Asynchronous Spiking Neural P Systems with Local Synchronization

Tao Song[1], Linqiang Pan[1], Gheorghe Păun[2]

[1] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Summary. Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. Asynchronous SN P systems are non-synchronized systems, where the use of spiking rules (even if they are enabled by the contents of neurons) is not obligatory. In this paper, with a biological inspiration (in order to achieve some specific biological functioning, neurons from the same functioning motif or community work synchronously to cooperate with each other), we introduce the notion of local synchronization into asynchronous SN P systems. The computation power of asynchronous SN P systems with local synchronization is investigated. Such systems consisting of general neurons (resp. unbounded neurons) and using standard spiking rules are proved to be universal. Asynchronous SN P systems with local synchronization consisting of bounded neurons and using standard spiking rules characterize the semilinear sets of natural numbers. These results show that the local synchronization is useful, it provides some "programming capacity" useful for achieving a desired computational power.

1 Introduction

Membrane computing is one of the recent branches of natural computing. It was initiated in [9] and has developed rapidly (already in 2003, ISI considered membrane computing as 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, etc.) from the structure and the functioning of a single cell and from complexes of

cells, such as tissues and organs, including the brain. The obtained models are distributed and parallel computing devices, usually called *P systems*. There are three main classes of P systems investigated: cell-like P systems (based on a cell-like (hence hierarchical) arrangement of membranes delimiting compartments where multisets of chemicals evolve according to given evolution rules) [9], tissue-like P systems (instead of hierarchical arrangement of membranes, one considers arbitrary graphs as underlying structures, with membranes placed in the nodes, and with the edges corresponding to communication channels) [7], and neural-like P systems. Many variants of all these systems have been considered; an overview of the field can be found in [10] and [11], with up-to-date information available at the membrane computing website (`http://ppage.psystems.eu`). For an introduction to membrane computing, one may consult [10] and [11]. The present work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short), introduced in [5].

SN P systems are a class of distributed and parallel computing models inspired by spiking neurons. As we know, neurons are one of the most interesting cell-types in the human body. A large number of neurons working in a cooperative manner are able to perform tasks (such as thought, self-awareness, intuition) that are not yet matched by the tools we can build with our current technology. However, we believe that the distributed manner in which the brain processes information is important in obtaining better performance for electronic computers, that is why we are interested in SN P systems defined as a *computation model*. We stress that in this work SN P systems are a subject of a theoretical computer science investigation, without any intention to propose a platform for modeling biological processes.

Briefly, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph, where neurons send signals (called *spikes* and denoted by the symbol a in what follows) along *synapses* (arcs of the graph). Spikes evolve by means of *standard spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{\mathrm{a}\}$ and c, d are natural numbers, $c \geq 1$, $d \geq 0$. The meaning is that if a neuron contains k spikes such that $a^k \in L(E)$ and $k \geq c$, then it 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 starting in the neuron where the rule was applied. There are also *standard 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. *Extended rules* were considered in [3]: these rules are of the form $E/a^c \to a^p; d$, with the meaning that when using this rule, c spikes are consumed and p spikes are produced. Because p can be 0 or greater than 0, we obtain a generalization of both standard spiking and forgetting rules.

A rule is *bounded* if it is of the form $a^i/a^c \to a^p; d$, where $0 < c \leq i$, $0 < p \leq c$, $d \geq 0$. A neuron is *bounded* if it contains only bounded rules. A rule is called *unbounded* if it is of the form $a^i(a^j)^*/a^c \to a^p; d$, with $i \geq 0$, $j \geq 1$, $c \geq p > 0$, $d \geq 0$. (Note that also rules of the form $a^i(a^j)^+/a^c \to a^p; d$ are covered, as they can be rewritten in the form $a^{i+j}(a^j)^*/a^c \to a^p; d$.) A neuron is *unbounded* if it

contains only unbounded rules. A neuron is *general* if it contains both bounded and unbounded rules. An SN P system is *bounded* if all neurons in the system are bounded. It is *unbounded* if it has both bounded and unbounded neurons. An SN P system is *general* if it contains at least one general neuron (i.e., a neuron containing both bounded and unbounded rules).

An SN P system works in a synchronized manner. A global clock is assumed, and in each time unit, the rule to be applied in each neuron is non-deterministically chosen; one rule must be applied in each neuron with applicable rules. 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 one, 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.

Synchronized SN P systems using standard rules were proved to be computationally complete both in the generating and the accepting case [5]. In the proof of these results, the synchronization plays a crucial role. However, both from a mathematical point of view and from a neuro-biological point of view, it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory. Even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used. The neuron may remain unfired, maybe receiving spikes from the neighboring neurons. If the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused. If further spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now). With this motivation, asynchronous SN P systems were introduced in [2]. It was proved that asynchronous general SN P systems with extended rules are equivalent with Turing machines; asynchronous unbounded SN P systems with extended rules are not universal. However, it remains open whether asynchronous general SN P systems with standard rules are universal.

In a biological neural system, motifs with 4-5 neurons and communities with 12-15 neurons, associated with some specific functioning, are rather common [1]. The neurons from the same motif or community will work synchronously to cooperate with each other. That is, in a biological neural system, neurons work asynchronously at the global level, but neurons from the same functioning motif or community work synchronously at the local level. With this biological inspiration, we introduce asynchronous SN P systems with *local synchronization*, where a family of sets of neurons (we call them *ls-sets*) is specified; if one of the neurons from an ls-set fires, then all neurons from this set should fire, provided that they have enabled rules. Of course, it is possible that all neurons from an ls-set remain unfired even if they have enabled rules, because of the global asynchronous mode.

In this work, we prove that asynchronous general or unbounded SN P systems with local synchronization using standard rules are universal; in the bounded case a characterization of semilinear sets of numbers is obtained.

In the asynchronous SN P systems constructed in [2], the non-determinism comes from two resources: (1) the asynchronous mode; (2) the non-deterministic choice of enabled rules to be applied. Each neuron of a system constructed in the present paper works in a deterministic way, in the sense that at each step each neuron has at most one enabled rule; however, the whole system works in a non-deterministic way because of the asynchronous mode.

In the proofs of the universality results in this work, the feature of local synchronization plays a crucial role. An asynchronous general SN P system with standard rules loses this "programming capacity" ensured by the extended rules and the local synchronization. So, our research gives some hint to support the conjecture that an asynchronous general SN P system with standard rules is non-universal [2].

2 Preliminaries

It is useful for readers to have some familiarity with basic elements of language theory, e.g., from [13], as well as basic membrane computing [10]. We here only introduce the necessary prerequisites.

The set of natural numbers is denoted by $\mathbb{N}$.

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: (i) λ and each $a \in V$ is a regular expression, (ii) 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 (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, 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 of Turing computable sets of numbers. ($SLIN$ is the family of length sets of regular languages – languages characterized by regular expressions; NRE is the family of length sets of recursively enumerable languages – those recognized by Turing machines.)

In the university proofs, the notion of a *register machine* is used. A register machine 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 element of 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 non-zero, 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. The register machine starts with all registers empty (i.e., storing the number zero). It applies the instruction with label l_0 and proceeds to apply instructions as indicated by labels (and, in the case of SUB instructions, by the contents of registers). If the register machine reaches the halt instruction, then 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 which are Turing computable, hence they characterize NRE [8].

Without loss of generality, it can be assumed that l_0 labels an ADD instruction, 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 (its content is only added to).

A strongly monotonic register machine is a non-deterministic machine with only one register (this is also the output register). This register is initially zero and can only be incremented by 1 at each computation step (that is why we call such a machine strongly monotonic). When the machine halts, the value stored in the register is said to be generated. It is known that a set of natural numbers is semilinear if and only if it can be generated by a strongly monotonic register machine.

A register machine can also be used in the accepting mode: one starts with all registers empty, except one specified register, the input one, where a number x is introduced; the computation starts (with instruction with label l_0) and, if it halts, then the number x is accepted. In this way, again all sets of numbers from NRE are characterized. Furthermore, a register machine can be used for computing functions $\varphi : \mathbb{N} \longrightarrow \mathbb{N}$: certain registers are designated as input registers and a specific one as the output register; numbers $x_1, \ldots, x_k$ are introduced in the input registers and the value of a function $\varphi(x_1, \ldots, m_k)$ is obtained in the output register – providing that φ is defined for $x_1, x_2, \ldots, x_k$, otherwise the computation never halts. Turing computable functions can be computed in this way.

We use the following convention. When the power of two number generating/accepting devices D_1 and D_2 are compared, number zero is ignored, that is, $N(D_1) = N(D_2)$ 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 Asynchronous Spiking Neural P Systems with Local Synchronization

In this section, we introduce the variant of SN P systems investigated in this work – asynchronous spiking neural P systems with local synchronization. The definition is complete, but familiarity with the basic elements of classic SN P systems (e.g., from [11], [12]) is helpful.

An *asynchronous spiking neural P system (without delay) with local synchronization* is a construct of the form:

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, Loc, syn, out), \text{ where}$$

- $m \geq 1$ is the *degree* of the system;
- $O = \{a\}$ is the singleton alphabet (a is called *spike*);
- $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons* of the form $\sigma_i = (n_i, R_i)$ with $1 \leq i \leq m$, where
  (1) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
  (2) R_i is a finite set of *extended rules* of the following form: $E/a^c \to a^p$, where E is a regular expression over O, $c \geq 1$ and $c \geq p \geq 0$;
- $Loc = \{loc_1, loc_2, \ldots, loc_l\} \subseteq \mathcal{P}(\{\sigma_1, \sigma_2, \ldots, \sigma_m\})$ is the *family of sets of locally synchronous neurons* (we call these sets ls-sets), where $\mathcal{P}(\{\sigma_1, \sigma_2, \ldots, \sigma_m\})$ is the power set of $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$; the number $\max\{|loc_1|, |loc_2|, \ldots, |loc_l|\}$ is the *local synchronization degree* of the system;
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ is the set of *synapses* between neurons;
- $out \in \{1, 2, \ldots, m\}$ indicates the *output* neuron.

A rule $E/a^c \to a^p$ with $p \geq 1$ is called *extended firing* (we also say *spiking*) *rule*; a rule $E/a^c \to a^p$ with $p = 0$ is written in the form $E/a^c \to \lambda$ and is called a *forgetting rule*. If $L(E) = \{a^c\}$, then the rules are written in the simplified forms $a^c \to a^p$ and $a^c \to \lambda$. A rule of the type $E/a^c \to a$ and $a^c \to \lambda$ is said to be *standard*.

The rules are applied as follows. If neuron σ_i contains k spikes and $a^k \in L(E)$, $k \geq c$, then the rule $E/a^c \to a^p \in R_i$ is enabled and can be applied. This means that c spikes are consumed (thus $k - c$ spikes remain in neuron σ_i), the neuron is fired, and it produces p spikes. The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each neuron σ_j receives p spikes). Every neuron can contain several rules. Because two firing rules, $E_1/a^{c_1} \to a^{p_1}$ and $E_2/a^{c_2} \to a^{p_2}$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more spiking rules are enabled in a neuron at some moment, and then one of them is chosen non-deterministically.

If the rule is a forgetting one of the form $E/a^c \to \lambda$, then, when it is applied, $c \geq 1$ spikes are removed.

A global clock is assumed, marking the time for all neurons. In each time unit, any neuron is free to use a rule or not, i.e., a neuron can remain still in spite of the fact that it contains rules which are enabled by its contents. If the content of the

neuron is not changed, a rule which is enabled in a given step can fire later. If new spikes are received, then it is possible that other rules will be enabled and applied or not. Furthermore, for neurons in the same ls-set loc_j, if one of these neurons fires, then all neurons in loc_j that have enabled rules should fire. Of course, it is possible that all neurons from loc_j remain unfired even if they have enabled rules. That is, all neurons from loc_j may remain still, or all neurons from loc_j with enabled rules fire at a same step (of course, neurons without enabled rules cannot fire). Hence, neurons work asynchronously at the global level, but neurons in each ls-set work synchronously.

The "state" of the system at a given time is described by the number of spikes present in each neuron. That is, the *configuration* of the system is of the form $\langle r_1, r_2, \ldots, r_m \rangle$ for $r_i \geq 0$, which indicates that neuron σ_i contains r_i spikes. With this notation, the *initial configuration* of the system is $\langle n_1, n_2, \ldots, n_m \rangle$. By using the rules as described above, one can define *transitions* among configurations. Any series of transitions starting from the initial configuration is called a *computation*. A computation is *successful* if it reaches a configuration where no rule can be applied in any neuron (i.e., the SN P system has halted). The *result of a computation* is defined here as the total number of spikes sent into the environment by the output neuron. (Because of the asynchronous mode, now "the time does not matter". The output neuron can remain still for any number of steps between two consecutive spikes, therefore the result of a computation can no longer be defined in terms of the steps between two consecutive spikes as in the standard SN P systems definition.)

Specifically, a number x is generated by an SN P system if there is a halting computation of the system where the output neuron emits exactly x spikes (if several spikes are emitted at the same time, all of them are counted). Because of the non-determinism in using the rules, a given system computes in this way a set of numbers.

The rules, the neurons, and the SN P systems are called *bounded, unbounded*, or *general* as defined in the Introduction – the definitions are obvious, so we do not recall them here.

Asynchronous SN P systems with local synchronization can be used as computing devices in various ways, but here we consider them only as generators of numbers. We denote by $N(\Pi)$ the set of numbers generated by an asynchronous SN P system with local synchronization Π, and by $NSpik_{out}P_m^{locsyn}(\alpha, ls)$ with $\alpha \in \{gen, boun, unb\}$ and $ls \geq 0$, the family of such sets of numbers generated by systems of type α (*gen* stands for general, *boun* for bounded, *unb* for unbounded), with at most m neurons, and local synchronization degree at most ls. If one of the parameters m and ls is not bounded, then it is replaced with $*$. The subscript *out* reminds us of the fact that we count all spikes sent into the environment as computation results.

4 Asynchronous General SN P Systems with Local Synchronization

In this section, we investigate the computation power of asynchronous general SN P systems with local synchronization, and we prove that such systems using standard rules are universal. It was formulated as an open problem whether asynchronous SN P systems with standard rules are universal and it was conjectured that the answer is negative in [2]. So, the feature of local synchronization provides a useful "programming capacity".

As it is usual in the area of spiking neural P systems, asynchronous SN P systems with local synchronization are represented graphically, which may be easier to understand than in a symbolic way. We use an oval with rules inside to represent a neuron, and a directed graph to represent the structure of the system: the neurons are placed in the nodes of the graph and the edges represent the synapses; the output neuron has an outgoing arrow, suggesting its communication with the environment.

Theorem 1. $NRE = NSpik_{out}P_*^{locsyn}(gen, *)$.

We only have to prove that $NRE \subseteq NSpik_{out}P_*^{locsyn}(gen, *)$, since the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). To this aim, we use the characterization of NRE by means of generating register machines. Let us consider a register machine $M = (m, H, l_0, l_h, I)$. As mentioned in Section 2, without any loss of generality, we may assume that in the halting configuration, all registers different from register 1 are empty, and that the output register is never decremented during a computation. For each register r of M, let s_r be the number of SUB instructions acting on register r. If there is no such SUB instruction, then $s_r = 0$, which is the case for the first register $r = 1$. In what follows, a specific asynchronous SN P system with local synchronization Π will be constructed to simulate the register machine M, where each neuron in system Π has only standard rules.

The system Π consists of three types of modules – ADD modules, SUB modules, and a FIN module. ADD modules and SUB modules are used to simulate the ADD and SUB instructions of M, respectively; the FIN module is used to output a computation result.

In general, a neuron σ_r is associated with each register r of M; the number stored in register r is encoded by the number of spikes in neuron σ_r. Specifically, if register r holds the number $n \geq 0$, then neuron σ_r contains $2n$ spikes. With each label l_i of an instruction in M, a neuron σ_{l_i} is associated. In the initial configuration, all neurons are empty, with the exception of neuron σ_{l_0} associated with the initial instruction l_0 of M, which contains one spike. During a computation, a neuron σ_{l_i} having one spike inside will become active and starts to simulate an instruction $l_i : (\mathtt{OP}(r), l_j, l_k)$ of M: starting with neuron σ_{l_i} activated, one changes neuron σ_r as requested by OP, then one introduces one spike into neuron σ_{l_j} or neuron σ_{l_k}, which becomes active in this way. When neuron σ_{l_h} (associated with the

label l_h of the halting instruction of M) is activated, a computation in M is completely simulated in Π; the FIN module starts to output the computation result (the number of spikes sent into the environment by the output neuron corresponds to the number stored in register 1 of M).

In what follows, the modules ADD, SUB, and FIN are given in the standard graphical way, also specifying their ls-sets, and their work is briefly analyzed.

Module ADD (shown in Figure 1) – simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

The initial instruction of M, the one with label l_0, is an ADD instruction. Let us assume that at step t, an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ has to be simulated, with one spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neurons, except in those neurons associated with registers. Having one spike inside, the rule $a \to a$ is enabled, neuron σ_{l_i} can fire, and at some time it will do it (otherwise, the computation does not halt), sending one spike to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$, respectively. Having one spike inside, neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ can fire; neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ will fire at the same step (because they are in the same ls-set $\{\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}\}$) sending two spikes to neurons $\sigma_{l_i^{(3)}}$, $\sigma_{l_i^{(4)}}$ and σ_r, respectively. In this way, the number of spikes in neuron σ_r is increased by two, which corresponds to the fact that the number stored in register r is increased by one. For neuron $\sigma_{l_i^{(3)}}$, we have two possible cases.

Proof. (1) Neuron $\sigma_{l_i^{(3)}}$ fires before neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire. Note that, in this case, when neuron $\sigma_{l_i^{(3)}}$ fires, neuron $\sigma_{l_i^{(4)}}$ does not fire (it has no enabled rule), although neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ are in the same ls-set $\{\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}\}$. Neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive one spike from neuron $\sigma_{l_i^{(3)}}$ and they keep inactive. After neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ receive two spikes from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$, both of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ can fire, and they will fire at a same step, sending two spikes to neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$. In this way, both of neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ accumulate 3 spikes. With 3 spikes inside, neuron $\sigma_{l_i^{(5)}}$ can fire at any step by the rule $a^3 \to a$, and send one spike to neuron σ_{l_j}. After neuron σ_{l_j} receives one spike, it becomes active, starting to simulate the instruction l_j of M.

When neuron σ_{l_j} fires, it is possible that neuron $\sigma_{l_i^{(6)}}$ still contains three spikes because of non-synchronization. In this case, neuron $\sigma_{l_i^{(6)}}$ should also fire (because they are in the same ls-set) at the same step removing all three spikes, which insures that no spike remains in the module (except those in σ_r).

(2) Neuron $\sigma_{l_i^{(3)}}$ fires after neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire. In this case, neuron $\sigma_{l_i^{(3)}}$ accumulates three spikes and neuron $\sigma_{l_i^{(4)}}$ has two spikes. Neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ will fire at a same step. Similar to case (1), neuron σ_{l_k} will become active, starting to simulate the instruction l_k of M (at the same step "cleaning" neuron $\sigma_{l_i^{(5)}}$).

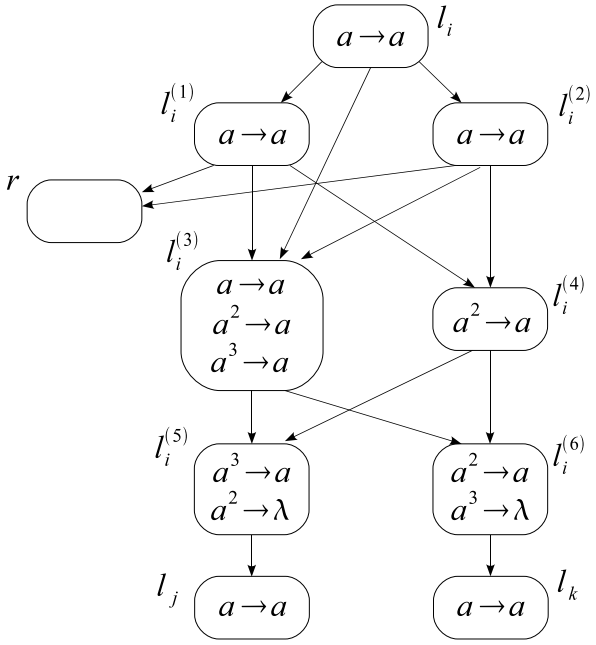

Fig. 1. Module ADD with four ls-sets $\{\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}\}$, $\{\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}\}$, $\{\sigma_{l_i^{(5)}}, \sigma_{l_k}\}$ and $\{\sigma_{l_i^{(6)}}, \sigma_{l_j}\}$ for simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

Therefore, after firing neuron σ_{l_i}, the system adds two spikes to neuron σ_r and non-deterministically fires one of neurons σ_{l_j} and σ_{l_k}, which correctly simulates the ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

Module SUB (shown in Figure 2) – simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$.

A SUB instruction l_i is simulated in Π in the following way. Initially, neuron σ_{l_i} has one spike, and other neurons are empty, except the neurons associated with registers. With one spike inside, the rule $a \to a$ in neuron σ_{l_i} is enabled, and it will fire at some step sending one spike to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r. These neurons will fire simultaneously, because they are in the same ls-set. For neuron σ_r, there are two cases.

(1) Before receiving one spike from neuron σ_{l_i}, neuron σ_r contains $2n$ $(n > 0)$ spikes (corresponding to the fact that the number stored in register r is n, and $n > 0$). In this case, neuron σ_r gets $2n + 1$ spikes and the rule $a(a^2)^+/a^3 \to a$ is enabled. So, when neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r fire at some step, they send three spikes to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. In neuron σ_r, three spike are consumed, ending with $2n + 1 - 3 = 2(n - 1)$ spikes, which simulates the fact that the number stored in register r is decreased by one. With three spikes

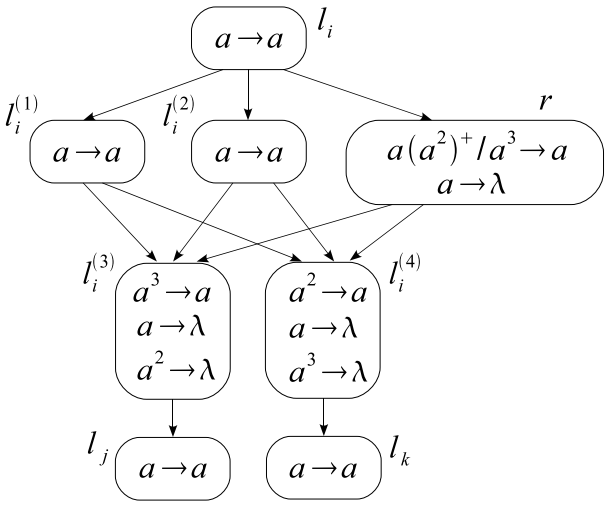

Fig. 2. Module SUB with three ls-sets $\{\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}, \sigma_r\}$, $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_k}\}$ and $\{\sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_j}\}$ for simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

inside, neuron $\sigma_{l_i^{(3)}}$ can fire, sending one spike to neuron σ_{l_j}, hence neuron σ_{l_j} will become active, and the system Π starts to simulate instruction l_j of M.

When neuron σ_{l_j} fires, it is possible that neuron $\sigma_{l_i^{(4)}}$ also contains three spikes because of non-synchronization. In this case, all neurons $\sigma_{l_s^{(4)}}$ should also fire (because they are in the same ls-set $\{\sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_j}\}$) removing simultaneously the three spikes, hence these neurons return to the initial state, with no spike inside.

(2) When receiving one spike from neuron σ_{l_i}, neuron σ_r has no spike inside (corresponding to the fact that the number stored in register r is 0). In this case, after neuron σ_r receives one spike from neuron σ_{l_i}, the rule $a \to \lambda$ in neuron σ_r is enabled. When neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r fire at some step, they send two spikes to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. In neuron σ_r, one spike is consumed, ending with 0 spikes, which means that the number stored in register r of M is zero. With two spikes inside, neuron $\sigma_{l_i^{(4)}}$ can fire, sending one spike to neuron σ_{l_k}, hence neuron σ_{l_k} will become active, and the system Π starts to simulate instruction l_k of M.

Similar to case (1), the ls-set $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_k}\}$ ensures that no "wrong" step is done in system Π.

The simulation of SUB instruction is correct: system Π starts from spiking neuron σ_{l_i} and ends in firing neuron σ_{l_j} (if the number stored in register r is greater than 0 and it was decreased by one), or in firing neuron σ_{l_k} (if the number stored in register r is 0).

Note that there is no interference between the ADD modules 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 interferences between two SUB modules. Specifically, if there are several SUB instructions l_v that act on the same register r, then neuron σ_r has synapses to all neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$. When a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated, in the SUB module associated with l_v ($l_v \neq l_i$) all neurons receive no spike except for neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$, each of them having one spike inside. Because we have the ls-sets $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_k}\}$ and $\{\sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_j}\}$, when neuron $\sigma_{l_i^{(3)}}$ (resp. $\sigma_{l_i^{(4)}}$) fires, each of neurons $\sigma_{l_v^{(3)}}$ (resp. $\sigma_{l_v^{(4)}}$) ($l_v \neq l_i$) should also fire at the same step removing its spike. Consequently, the interference among SUB modules will not cause undesired steps in Π (i.e., steps that do not correspond to correct simulations of instructions of M).

Module FIN (shown in Figure 3) – outputting the result of computation.

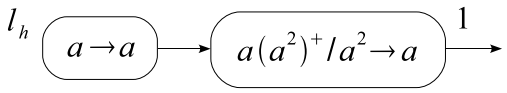

Fig. 3. The FIN module of Π

The functioning of this module is obvious: after activating the neuron σ_{l_h}, neuron σ_1 outputs one spike for each two spikes present inside; the last spike remains idle in the system.

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

In Theorem 1, the number m of neurons and the local synchronization degree ls are not bounded (thus, denoted by $*$). As expected, these parameters can be bounded by making use of the fact that there are (small) universal register machines. Such machines are given, e.g., in [6], but they are used in the accepting mode: the code of a particular register machine is introduced in register 1, an input for the particular machine is introduced in register 2, and the result is obtained in register 0. The universal machine halts if and only if the particular machine halts for the given input. The problem which remains to be solved is to pass from such an universal register machine to a generative SN P system.

Corollary 1. $NRE = NSpik_{out}P_{152}^{locsyn}(gen, 5)$.

Consider the universal register machine M_u from [6] as shown in Figure 4. It takes the code $code(M)$ of a particular register machine M which computes a function φ in register 1 and a number x in register 2, and outputs the value of $\varphi(x)$ in register 0.

Take an arbitrary set K in NRE and consider its membership function $\varphi_K : \mathbb{N} \longrightarrow \{0,1\}$. There is a register machine M_K computing this function. Starting with a number x in its input register, M_K halts if and only if $x \in K$, hence $\varphi_K(x) = 1$. Let $code(M_K)$ be its code; we introduce $code(M_K)$ in register 1 of M_u, thus obtaining a register machine $M_u(K)$. It takes any natural number n (in register 2) and halts (with 1 in register 0) if and only if $n \in K$. We modify $M_u(K)$ as follows. A further register is added, labeled with 8; consider two further labels, l_{-1}, l_{-2}, with l_{-1} being the initial label of the machine we want to obtain. Consider also the instructions $l_{-1} : (\texttt{ADD}(2), l_{-2}, l_{-2})$, $l_{-2} : (\texttt{ADD}(8), l_{-1}, l_0)$, where l_0 is the start label of M_u. The register machine M' obtained in this way has 9 registers, 11 ADD and 13 SUB instructions, and 25 labels; the result of a computation is stored in register 8, which is never decremented during a computation. Clearly, $n \in N(M')$ if and only if $\varphi_K(n) = 1$, hence $n \in K$, therefore $N(M') = K$: M' first "proposes" a number x to machine $M_u(K)$, by introducing it both in register 2, as needed for M_u and in register 8, the output one of M'; when the computation of $M_u(K)$ halts, also the computation of M' halts. Therefore, the number x is accepted and the computation of M' halts if and only if $x \in K$.

$l_0 : (\texttt{SUB}(1), l_1, l_2)$, $l_1 : (\texttt{ADD}(7), l_0)$,
$l_2 : (\texttt{ADD}(6), l_3)$, $l_3 : (\texttt{SUB}(5), l_2, l_4)$,
$l_4 : (\texttt{SUB}(6), l_5, l_3)$, $l_5 : (\texttt{ADD}(5), l_6)$,
$l_6 : (\texttt{SUB}(7), l_7, l_8)$, $l_7 : (\texttt{ADD}(1), l_4)$,
$l_8 : (\texttt{SUB}(6), l_9, l_0)$, $l_9 : (\texttt{ADD}(6), l_{10})$,
$l_{10} : (\texttt{SUB}(4), l_0, l_{11})$, $l_{11} : (\texttt{SUB}(5), l_{12}, l_{13})$,
$l_{12} : (\texttt{SUB}(5), l_{14}, l_{15})$, $l_{13} : (\texttt{SUB}(2), l_{18}, l_{19})$,
$l_{14} : (\texttt{SUB}(5), l_{16}, l_{17})$, $l_{15} : (\texttt{SUB}(3), l_{18}, l_{20})$,
$l_{16} : (\texttt{ADD}(4), l_{11})$, $l_{17} : (\texttt{ADD}(2), l_{21})$,
$l_{18} : (\texttt{SUB}(4), l_0, l_h)$, $l_{19} : (\texttt{SUB}(0), l_0, l_{18})$,
$l_{20} : (\texttt{ADD}(0), l_0)$, $l_{21} : (\texttt{ADD}(3), l_{18})$,
$l_h : \texttt{HALT}$

Fig. 4. A universal register machine M_u from Korec [6]

As in the proof of Theorem 1, we can construct an asynchronous SN P system with local synchronization $\Pi_{M'}$ to simulate the register machine M'. The system $\Pi_{M'}$ has

Proof.
- 9 neurons for the 9 registers,
- 25 neurons for the 25 labels,

- 6×11 neurons for the 11 ADD instructions,
- 4×13 neurons for the 13 SUB instructions,

which gives a total of 152 neurons.

The maximal size of ls-sets in the ADD module shown in Figure 1 is 2; the maximal size of ls-sets in the SUB module shown in Figure 2 is $\max\{3,\ s_r + 1\}$, where s_r is the number of SUB instructions acting on register r; the maximal size of ls-sets in the FIN module shown in Figure 3 is 0. So, the local synchronization degree is not more than $\max\{3, s_r + 1\}$. We can check that register 5 in M'_u has 4 SUB instructions, which is maximal, hence $\max\{3, s_r + 1\} = 5$. Therefore, our corollary holds. □

The above way of passing from computing universal register machines to register machines used in the generative way can be used in all results in membrane computing which show the computational completeness of a class of P systems by means of proofs based on the simulation of register machines. In this way, bounds on many parameters of the resulting P systems can be obtained. Note that many such bounds were reported in membrane computing (obtained by directly simulating a generating register machine), but no such result was reported for the number of neurons in SN P systems which characterize NRE. For instance, all universality results given in Chapter 13 of [11] are stated for SN P systems with arbitrarily many neurons. All these results can be improved from this point of view by using the technique from the proof of the previous corollary. Actually, also the number of rules in an SN P system should be considered (neurons can be saved by putting in the same neuron several rules), hence a double hierarchy should be discussed: number of neurons and number of rules.

5 Asynchronous Unbounded SN P System with Local Synchronization

In [2] it was proved that asynchronous unbounded SN P systems with extended rules can only characterize $SLIN$. In this section, making use of the "programming capability" of local synchronization, we prove that asynchronous unbounded SN P systems with local synchronization can achieve the Turing completeness.

Theorem 2. $NRE = NSpik_{out}P_*^{locsyn}(unb, *)$.

We only have to prove that $NRE \subseteq NSpik_{out}P_*^{locsyn}(unb, *)$, since the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). As in the proof of Theorem 1, let us consider a register machine $M = (m, H, l_0, l_h, I)$, with the properties specified in Section 2. For each register r of M, let s_r be the number of instructions of the form $l_i : (\texttt{SUB}(r), l_j, l_k)$.

As in the proof of Theorem 1, we construct an SN P system Π' consististing of three types of modules – ADD, SUB, and FIN. The first and the third types of modules can be easily obtained by modifying the ADD and FIN modules from Figures 1 and 3, respectively, while the SUB module is given in Figure 5.

Specifically, we observe that in the ADD and FIN modules from the proof of Theorem 3, all bounded neurons never contain more than three spikes. Then, each bounded rule $a^c \to a$, $a^c \to \lambda$ can be replaced with the unbounded rule $a^c(a^4)^*/a^c \to a$, $a^c(a^4)^*/a^c \to \lambda$, respectively, and the functioning of modules ADD and FIN is not changed.

The difficulty with the module SUB from Figure 2 is that the neurons σ_r contain both unbounded rules and the bounded rule $a \to \lambda$, which we do not see how can be turned to an unbounded rule. That is why we present a different module SUB, more complex than that from Figure 2.

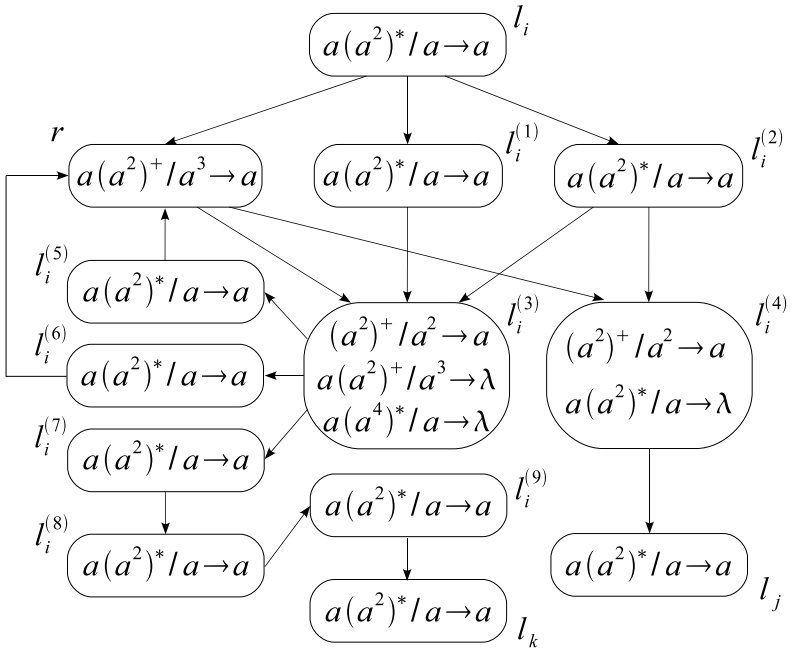

Fig. 5. The SUB module of Π' with ls-sets $\{\sigma_r, \sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}, \sigma_{l_i^{(8)}}\}$, $\{\sigma_{l_i^{(5)}}, \sigma_{l_i^{(6)}}, \sigma_{l_i^{(7)}}\}$, $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_1^{(9)}}, \sigma_{l_2^{(9)}}, \ldots, \sigma_{l_{s_r}^{(9)}}\}$

It is given in Figure 5 and it works as follows.

The simulation of a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is started with neuron σ_{l_i} having one spike. The rule $a(a^2)^*/a \to a$ is enabled and neuron σ_{l_i} fires at some step, sending a spike to neurons $\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}$ and σ_r, respectively. For neuron σ_r, there are two cases.

Proof. (1) Neuron σ_r has $2n$ ($n > 0$) spikes (corresponding to the fact that the number stored in register r is n, and $n > 0$) before it receives one spike from neuron σ_{l_i}. In this case, neuron σ_r gets $2n+1$ spikes and the rule $a(a^2)^+/a^3 \to a$ is enabled. When neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r fire at some step, they send three

spikes to neuron $\sigma_{l_i^{(3)}}$, as well as two spikes to neuron $\sigma_{l_i^{(4)}}$. (Note that neuron $\sigma_{l_i^{(8)}}$ is in the same ls-set with neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r, but it has no spike inside, so it cannot fire.) In neuron σ_r, three spikes are consumed, ending with $2n+1-3 = 2(n-1)$ spikes, which simulates the fact that the number stored in register r is decreased by one. Neuron $\sigma_{l_i^{(3)}}$ removes the three spikes inside by the forgetting rule $a(a^2)^+/a^3 \to \lambda$, meanwhile neuron $\sigma_{l_i^{(4)}}$ fires by the rule $(a^2)^+/a^2 \to a$, sending one spike to neuron σ_{l_j}, hence neuron σ_{l_j} will become active, and the system Π starts to simulate instruction l_j of M.

(2) Neuron σ_r has no spike inside (corresponding to the fact that the number stored in register r is 0) before it receives one spike from neuron σ_{l_i}. In this case, after neuron σ_r receives one spike from neuron σ_{l_i}, it keeps inactive for no rule is enabled. Neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at some step, sending two spikes to neuron $\sigma_{l_i^{(3)}}$, as well as one spike to neuron $\sigma_{l_i^{(4)}}$. With two spikes inside, neuron $\sigma_{l_i^{(3)}}$ can fire at some step, sending one spike to neurons $\sigma_{l_i^{(5)}}, \sigma_{l_i^{(6)}}$, and $\sigma_{l_i^{(7)}}$, respectively. At the same moment, neuron $\sigma_{l_i^{(4)}}$ removes the spike inside by the forgetting rule $a(a^2)^*/a \to \lambda$. Neurons $\sigma_{l_i^{(5)}}, \sigma_{l_i^{(6)}}$, and $\sigma_{l_i^{(7)}}$ fire at some step sending two spikes to neuron σ_r and one spike to neuron $\sigma_{l_i^{(8)}}$. After neuron σ_r receives the two spikes, it has three spikes inside and the rule $a(a^2)^+/a^3 \to a$ is enabled. Neurons σ_r and $\sigma_{l_i^{(8)}}$ fire at some moment, sending one spike to each of the neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$, and $\sigma_{l_i^{(9)}}$. Neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$, and $\sigma_{l_i^{(9)}}$ fire at some step, removing the spikes in neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$ and sending one spike to neuron σ_{l_k} from $\sigma_{l_i^{(9)}}$. Therefore, neuron σ_{l_k} becomes active, and the system Π starts to simulate instruction l_k of M. In neuron σ_r, there is no spike inside, which means that the number stored in register r of M is zero.

Note that it is possible to have interferences between two SUB modules. Specifically, if there are several SUB instructions l_v that act on the same register r, then neuron σ_r has synapses to all neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$. When a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated, in the SUB module associated with l_v ($l_v \neq l_i$) all neurons receive no spike except for neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$. Each of neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$ has one spike inside. Because we have the ls-set $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_1^{(9)}}, \sigma_{l_2^{(9)}}, \ldots, \sigma_{l_{s_r}^{(9)}}\}$, when neuron $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ fire, each of neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$ ($l_v \neq l_i$) should also fire at the same step removing its spike. Consequently, the interference among SUB modules will not cause undesired steps in Π (i.e., steps that do not correspond to correct simulations of instructions of M). Therefore, the simulation of SUB instruction is correct.

With the above description, we can see that the unbounded SN P system Π' can correctly simulate register machine M, hence $N(M) = N(\Pi')$. □

Similar with Corollary 1, we can also have the following corollary.

Corollary 2. $NRE = NSpik_{out}P_{217}^{locsyn}(unb, 12)$.

Indeed, the system Π' has

- 9 neurons for the 9 registers,
- 25 neurons for the 25 labels,
- 6×11 neurons for the 11 ADD instructions,
- 9×13 neurons for the 13 SUB instructions,

which gives a total of 217 neurons. The reader can easily check that the local synchronization degree is 12.

6 Asynchronous Bounded SN P Systems with Local Synchronization

In this section, we investigate the computation power of asynchronous bounded SN P systems with local synchronization. It was shown that asynchronous bounded SN P systems with extended rules can characterize semilinear sets of numbers [4], but it is open whether this result holds when the systems are restricted to use only standard rules. In the following, we prove that a set of natural numbers is semilinear if only if it can be generated by asynchronous bounded SN P systems with local synchronization using standard rules.

Lemma 1. $NSpik_{out}P_*^{locsyn}(boun, *) \subseteq SLIN$.

Take an asynchronous bounded SN P system with local synchronization using standard rules, Π. The number of neurons is fixed, and the number of spikes in each neuron is bounded, hence the number of configurations reached by Π is finite. Let $\mathcal{C}$ be the set of configurations of Π, and C_0 be the initial configuration of Π.

We construct the right-linear grammar $G = (\mathcal{C}, \{a\}, C_0, P)$, where P contains the following productions:

Proof. (1) $C \to C'$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Rightarrow C'$ in Π during which the output neuron does not spike;

(2) $C \to aC'$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Rightarrow C'$ in Π during which the output neuron spikes;

(3) $C \to \lambda$, for any $C \in \mathcal{C}$ in which all neurons have no enabled rules.

Clearly, the construction of G ensures the fact that $N(\Pi)$ is the length set of the regular language $L(G)$, hence it is semilinear. Therefore, $NSpik_{out}P_*^{locsyn}(boun, *) \subseteq SLIN$. □

Lemma 2. $SLIN \subseteq NSpik_{out}P_*^{locsyn}(boun, *)$.

Since a set of natural numbers is semilinear if and only if it can be generated by a strongly monotonic register machine, it is enough to prove that any strongly monotonic register machine can be simulated by an asynchronous bounded SN P system with local synchronization using standard rules. Let M be a strongly monotonic register machine. Clearly, the machine M has only register 1 and the ADD instructions of the forms $l_i : (\texttt{ADD}(1), l_j, l_k)$.

An asynchronous bounded SN P system with local synchronization using standard rules Π can be constructed as in the proof of Theorem 1 to simulate the strongly monotonic register machine M: we place the rule $a^2 \to a$ in neuron σ_1 (it is associated with register 1 and it is also the output neuron), also considering the ls-set consisting of neurons σ_1 and $\sigma_{l_v^{(3)}}, \sigma_{l_v^{(4)}}$ for all ADD instructions l_v of M. Moreover, in neuron σ_{l_h} we provide no rule (when M halts, also Π halts, so the FIN module is here not necessary). The above mentioned new ls-set make sure that as soon as the simulation of an ADD instruction of M reaches the neurons $\sigma_{l_j^{(3)}}, \sigma_{l_j^{(4)}}$, then also neuron σ_1 spikes, thus getting empty, ready for a further increment by one. That is, neuron σ_1 is either empty or it contains two spikes – in the latter case, one spike is sent to the environment. Thus, the equivalence of M and Π is obvious. □

By Lemmas 1 and 2, the following theorem holds.

Proof. **Theorem 3.** $NSpik_{out}P_*^{locsyn}(boun, *) = SLIN$.

7 Conclusions and Remarks

In this work, inspired by the fact that neurons in the same functioning brain motif or community work synchronously to achieve some specific biological functions, we introduce local synchronization into the framework of SN P systems. The computation power of asynchronous SN P systems with local synchronization is investigated. Asynchronous SN P systems with local synchronization consisting of general neurons (resp. unbounded neurons) and using standard spiking rules are proved to be universal. Asynchronous SN P systems with local synchronization consisting of bounded neurons and using standard spiking rules can characterize the semilinear sets of natural numbers. It was already known that (1) asynchronous general SN P systems with extended rules are universal; (2) asynchronous unbounded SN P systems with extended rules can only characterize semilinear sets of natural numbers [2]. However, the computation power of asynchronous general (resp. unbounded) SN P systems with standard rules is unknown. The results from this paper show that such systems can reach universality if they are provided with the "programming capability" of local synchronization. So, local synchronization is a powerful ingredient for achieving "Turing creative capability".

The local synchronization degrees in Corollary 1 and Corollary 2 are 5 and 12, respectively. It remains open whether or not these values can be improved. We conjecture that the value two is enough to achieve universality for asynchronous SN P systems with local synchronization consisting of general neurons (resp. unbounded neurons) and using standard spiking rules.

In the systems constructed in Theorem 1 and Theorem 2, forgetting rules are used. It remains open whether forgetting rules can be removed without any loss of computation power. We conjecture the answer is positive (that is, we believe that the feature of local synchronization is powerful enough to remove forgetting rules without decreasing the computation power).

Acknowledgment

This work of the first two authors was supported by National Natural Science Foundation of China (61033003, 91130034 and 30870826), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), Fundamental Research Funds for the Central Universities (2010ZD001 and 2010MS003), and National Science Foundation of Hubei Province (2008CDB113 and 2011CDA027). The work of Gh. Păun is supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. U. Alon: *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman&Hall/CRC, 2006
2. M. Cavaliere, O. Egecioglu, O.H. Ibarra, S. Woodworth, M. Ionescu, Gh. Păun: Asynchronous spiking neural P systems: decidability and undecidability. *Proc. 13th Int. Meeting on DNA Computing* (M.H. Garzon, H. Yan, eds.), Memphis, USA, LNCS 4848, Springer, Berlin, 2008, 246–255.
3. H. Chen, M. Ionescu, T. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7 (2008), 147–166.
4. O.H. Ibarra, S. Woodwort: Spiking neural P systems: some characterizations. *Preproc. 16th International Symposium on Fundamentals of Computation Theory, FCT 2007* (E. Csuhaj-Varjú, Z. Ésik, eds.), Budapest, Hungary, LNCS 4639, Springer, Berlin, 2007, 23–37.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2006), 279–308.
6. I. Korec: Small universal register machines. *Theoretical Computer Sci.*, 168 (1996), 267–301.
7. C. Martin-Vide, J. Pazos, Gh. Păun, A. Rodriguez-Patón: Tissue P systems. *Theoretical Computer Sci.*, 296 (2003), 295–326.
8. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, New Jersey, 1967.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
10. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
11. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford Univ. Press, Oxford, 2010.
12. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems: An overview. *Advancing Artificial Intelligence through Biological Process Applications* (A.B. Porto, A. Pazos, W. Buno, eds.), PA: Medical Information Science Reference, Hershey, 2008, 60–73.
13. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer, Berlin, 1991.