{1, 2, 3\}) \cup (P_{1,l} \cap P_{2,2}) \cup (P_{1,l} \cap P_{2,3}).$$

Let $l_1 = \min(P_{1,l} \cap P_{2,2})$ and $l_2 = \min(P_{1,l} \cap P_{2,3})$. (Note that $l_1 \ge 4$ and $l_2 \ge 5$.) Then we have

$$(P_{1,l} \cap P_{2,2}) = \{l_1\} \cup P_{2,l_1}, \quad (P_{1,l} \cap P_{2,3}) = \{l_2\} \cup P_{2,l_2}.$$

Using again lemmas 4 and 5, we get $L_{1,l} \in N_2 W_{\mathbb{N}} SNP_*$, and this completes the proof of the inclusion $SLIN \subseteq N_2 W_{\mathbb{N}} SNP_*$.

(ii) The inclusion $N_2 W_{\mathbb{N}} SNP_* \subseteq SLIN$ is somewhat straightforward, after making the observation that because all weights are positive, the potential accumulated in a neuron can be decreased only if it is smaller than or equal to the firing threshold of that neuron. Otherwise stated, if a neuron σ_i accumulates a potential strictly larger than T_i, the potential remains larger than T_i forever (and no rule can be applied in σ_i). Therefore, the configurations of a system $\Pi = (\sigma_1, \dots, \sigma_m, syn, out)$ can be described by a vector $\langle \alpha_1, \dots, \alpha_m \rangle$ where $\alpha_i \in \{0, 1, \dots, T_i\} \cup \{\bar{T}\}$, where $\bar{T}$ is just a symbol indicating that the potential of σ_i is strictly greater than T_i. If new amounts of potential are brought to a neuron whose content is already described by $\bar{T}$, the same symbol will describe that neuron at the next step. In this way, the functioning of the system can be described by a finite state device, for example, by a regular (actually, right-linear, because we also need rules producing no terminal symbol) grammar. We start from the initial configuration (its description is the axiom of the grammar), and to each transition, we associate a rule. Because we have finitely many configurations, we have finitely many rules. As long as no spike is sent to the environment, no terminal is produced. When the first spike exits the system, the reached nonterminal is marked, and from now on, we produce a terminal symbol in each step (and we carry on the marking of nonterminals). When a second spike is produced by the output neuron, the derivation stops, and we no longer introduce a nonterminal. The number of terminals produced is exactly the number generated by the system. The formal details are left to the reader.

A similar result is expected for the case when WSN P systems with natural numbers are used in the accepting mode.

8 Efficiency of WSN P Systems

The trigger subsystem mentioned at the beginning of the proof of lemma 5 suggests the possibility of using WSN P systems in order to solve computationally hard problems nondeterministically. This possibility is explored below and illustrated with two well-known **NP**-complete problems: `Subset Sum and SAT`.

8.1 Time Complexity for Nondeterministic SN P Systems.

We first recall some basic definitions concerning the solution of decision problems by means of SN P systems (Leporati, Mauri, Zandron, Păun, & Pérez-Jiménez, 2009):

Definition 1. *Let $X = (I_X, \theta_X)$ be a decision problem, where I_X is the set of instances and θ_X is a total Boolean function over I_X. Let $p : \mathbb{N} \to \mathbb{N}$ be a computable function. We say that X is solvable by a family $\Pi = (\Pi(n))_{n\in\mathbb{N}}$ of SN P systems, in time bounded by p, in a nondeterministic and uniform way, if the following holds:*

- *The family Π is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine working in polynomial time with respect to n, which constructs the SN P system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *There exist polynomial time computable functions, cod and s, over I_X, such that:*
    - *For each instance $w \in I_X$, $s(w)$ is a natural number, and $cod(w)$ is a sequence of integers, representing potentials, which enter one by one the input neuron of $\Pi(s(w))$ in the first steps of the computation (the computation starts when the first digit, negative, zero, or positive, of $cod(w)$ enters the system).*

- *The family Π is p-bounded with respect to (X, cod, s); that is, for each instance $w \in I_X$, the minimum length of an accepting computation of $\Pi(s(w))$ with input $cod(w)$ is bounded by $p(|w|)$; a computation is accepting if during the output neuron of $\Pi(s(w))$ it fires, hence it sends some potential units out of the system.*
- *The family Π is sound and complete with respect to (X, cod, s); that is, for every $w \in I_X$, $\theta_X(w) = 1$ if and only if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$.*

We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\Pi = (\Pi(n))_{n\in\mathbb{N}}$ of SN P systems, in a nondeterministic and uniform way, if X is solvable by the family Π in time bounded by a polynomial, in a nondeterministic and uniform way.

Note that, as usual when dealing with the nondeterministic computations of deciding problems, the computations are considered accepting if they send out a signal, whereas the rejecting computations (i.e., computations that never fire the output neuron of the system) are ignored.

Definition 2. *Let $X = (I_X, \theta_X)$ be a decision problem and $p : \mathbb{N} \to \mathbb{N}$ a computable function. We say that X is solvable by a family $\Pi = (\Pi(w))_{w\in I_X}$ of SN P systems, in time bounded by p, in a nondeterministic and semiuniform way, if the following holds:*

- *The family Π is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine working in polynomial time that constructs the SN P system $\Pi(w)$ from the instance $w \in I_X$.*
- *The family Π is p-bounded with respect to X; that is, for each $w \in I_X$, the minimum length of an accepting computation of $\Pi(w)$ is bounded by $p(|w|)$.*
- *The family Π is sound and complete with respect to X; that is, for every $w \in I_X$, $\theta_X(w) = 1$ if and only if there exists an accepting computation of $\Pi(w)$.*

We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\Pi = (\Pi(w))_{w\in I_X}$ of SN P systems, in a nondeterministic and semiuniform way, if X is solvable by the family Π in time bounded by a polynomial, in a nondeterministic and semi-uniform way.

Some informal explanations are provided here for readers unfamiliar with computational complexity. Problems can be solved by deterministic (in each step, at most one continuation is possible) or nondeterministic devices from a given class: a solving "machine" can be constructed in a uniform way (starting from the problem itself) or in a nonuniform way (starting from the specific instance to solve). In all cases, we distinguish the precomputation (building the "machine" starting from the problem or from the instance to solve) from the computation (the work of the "machine," after receiving as input the code of the instance, for solving it). The precomputation is in general done by a Turing machine, and its duration must not be "too long"; that is why one requests that the computation time is at most polynomial time with respect to the length of the problem instance. The computation time is the main goal of the investigation, and this can be parallel time in the case of distributed and parallel devices—which is also the case for SN P systems.

The soundness property requires that if we obtain an acceptance response from the system (associated with an instance) through some computation, then the answer of the problem (for that instance) is affirmative. The completeness property means that if the answer to that instance of the problem is positive, there exists an accepting computation of the system (associated with that instance).

8.2 Semiuniform Solution to `Subset Sum`. Let us start by recalling the NP-complete problem `Subset Sum` (Garey & Johnson, 1979).

Problem 1

NAME: `Subset Sum`
INSTANCE: A (multi)set $V = \{v_1, v_2, \ldots, v_n\}$ of positive integer numbers, and a positive integer number S.
QUESTION: Is there a sub(multi)set $B \subseteq V$ such that $\sum_{b \in B} b = S$?

Let us consider the WSN P system depicted in Figure 9. Initially neurons $\sigma_{d_i}, \sigma_{e_i}, \sigma_{out}$ contain the resting potential 0; neurons σ_{a_i} have potential 1; neurons σ_{b_i} have potential 1; neurons σ_{c_i} have potential 2; and neurons σ_i have potential $v_i + 1$, for $i = 1, 2, \ldots, n$. The system solves the `Subset Sum` problem nondeterministically, where the nondeterminism originates from the choice of rules $2/2 \to 1$ and $2/1 \to 1$ in neurons σ_{c_i}.

At step 1, all neurons $\sigma_{a_i}, \sigma_{b_i}, \sigma_{c_i}$ spike. Receiving two unit potentials from neurons σ_{a_i} and σ_{c_i}, neuron σ_{d_i} spikes at step 2, sending one unit potential to neuron σ_{e_1}. For neuron σ_{c_i}, we have two cases:

1. If in neuron σ_{c_i}, at step 1, rule $2/2 \to 1$ is applied, then the initial two-unit potential is consumed; at the same time, neuron σ_{c_i} receives one unit potential from neuron σ_{b_i}, which is less than its threshold, and vanishes to the resting potential 0 at the next step without sending any potential out. In this case, neuron σ_{e_i} receives only one unit potential from σ_{d_i}, which is less than its threshold, and returns to the resting potential 0, without firing. Neuron σ_i does not fire, and this corresponds to the case when the number v_i does not appear in the corresponding subset.
2. If in neuron σ_{c_i}, at step 1, rule $2/1 \to 1$ is used, one initial unit potential is consumed; at the same time, neuron σ_{c_i} receives one unit potential from neuron σ_{b_i}. So with potential 2, neuron σ_{c_i} spikes again at step

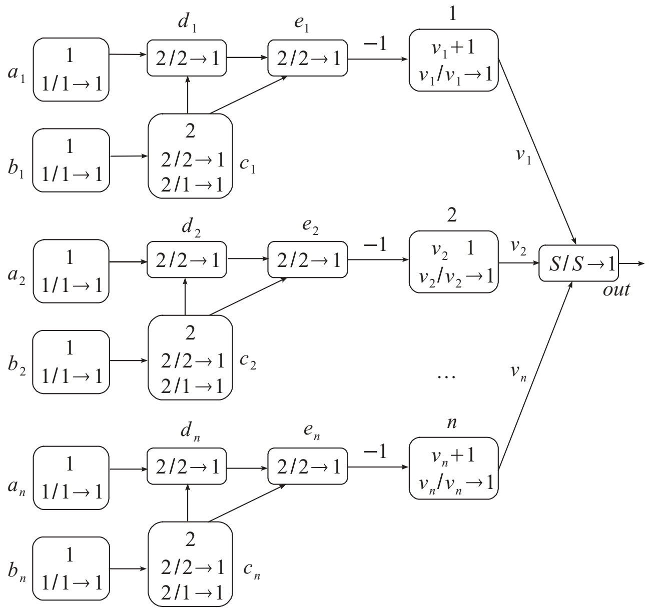

Figure 9: A WSN P system solving the `Subset Sum` problem.

2, sending one unit potential to neuron σ_{e_i}. In this case, having potential 2, neuron σ_{e_i} spikes. Neuron σ_i receives potential -1 by synapse $(e_i, i, -1)$, and its potential becomes v_i. With potential v_i, neuron σ_i spikes, sending one unit potential to neuron σ_{out}. This corresponds to selecting the number v_i in a subset (whose total sum remains to be compared with S).

Neuron σ_{out} checks whether the sum of the chosen numbers equals S. If this is the case, one unit potential is sent to the environment by rule $S/S \to 1$. Hence, the instance of the problem encoded in the system, by means of the initial potential in neurons $\sigma_1, \sigma_2, \dots, \sigma_n$ as well as of the rules of these neurons (and also of σ_{out}, which "knows" the value of S), has a solution if and only if there is a computation that spikes in step 4.

Although the system is rather simple, it is nonuniformly constructed; the instance is encoded in the system. As we see, the subsystems that consist of neurons $\sigma_{a_i}, \sigma_{b_i}, \sigma_{c_i}, \sigma_{d_i}, \sigma_{e_i}$ work for nondeterministically choosing some numbers among $v_1, v_2, \dots, v_n$. They are independent of the instance of the problem and necessary for the nondeterministic behavior of the system.

In the instance of the problem, the numbers $v_1, v_2, \ldots, v_n$ and S are all positive integers. From the solution, it is not difficult to see that the systems also work for real numbers. This is a new aspect in membrane computing: dealing with real numbers.

It remains open how to construct WSN P systems that solve `Subset Sum` in a uniform way.

8.3 Uniform Solution to `SAT`. Let us now move on to `SAT`, one of the most popular **NP**-complete problems. The instances of `SAT` depend on two parameters: the number of variables n and the number of clauses m. As a consequence, `SAT` will be uniformly solved by means of a family $(\Pi(\langle n, m\rangle))_{n,m\in\mathbb{N}}$ of WSN P systems, where $\langle n, m\rangle$ is a primitive recursive and bijective function from $\mathbb{N}^2$ to $\mathbb{N}$: $\langle n, m\rangle = ((m+n)(m+n+1)/2) + m$, and $\Pi(\langle n, m\rangle)$ solves all the instances composed by m clauses, built using n variables.

We recall that a clause is a disjunction of literals, occurrences of x_i or $\neg x_i$, built on a given set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables. In what follows, we will require that no repetitions of the same literal may occur in any clause; in this way, a clause can be seen as a subset of all possible literals. A truth assignment of the variables $x_1, x_2, \ldots, x_n$ is a mapping $a : X \to \{0, 1\}$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that a truth assignment satisfies the clause C if, after assigning the truth values to all the variables that occur in C, the evaluation of C (considered as a Boolean formula) gives 1 (true) as a result.

Problem 2

NAME: `SAT`
INSTANCE: A set $C = \{C_1, C_2, \ldots, C_m\}$ of clauses, built on a finite set $\{x_1, x_2, \ldots, x_n\}$ of Boolean variables
QUESTION: Is there a truth assignment of the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C?

Associated with such an instance, it is customary to consider the propositional formula $\gamma = C_1 \wedge C_2 \wedge \cdots \wedge C_M$, with each clause being a disjunction of literals; this is said to be a propositional formula in the conjunctive normal form.

In what follows, we show that a family of WSN P systems can be constructed in a uniform manner that solves `SAT` in a number of steps that is linear with respect to $\max\{n, m\}$, working nondeterministically, where n is the number of variables, and m is the number of clauses. The general structure of such system is given in Figure 10. Several modules appear in this figure, which will be explained below.

Because the construction is uniform, we need a way to encode a given instance of `SAT` . Let us consider a propositional formula in the conjunctive

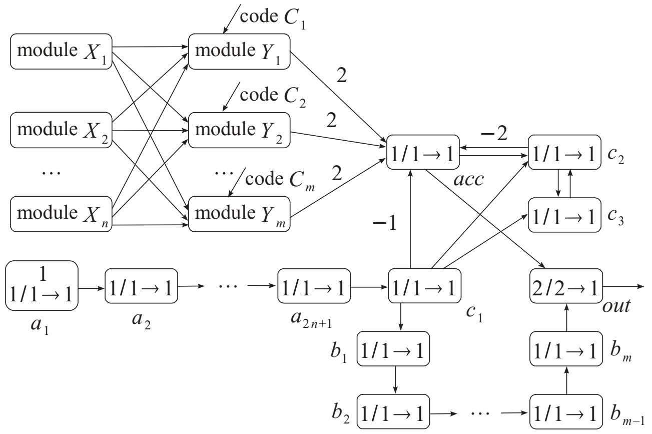

Figure 10: The structure of WSN P systems solving the `SAT` problem.

normal form, $\gamma = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Clause C_j is encoded as $code(C_j) = 00\alpha_{j,1}(-2)\alpha_{j,2}(-2)\cdots(-2)\alpha_{j,n}$, where

$$\alpha_{j,i} = \begin{cases} 0, & \text{if } x_i \text{ and } \neg x_i \text{ do not appear in } C_j \\ 1, & \text{if } x_i \text{ appears in } C_j \\ 2, & \text{if } \neg x_i \text{ appears in } C_j \end{cases}$$

for $i = 1, 2, \ldots, n$. Actually, we consider m input neurons (one for each clause), and in each of them, we introduce a sequence of $2n + 2$ digits $-2, 0, 1, 2$ (with a corresponding potential sent inside at each step), where 00 at the first two positions of the sequence is used as a delay, number -2 separates each consecutive pair of $\alpha_{j,i}$ and $\alpha_{j,i+1}$, and numbers $0, 1, 2$ describe the situation of each variable $x_1, x_2, \ldots, x_n$ in the corresponding clause.

For instance, for the formula $\gamma = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$, we have two input neurons—the first one receiving the potential sequence $code(C_1) = 001(-2)1(-2)0$, and the second one receiving the potential sequence $code(C_2) = 001(-2)2(-2)1$. In general, it takes $2n + 2$ steps to introduce the input for a formula with n variables.

A module X_i (shown in Figure 11) exists for each variable x_i, $1 \le i \le n$, and a module Y_j (shown in Figure 12) is associated with each clause C_j, $1 \le j \le m$. Each module X_i has a synapse going to each module Y_j.

These modules nondeterministically produce a truth assignment for the variables $x_1, \ldots, x_n$, using the same idea as in the case of `Subset Sum` : the nondeterministic choice between rules $2/2 \to 1$ and $2/1 \to 1$ in neuron $\sigma_{d_{i,5}}$. If rule $2/2 \to 1$ is used, neuron $\sigma_{d_{i,3}}$ will not spike. If rule $2/1 \to 1$ is used, then neuron $\sigma_{d_{i,3}}$ spikes, and this unit potential moves to neuron $\sigma_{f_{j,1}}$ in module Y_j along the path $\sigma_{e_{i,1}}, \ldots, \sigma_{e_{i,2(i-1)}}, \sigma_{f_{j,1}}$.

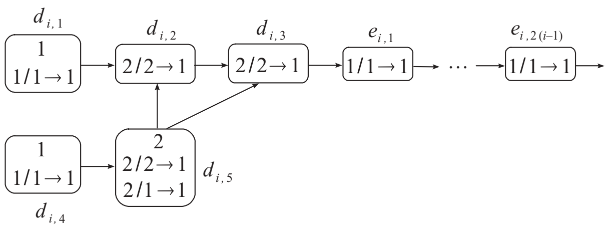

Figure 11: Module X_i.

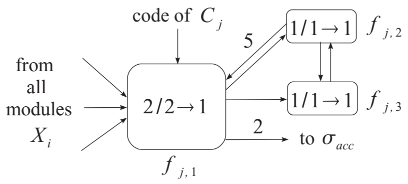

Figure 12: Module Y_j.

In order to synchronize the check performed in neurons $\sigma_{f_{j,1}}$ (i.e., to bring here the truth assignment of variable x_i in the moment when the code describing the possible occurrence of x_i in C_j arrives in this neuron), we put 00 at the beginning of the input sequence and use the delaying neurons $\sigma_{e_{i,1}}, \ldots, \sigma_{e_{i,2(i-1)}}$. No such neurons appear in module X_1; two neurons $\sigma_{e_{2,1}}, \sigma_{e_{2,2}}$ appear in X_2; and in general, $2(i-1)$ neurons $\sigma_{e_{i,1}}, \ldots, \sigma_{e_{i,2(i-1)}}$ appear in module X_i. In this way, a delay of $2(i-1)$ steps is enforced, thus ensuring the synchronization: at steps $2i+1$, all neurons $\sigma_{f_{j,1}}$ receive both the truth assignment of x_i and the code of x_i related with C_j.

As one can see from the previous explanations, at steps $3, 5, \ldots, 2n+1$, neurons $\sigma_{f_{j,1}}$, $1 \leq j \leq m$, receive a number of spikes as follows:

0, if $x_i = 0$ and $x_i, \neg x_i$ do not appear in C_j
1, if $x_i = 1$ and $x_i, \neg x_i$ do not appear in C_j
1, if $x_i = 0$ and x_i appears in C_j
2, if $x_i = 1$ and x_i appears in C_j
2, if $x_i = 0$ and $\neg x_i$ appears in C_j
3, if $x_i = 1$ and $\neg x_i$ appears in C_j

Thus, the rule in $\sigma_{f_{j,1}}$ is enabled and produces one unit potential only in the case when the clause C_j becomes true for the corresponding truth assignment of variable x_i. This unit potential reaches both the neuron σ_{acc} and the "flooding neurons" $\sigma_{f_{j,2}}, \sigma_{f_{j,3}}$. From now on, at each step, neurons $\sigma_{f_{j,2}}$ and $\sigma_{f_{j,3}}$ send one unit potential to each other, and neuron $\sigma_{f_{j,1}}$ receives potential 5 from $\sigma_{f_{j,2}}$. In this way, the potential of neuron $\sigma_{f_{j,1}}$ keeps more than 2 and does not spike anymore, which ensures that neuron σ_{acc} receives at most potential 2 from each module Y_j, only if clause C_j has been satisfied. Consequently, the potential of neuron σ_{acc} reaches $2m$ (at step $2n+1$) only if the truth assignment produced nondeterministically by modules X_i satisfies formula γ.

It should be noted that before neuron $\sigma_{f_{j,1}}$ spikes, at each step $2i+1$, neuron $\sigma_{f_{j,1}}$ receives at most potential 3. If its potential equals 2, it spikes; after that moment, it keeps a potential greater than 3 and does not spike again. If its potential does not equal 2, then receiving potential -2 (corresponding to the digit -2 in the input sequences) at the next step, it returns to the resting potential 0. So the check performed in neurons $\sigma_{f_{j,1}}$ processes in the order of $x_1, x_2, \ldots, x_n$ (first x_1, then x_2, and so on), and the check process corresponding to each variable does not interfere with each other.

From step $2n+2$ on, we check whether the potential of neuron σ_{acc} equals $2m$. Suppose that the potential of neuron σ_{acc} equals $2l$ at step $2n+1$. The unit potential in neuron σ_{a_1} arrives in neuron σ_{c_1} at step $2n+1$, and neuron σ_{c_1} spikes at step $2n+2$. Neuron σ_{b_1} receives one unit potential from neuron σ_{c_1}, and this unit potential moves along the path $\sigma_{b_1}, \ldots, \sigma_{b_m}, \sigma_{out}$, arriving in neuron σ_{out} at step $2n+m+2$. At step $2n+2$, neuron σ_{acc} receives potential -1; from step $2n+2$ on, at each step, it receives potential -2 from neuron σ_{c_2}. Hence, neuron σ_{acc} has potential 1 at step $2n+l+1$ and spikes at step $2n+l+2$. Neuron σ_{out} receives one unit potential from σ_{b_m} at step $2n+m+2$ and one unit potential from σ_{acc} at step $2n+l+2$. If $l \neq m$, then these two unit potentials do not arrive in neuron σ_{out} at the same time and they vanish, and the potential of σ_{out} is always less than 2, hence neuron σ_{out} does not spike. If $l = m$, these two potential units meet in neuron σ_{out} at step $2n+m+2$, and neuron σ_{out} spikes at step $2n+m+3$. Therefore, the system spikes only if the truth assignment produced nondeterministically by modules X_i satisfies the formula γ.

9 Conclusion

We have introduced a variant of SN P systems with weighted synapses, where the potentials of neurons are processed by spiking rules, and the applicability of spiking rules is under the control of given firing thresholds. WSN P systems are universal if integers are used to represent the values of these parameters—weights, potentials and thresholds. When natural numbers are used for weights, potentials, and thresholds, a characterization of semilinear sets of natural numbers is obtained.

We also investigated the possibility of using WSN P systems as a framework for solving computationally hard problems. A semiuniform solution to `Subset Sum` problem is given, which works in constant time. A uniform solution to the `SAT` problem is presented, which is linear with respect to $\max\{n, m\}$, where n is the number of variables and m is the number of clauses. Note that the systems solving `Subset Sum` and `SAT` work in a nondeterministic way.

Several issues about WSN P systems remain to be clarified. In this work, noncomputable real numbers are ignored; their effect on the functioning and the computing power of WSN P systems remains an open question.

The result of a computation is encoded by the number of computation steps (i.e., time elapsed) between the first two spikes in the spike train produced by the system. If the result of a computation is associated with the potential of the output neuron in the halting configuration (in this case, the system computes real numbers, a rather new concept in membrane computing), it is of interest to investigate what results can be obtained. In particular, it deserves to be investigated whether the feature of computing real numbers is useful for applications of SN P systems in learning and pattern recognition. This suggests a question formulated in other contexts (see, e.g., Păun, 2007): bring problems and techniques from the neural computing area to the SN P systems area; we know of no attempt in this respect.

In the definition of WSN P systems and in the proofs, the following fact is assumed and essentially used: if the potential of a neuron is strictly smaller than its firing threshold, then it vanishes (it is reset to 0). If this resetting does not hold but the potential remains as is, in the same way as a potential greater than the threshold remains unmodified, it is open as to what results can be obtained. The same question holds for when a neuron fires when its potential is higher than the threshold, thus getting closer to the situation in biology. The decaying of potential is an important feature of neurons. It deserves to investigate WSN P systems with decaying of potential (as in Freund, Ionescu, & Oswald, 2008): the unused potential, irrespective of its size, decreases in each step with a specified amount—(one unit, for instance).

We end with the belief that SN P systems with weights and potentials deserve further research efforts.

Acknowledgments

The work of J.W. and L.P. was supported by the National Natural Science Foundation of China (grant nos. 60674106, 30870826 and 60703047), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of Gh.P. and M.J.P.-J. was supported by project TIN2009–13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds, and the Proyecto de Excelencia con Investigador de Reconocida Valía, of the Junta de Andalucía, grant P08–TIC 04200.

A series of suggestions made by two anonymous referees, who carefully read the letter, are gratefully acknowledged.

Ho Simon Wang, at HUST Academic Writing Center, provided tutorial help to improve the presentation of the article. We are also deeply indebted to K. G. Subramanian for his help in improving the English in the letter.

This is an expanded version of a paper presented at the Tenth Workshop on Membrane Computing, WMC10, Curtea de Argeş, Romania, 2009.

References

Freund, R., Ionescu, M., & Oswald, M. (2008). Extended spiking neural P systems with decaying spikes and/or total spiking. *Intern. J. Found. Computer Sci.*, *19*, 1223–1234.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory on NP-completeness*. New York: W. H. Freeman.

Gerstner, W., & Kistler, W. (2002). *Spiking neuron models. Single neurons, populations, plasticity*. Cambridge: Cambridge University Press.

Ionescu, M., Păun, Gh., & Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae*, *71*, 279–308.

Leporati, A., Mauri, G., Zandron, C., Păun, Gh., & Pérez-Jiménez, M. J. (2009). Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, *8*, 681–702.

Leporati, A., Zandron, C., Ferretti, C., & Mauri, G. (2007). On the computational power of spiking neural P systems. In M. A. Gutiérrez-Naranjo, Gh. Păun, A. Romero, & A. Riscos (Eds.), *Proceedings of Fifth Brainstorming Week on Membrane Computing* (pp. 227–246). Sevilla: Fenix Editora.

Maass, W. (2002). Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, *8*, 32–36.

Maass, W., & Bishop, C. (Eds.). (1999). *Pulsed neural networks*. Cambridge, MA: MIT Press.

Martin-Vide, C., Pazos, J., Păun, Gh., & Rodriguez-Patón, A. (2003). Tissue P systems. *Theoretical Computer Sci.*, *296*, 295–326.

Minsky, M. (1967). *Computation: Finite and infinite machines*. Englewood Cliffs, NJ: Prentice Hall.

Păun, Gh. (2000). Computing with membranes. *J. Computer Syst. Sci.*, *43*, 108–143.

Păun, Gh. (2002). *Membrane computing: An introduction*. Berlin: Springer-Verlag.

Păun, Gh. (2007). Twenty six research topics about spiking neural P systems. In M. A. Gutiérrez, Gh. Păun, A. Romero, & A. Riscos (Eds.), *Proc. Fifth Brainstorming Week on Membrane Computing* (pp. 263–280). Sevilla: Fenix Editora.

Păun, Gh., & Pérez-Jiménez, M. J. (2008). Spiking neural P systems. An overview. In A. B. Porto, A. Pazos, & W. Buno (Eds.), *Advancing artificial intelligence through biological process applications* (pp. 60–73). Hershey, PA: Medical Information Science Reference.

Păun, Gh., & Rozenberg, G. (2002). A guide to membrane computing. *Theoretical Computer Sci.*, *287*, 73–100.

Păun, Gh., & Rozenberg, G. (2010). An introduction to and an overview of membrane computing. In Gh. Păun, G. Rozenberg, & A. Salomaa (Eds.), *Handbook of membrane computing*. New York: Oxford University Press.

Păun, Gh., Rozenberg, G., & Salomaa, A. (Eds.). (2010). *Handbook of membrane computing*. New York: Oxford University Press.

Rozenberg, G., & Salomaa, A. (Eds.). (1997). *Handbook of formal languages*. Berlin: Springer-Verlag.

Received October 7, 2009; accepted April 6, 2010.