Proceedings of CMC-15
August 20 — 22, 2014, Prague, Czech Republic

Spiking Neural P Systems with Cooperating Rules

Venkata Padmavati Metta[1], Srinivasan Raghuraman[2], and Kamala Krithivasan[2]

[1] Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic
[2] Indian Institute of Technology, Chennai, India
vmetta@gmail.com, srini131293@gmail.com, kamala@iitm.ac.in

Abstract. The concept of cooperation and distribution as known from grammar systems is introduced to spiking neural P systems (in short, SN P systems) in which each neuron has a finite number of sets (called *components*) of rules. During computations, at each step only one of the components can be active for the whole system and one of the enabled rules from this active component of each neuron fires. The switching between the components occurs under different cooperation strategies. This paper considers the terminating mode, in which the switching occurs when no rule is enabled in the active component of any neuron in the system. By introducing this new mechanism, the computational power of asynchronous and sequential SN P systems with standard rules is investigated. The results are that asynchronous standard SN P systems with two components and strongly sequential unbounded SN P systems with two components are Turing complete.

1 Introduction

Cooperative grammar systems were introduced by Meersman and Rozenberg in [6], in the context of two-level grammars. The systematic study of cooperating distributed (for short, CD) grammar systems was initiated by Csuhaj-Varjú and Dassow in [2], where productions are distributed over a finite number of sets, called components. These components cooperate during the derivation process by applying productions on a common sentential form; following some fixed cooperation protocol.

The concept of cooperation and distribution as known from CD grammar systems is introduced to spiking neural P systems. Spiking neural P systems [5] are parallel and distributed computing models inspired by the neurophysiological behaviour of neurons sending electrical pulses of identical voltages called spikes to the neighbouring neurons through synapses. An SN P system is represented as a directed graph where nodes correspond to the neurons having spiking rules and forgetting rules. The rules involve the spikes present in the neuron in the form of occurrences of a symbol a. The arcs indicate the synapses among the neurons. The spiking rules are of the form $E \,/\, a^r \rightarrow a$ and are used only if

the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$, where $L(E)$ is the language represented by regular expression E. In this case a^r number of spikes are consumed and one spike is sent out. When neuron σ_i sends a spike, it is replicated in such a way that one spike is immediately sent to all neurons j such that $(i, j) \in syn$, where syn is the set of arcs between the neurons. The transmission of spikes takes no time, the spike will be available in neuron j in the next step. The forgetting rules are of the form $a^s \rightarrow \lambda$ and are applied only if the neuron contains exactly a^s spikes. The rule simply removes s spikes. For all forgetting rules, s must not be the member of $L(E)$ for any firing rule within the same neuron.

A rule is *bounded* if it is of the form $a^i/a^j \rightarrow a$, where $1 \leq j \leq i$, or of the form $a^k \rightarrow \lambda$, where $k \geq 1$. A neuron is bounded if it contains only bounded rules. A rule is called *unbounded* if is of the form $a^c(a^i)^*/a^j \rightarrow a$, where $c \geq 0$, $i \geq 1$, $j \geq 1$. 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 the neurons in the system are bounded. It is unbounded if it has bounded and unbounded neurons. Finally, an SN P system is general if it has general neurons (i.e., it contains at least one neuron which has both bounded and unbounded rules).

The usual SN P systems are synchronous (a global clock is assumed) and work in a maximally parallel manner, in the sense that all neurons that are fireable must fire. However, in any neuron, at most one rule is allowed to fire. One neuron is designated as the output neuron of the system and its spikes can exit into the environment, thus producing a spike train. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and a set of numbers, obtained by considering the total number of spikes emitted by the output neuron until the system halts. Two main types of results were obtained for synchronous SN P systems using standard rules (producing one spike): computational completeness in the case when no bound was imposed on the number of spikes present in the system, and a characterization of semi-linear sets of numbers in the case when a bound was imposed [5].

This paper introduces spiking neural P system with cooperating rules where each neuron has a finite number of sets of spiking and forgetting rules. Each set is called a component which can be empty. At any step or during a sequence of steps (depending on the mode of application) only one of the components is active for the whole system and only one of the enabled rules from this component of each neuron can fire during that step. After that another (not necessarily different) component of each neuron becomes active. The way of passing active control is called a protocol. Similar to CD grammar systems, series of cooperation protocols among the components in neurons of an SN P system can be considered, where for example any component, once started, has to perform exactly k, at most k, at least k, $k \geq 1$ or an arbitrary number of transition steps. In the so-called terminating mode, a component may stop working if and only if none of the rules

in that component of any neuron is applicable. In any case, the selection of the next active component is non-deterministic and only one component generates the output at a step, other components wait for receiving control.

This paper considers asynchronous SN P systems [1], where in any step, a neuron can apply or not apply its rules which are enabled by the number of spikes it contains (further spikes can come, thus changing the rules enabled in the next step). Because the time between two firings of the output neuron is now irrelevant, the result of a computation is the number of spikes sent out by the system, not the distance between certain spikes leaving the system. It was proved that such asynchronous SN P systems with extended rules are equivalent to Turing machines (as generators of sets of natural numbers) but universality of such systems with standard rules is still an open problem. The additional non-determinism introduced in the functioning of the system by the non-synchronization has more computing power in the case of using two components. That is, two component SN P systems with standard rules working asynchronously are equivalent to the Turing machines (interpreted as generators of sets of (vectors of) numbers).

The paper also considers sequential SN P systems in which, at every step of the computation, if there is at least one neuron with at least one rule that is fireable, we only allow one such neuron and one such rule (both nondeterministically chosen) to be fired. Here, not every step has at least one neuron with a fireable rule. (Thus, the system might be dormant until a rule becomes fireable. However, the clock will keep on ticking.) The sequential unbounded as well as general SN P systems are proved to be universal [4]. A system is strongly sequential, if at every step, there is at least one neuron with a fireable rule. It is shown that strongly sequential general SN P systems are universal but strongly sequential unbounded SN P systems are not universal [4]. In this paper, we also prove that strongly sequential unbounded SN P systems with two components are universal.

The paper is organized as follows. In the next section, register machines are defined. SN P systems with cooperating rules are introduced in Section 3. The universality of asynchronous two component SN P systems with standard rules is proved in Section 4 and that of strongly sequential SN P systems with standard unbounded neurons is proved in Section 5.

2 Prerequisites

We assume the reader to be familiar with formal language theory, CD grammar systems and membrane computing. The reader can find details about them in [10], [3], [9] etc.

The family of Turing computable sets of natural numbers is denoted by NRE (the notation comes from the fact that these numbers are the length sets of recursively enumerable languages). The family of NRE is also the family of sets of numbers generated/recognized by register machines. For the universality proofs in this paper, we use the characterization of NRE by means of register

machines [7]. Such a device - in the non-deterministic version - is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labelling an ADD instruction), l_h is the halt label (assigned to instruction $HALT$), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it.

The labelled instructions are of the following forms:

1. $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j,l_k non-deterministically chosen),
2. $l_i : (SUB(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
3. $l_h : HALT$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers). If we reach the halt instruction, then the number n present in register 1 (the registers are numbered from 1 to m) at that time is said to be generated by M. It is known (e.g., see, [7]) that register machines generate all sets of numbers which are Turing computable.

A register machine can also accept a set of numbers: a number n is accepted by M if, starting with n in register 1 and all other registers empty, the computation eventually halts (without loss of generality, we may assume that in the halting configuration all registers are empty). Deterministic register machines (i.e., with ADD instructions of the form $l_i : (ADD(r), l_j)$ working in the accepting mode are known to be equivalent to Turing machines.

It is also possible to consider register machines producing sets of vectors of natural numbers. In this case a distinguished set of v registers (for some $v \geq 1$) are designated as the output registers. A v-tuple $(l_1, l_2, \ldots, l_v) \in N^v$ is generated if M eventually halts and the contents of the output registers are $l_1, l_2, \ldots, l_v$ respectively.

Without loss of generality we may assume that in the halting configuration all the registers, except the output ones, are empty. We also assume (without loss of generality) that the output registers are non-decreasing and their contents is only incremented (i.e., the output registers are never the subject of SUB instructions, but only of ADD instructions).

We will refer to a register machine with v-output registers (the other registers are auxiliary registers) as a v-output register machine. It is well known that a set S of v-tuples of numbers is generated by a v-output register machine if and only if S is recursively enumerable. When dealing with vectors of numbers, hence with the Parikh images of languages (or with sets of vectors generated/recognized by register machines), we write $PsRE$.

3 Spiking Neural P Systems with Cooperating Rules

We pass on now to introducing SN P systems with cooperating rules investigated in this paper.

Definition 1. [**SN P system with cooperating rules**] An SN P system with cooperating rules is an SN P system of degree $m \geq 1$ with $p \geq 1$ components, of the form

$$\Pi = (O, \Sigma, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m, syn, out), \text{ where}$$

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\Sigma = \{1, 2, \ldots, p\}$ is the label alphabet for components;
3. $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m$ are neurons, of the form

$$\sigma_i = (n_i, R_i),\ 1 \leq i \leq m;$$

   where
   (a) $n_i \geq 0$ is the *initial number of spikes* contained in the neuron;
   (b) $R_i = \cup_{l \in \Sigma} R_{il}$, where each R_{il}, $1 \leq l \leq p$, is a set (can be empty) of rules representing a component l in σ_i having rules of the following two forms:
      i. $E/a^r \to a$, where E is a regular expression over O, $r \geq 1$ (if $L(E) = a^r$, then we simply write $a^r \to a$);
      ii. $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \to a$ of type i. from R_{il};
4. $syn \subseteq \{1, 2, 3, \ldots, m\} \times \{1, 2, 3, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* among cells);
5. $out \in \{1, 2, 3, \ldots, m\}$ indicates the *output* neuron.

Because we do not need the delay between firing and spiking (i.e., rules of the form $E/a^r \to a; d$, with $d \geq 1$) as well as extended rules (i.e., rules of the form $E/a^r \to a^q$, with $q \geq 1$) in the proofs below, we do not consider these features in the definition, but such rules can be introduced in the usual way.

The rules of the type $E/a^r \to a$ are spiking rules, and can be applied only if the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$. When neuron σ_i spikes, its spike is replicated in such a way that one spike is sent immediately to all neurons σ_j such that $(i, j) \in syn$. The rules of type $a^s \to \lambda$ are forgetting rules; s spikes are simply removed ("forgotten") when applying such a rule. Like in the case of spiking rules, the left-hand side of a forgetting rule must "cover" the contents of the neuron, that is, $a^s \to \lambda$ is applied only if the neuron contains exactly s spikes. For simplicity, in the graphical representation of the system, the rules in the component l of neuron σ_i are prefixed with l and the components inside the neuron is separated by lines.

As defined above, each component of the neurons can contain several rules. More precisely, it is allowed to have two spiking rules $E_1/a^{r_1} \to a$ and $E_2/a^{r_2} \to a$ with $L(E_1) \cap L(E_2) \neq \emptyset$ in the same component. This leads to a non-deterministic way of using the rules and we cannot avoid the non-determinism (deterministic systems will compute only singleton sets).

The *configuration* of an SN P system is described by the number of spikes in each neuron. Thus, the initial configuration of the system is described as $\mathcal{C}_0 = \langle n_1, n_2, n_3, \ldots, n_m \rangle$.

The SN P system is synchronized by means of a global clock and works in a locally sequential and globally maximal manner with one component active at a step for the whole system. That is, the working is sequential at the level of each neuron. In each neuron, at each step, if there is more than one rule enabled from the active component by its current contents, then only one of them (chosen non-deterministically) can fire. But still, the system as a whole evolves in a parallel and synchronising way, as in each step, all the neurons (that have an enabled rule) choose a rule from the active component and all of them fire at once. Using the rules, the system passes from one configuration to another configuration; such a step is called a *transition*.

In a component-l-restricted transition, we say that the symbol 1 is generated if at least one rule with label l is used and a spike is sent out to the environment by the output neuron and the symbol 0 is generated if no spike is sent out to the environment. Similar to CD grammar systems, several cooperation strategies among the components can be considered: we here consider only the five basic ones.

For two configurations $\mathcal{C}$ and $\mathcal{C}'$, we write $\mathcal{C} \Longrightarrow_l^* \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^{=k} \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^{\leq k} \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^{\geq k} \mathcal{C}'$, $\mathcal{C} \Longrightarrow_l^t \mathcal{C}'$, for some $k \geq 1$, if configuration $\mathcal{C}'$ can be reached from $\mathcal{C}$ as follows: (1) by any number of transitions, (2) by k transition steps, (3) by at most k transition steps, (4) by at least k transition steps, (5) by a sequence of transition steps using rules from lth component of each neuron, which cannot be continued, respectively.

A *computation* of Π is a finite or infinite sequence of transitions starting from the initial configuration, and every configuration appearing in such a sequence is called reachable. Therefore a finite (step) computation γ_α of Π in the mode $\alpha \in \{*, t\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$, is defined as $\gamma_\alpha = \mathcal{C}_0 \Longrightarrow_{j_1}^{\alpha} \mathcal{C}_1 \Longrightarrow_{j_2}^{\alpha} \ldots \Longrightarrow_{j_y}^{\alpha} \mathcal{C}_y$ for some $y \geq 1$, $1 \leq j_y \leq p$, where $\mathcal{C}_0$ is the initial configuration. A computation γ_α of Π halts when the system reaches a configuration where no rule can be used as per the cooperating protocol α (i.e., the SN P system has halted). This paper only works in the terminating mode, so for convenience the mode is not explicitly used in the definitions hereafter.

With any computation, halting or not, we associate a spike train: a sequence of digits 0 and 1, with 1 appearing at positions corresponding to those steps when the output neuron sent a spike out of the system. With a spike train we can associate various numbers, which can be considered as generated by an SN P system. For instance, the distance in time between the first two spikes, between all consecutive spikes, the total number of spikes (in the case of halting computations), and so on.

It is clear that the standard SN P system introduced in [5] is a special case of the cooperating SN P system where the number of components is one. Similar to the standard SN P system, there are various ways of using this device. In the generative mode, one of the neurons is considered as the output neuron and the

spikes of the output neuron are sent to the environment. SN P systems can also be used for generating sets of vectors, by considering several output neurons, $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_v}$. In this case, the system is called a v-output SN P system. Here a vector of numbers, $(n_1, n_2, \ldots, n_v)$, is said to be generated by the system if n_j is the number corresponding to the spike train from σ_{i_j}, where $1 \leq j \leq v$.

We denote by $C_pN^{max}_{gen}(\Pi)$ $[C_pPs^{max}_{gen}(\Pi)]$ the set of numbers [of vectors, resp.] generated by a p-component SN P system Π in a maximally parallel manner, and by $NC_pSpik_2P^{max}_m(\beta)$ $[PsC_pSpik_2P^{max}_m(\beta)]$, $\beta \in \{gene, unb, boun\}$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by cooperating SN P systems of type β (*gene* stands for general, *unb* for unbounded, *boun* for bounded), with at most m neurons and p components. When m is not bounded, it is replaced by $*$. The subscript 2 reminds us of the fact that we count the number of steps elapsed between the first two spikes.

An SN P system can also work in the accepting mode: a neuron is designated as the input neuron and two spikes are introduced in it at an interval of n steps; input n is encoded by $2n$ in the input register; the number n is accepted if the computation halts.

In the asynchronous case, in each time unit, any neuron is free to use a rule or not. Even if enabled, a rule is not necessarily applied, the neuron can remain still in spite of the fact that it contains rules which are enabled by its contents. If the contents of the neuron are not changed, a rule which was enabled in a step t can fire later. If new spikes are received, then it is possible that other rules will be enabled – and applied or not. This way of using the rules also applies to the output neuron, hence now the distance in time between the spikes sent out by the system is no longer relevant. That is why, for asynchronous SN P systems we take as the result of a computation the total number of spikes sent out; this, in turn, makes necessary considering only halting computations (the computations never halting are ignored, they provide no output). We denote by $C_pN^{nsyn}_{gen}(\Pi)$ $[C_pPs^{nsyn}_{gen}(\Pi)]$ the set of numbers [of vectors, resp.] generated by an asynchronous cooperating SN P system Π with p components, and by $NC_pSpik_{tot}P^{nsyn}_m(\beta)$ $[PsC_pSpik_{tot}P^{nsyn}_m(\beta)]$, $\beta \in \{gene, unb, boun\}$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by an asynchronous cooperating SN P systems of type β, with at most m neurons and p components. When m is not bounded, it is replaced by $*$. The subscript *tot* reminds us of the fact that we count all spikes sent to the environment.

In the strongly sequential case, in each neuron, in each time unit, at least one neuron contains a fireable rule and exactly one of them is chosen to fire non-deterministically. Here, the output can be interpreted in any of the earlier suggested ways. In this paper, we consider the distance in time between the first two spikes. We denote by $C_pN^{sseq}_{gen}(\Pi)$ $[C_pPs^{sseq}_{gen}(\Pi)]$ the set of numbers [of vectors, resp.] generated by a strongly sequential cooperating SN P system Π, and by $NC_pSpik_2P^{sseq}_m(\beta)$ $[PsC_pSpik_2P^{sseq}_m(\beta)]$, $\beta \in \{gene, unb, boun\}$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by strongly sequential cooperating SN P systems of type β, with at most m neurons and p components. When m is not bounded, it is replaced by $*$.

4 Computational completeness of asynchronous SN P systems with two components using standard rules

We pass now to prove that the power of two components in standard neurons (where standard rules, producing one spike at a time, are used) can compensate for the loss of power entailed by removing the synchronization.

Theorem 1. $NC_2Spik_{tot}P_*^{nsyn}(gene) = NRE$.

Proof. We only have to prove the inclusion $NRE \subseteq NC_2Spik_{tot}P_*^{nsyn}(gene)$, and to this aim, we use the characterization of NRE by means of register machines used in the generating mode.

Let $M = (m, H, l_0, l_h, I)$ be a register machine with m registers, having the properties specified above: the result of a computation is the number stored in register 1 at the end of the computation and this register is never decremented during the computation.

What we want is an asynchronous SN P system with two components Π which (1) simulates the register machine M, and (2) has its output neuron emitting the number of spikes equal to the number computed by M.

Instead of specifying all technical details of the construction, we present the three main types of modules of the system Π, with the neurons, their rules, and their synapses represented graphically. In turn, simulating M means to simulate the ADD instructions and the SUB instructions. Thus, we will have one type of module associated with ADD instructions, one associated with SUB instructions, and one dealing with the spiking of the output neuron (a FIN module). The modules of the three types are given in Figs. 1, 2 and 3 respectively.

For each register r of M, we consider a neuron σ_r in Π whose contents corresponds to the contents of the register. Specifically, if the register r holds the number $n > 0$, then the neuron σ_r will contain $2n$ spikes. With each label l_i of an instruction in M, we also associate a neuron σ_{l_i} with a single rule $a \to a$ in its first component and no rules in its second component. There are also some auxiliary neurons $\sigma_{l_i,q}$, $q = 1, 2, 3, \ldots$, thus precisely identified by label l_i. Initially, all these neurons are empty, with the exception of the neuron σ_{l_0} associated with the start label of M, which contains a single spike. This means that this neuron is activated. During the computation, the neuron σ_{l_i} which receives a spike will become active in its first component. Thus, simulating an instruction $l_i : (OP(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the register r as requested by OP, then introducing a spike in one of the neurons σ_{l_j}, σ_{l_k} which becomes active in this way. When activating the neuron σ_{l_h}, associated with the halting label of M, the computation in M is completely simulated in Π; we will then send to the environment a number of spikes equal to the number stored in the register 1 of M. Neuron σ_1 is the output neuron of the system.

Simulating $l_i : (ADD(r), l_j, l_k)$ (module ADD in Fig. 1).

The initial instruction, that labelled l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an instruction $l_i : (ADD(r), l_j, l_k)$,

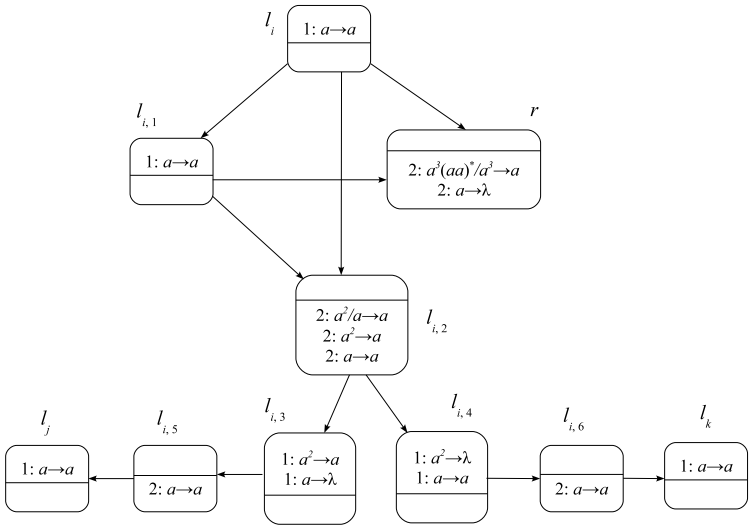

Fig. 1. ADD module: simulation of $l_i : (ADD(r), l_j, l_k)$

with a spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spikes in any other neurons, except in those associated with registers. Even if the system is in the second component at the time, it must switch over to the first component, since we are working in the terminating mode and there are no rules in the second component which are currently applicable anywhere in the system.

Having a spike inside and now in the first component, neuron σ_{l_i} can fire, and at some time it will do it, producing a spike. This spike will simultaneously go to neurons σ_r, $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$. The neurons σ_r and $\sigma_{l_{i,2}}$ cannot spike because the firing rules are present in their second components. The neuron $\sigma_{l_{i,1}}$ will spike at some time, then a spike will simultaneously go to the neurons σ_r and $\sigma_{l_{i,2}}$. Since no rules are enabled in the first component, the system switches to the second component. Before the system switches, the neuron σ_r receives two spikes from σ_{l_i} and $\sigma_{l_{i,1}}$, thus simulating the increase of the value of register r with 1. Now the system is in the second component. The neuron $\sigma_{l_{i,2}}$ has two spikes and it can fire by choosing one of its rules $a^2/a \to a$ or $a^2 \to a$ non-deterministically. If the neuron $\sigma_{l_{i,2}}$ uses its first rule $a^2/a \to a$, then it consumes one spike and sends a spike to each of the neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$. The neuron $\sigma_{l_{i,2}}$ is left with one spike and thus it has an enabled rule $a \to a$. The system switches to the first component only when no rules are enabled in the neuron $\sigma_{l_{i,2}}$. When the neuron $\sigma_{l_{i,2}}$ fires for the second time, neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ receive another spike and the system switches to the first component. The neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ have enabled rules and they can fire. The system will be in the first component as long as enabled rules are present in $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$. After some time, the neuron $\sigma_{l_{i,3}}$ uses it spiking rule and sends a spike to $\sigma_{l_{i,5}}$ and the neuron $\sigma_{l_{i,4}}$ forgets its

spikes. So eventually neuron $\sigma_{l_{i,5}}$ fires and sends a spike to σ_{l_j}, thus activating it.

If the neuron $\sigma_{l_{i,2}}$ uses its second rule $a^2 \to a$, then each of the neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ receive one spike finally. After some time, the neuron $\sigma_{l_{i,4}}$ uses it spiking rule and sends a spike to $\sigma_{l_{i,6}}$ and the neuron $\sigma_{l_{i,5}}$ forgets its spikes. So after some time, neuron $\sigma_{l_{i,6}}$ fires and sends a spike to σ_{l_k}, thus activating it. Therefore, from the firing of neuron σ_{l_i}, the system adds two spikes to neuron σ_r and non-deterministically fires one of the neurons σ_{l_j} and σ_{l_k}. Consequently, the simulation of the ADD instruction is possible in Π.

Simulating $l_i : (SUB(r), l_j, l_k)$ (module SUB in Fig. 2).

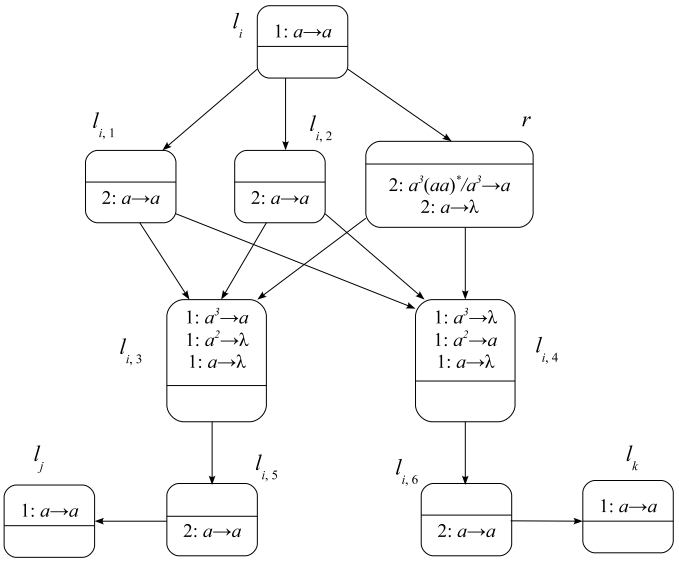

Fig. 2. SUB module: simulation of $l_i : (SUB(r), l_j, l_k)$

Let us now examine Fig. 2, starting from the situation of having a spike in neuron σ_{l_i} and no spike in other neurons, except neurons associated with registers; assume that neuron σ_r holds a number of spikes of the form $2n, n \geq 0$. Sometime, the neuron σ_{l_i} will fire and a spike goes immediately to each of the neurons $\sigma_{l_{i,1}}$, $\sigma_{l_{i,2}}$ and σ_r. The system must switch over to the second component, since we are working in the terminating mode and there are no rules in the first component which are currently applicable anywhere in the system.

If σ_r does not contain any spikes to begin with (this corresponds to the case when register r is empty), then eventually the spike sent by σ_{l_i} gets forgotten by virtue of the rule $a \to \lambda$ and σ_r is again left with no spikes, indicating that it is still zero. Eventually, neurons $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$ also send spikes using their rule $a \to a$. Thus, neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ end up with 2 spikes each and the system switches to the first component. After some steps, $\sigma_{l_{i,3}}$ forgets the two spikes

through the rule $a^2 \to \lambda$ and the neuron $\sigma_{l_{i,4}}$ fires using its rule $a^2 \to a$. With no rules applicable in the first component, the system switches to the second component and eventually neuron $\sigma_{l_{i,6}}$ sends a spike to neuron σ_{l_k}, as required, thus finishing the simulation of the SUB instruction for the case when register r is empty.

If neuron σ_r has $2n$ spikes to begin with, where $n \geq 1$, then after some steps, the rule $a(aa)^+/a^3 \to a$ is used in σ_r. Hence, σ_r now has two spikes less that what it began indicating that r has been reduced by 1. Further, neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ end up with 3 spikes each. After some steps, $\sigma_{l_{i,4}}$ forgets the three spikes through the rule $a^3 \to \lambda$ and the neuron $\sigma_{l_{i,3}}$ fires using its rule $a^3 \to a$. With no rules applicable in the first component, the system switches to the second component and eventually neuron $\sigma_{l_{i,5}}$ sends a spike to neuron σ_{l_j}, thus completing the simulation of the decrement case of the SUB instruction.

What remains to be examined is the possible interference between SUB modules. Note that there may be more than a single SUB instruction involving the same register r. Assume that we simulate the instruction $l_i : (SUB(r), l_j, l_k)$, hence neuron σ_r sends a spike to all neurons of the form $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$ for which there is an instruction $l'_i : (SUB(r), l'_j, l'_k)$ in M. These spikes will be forgotten using the rule $a \to \lambda$ and this is the correct continuation of the computation. Note that the system will be in the first component as long as any spikes are present in the neurons of the form $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$. Thus, the neurons $\sigma_{l_{i,5}}$ and $\sigma_{l_{i,6}}$ will become active only after the forgetting rule $a \to \lambda$ is applied in each neuron of the form $\sigma_{l_{i',3}}$ and $\sigma_{l_{i',4}}$.

This means that the simulation of the SUB instruction is correct, we started from l_i and ended in l_j if the register was non-empty (and we decreased it by one), and in l_k if the register was empty.

Simulating $l_h : (HALT)$ (module FIN in Fig. 3).

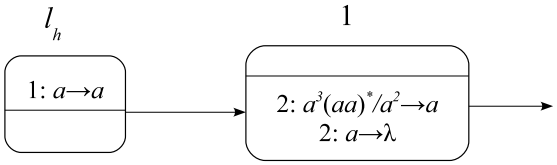

Fig. 3. FIN module: simulation of $l_h : HALT$

When the neuron σ_{l_h} is activated, it (eventually) sends one spike to neuron σ_1, corresponding to the register 1 of M. From now on, this neuron can fire, and it sends out one spike for each two spikes present in it, hence the system will emit a number of spikes which corresponds to the contents of the register 1 of M at the end of the computation (after reaching the instruction $l_h : HALT$).

Consequently, $C_2N^{nsyn}_{gen}(\Pi) = N(M)$ and this completes the proof. □

Clearly, the previous construction is the same for the accepting mode, and can be carried out for deterministic register machines (the ADD instructions are of the form $l_i : (ADD(r), l_j)$. Similarly, if the result of a computation is defined as the number of spikes present in a specified neuron in the halting configuration, then the previous construction is the same, we only have to add one further neuron which is designated as the output neuron and which collects all spikes emitted by neuron σ_1.

Theorem 1 can easily be extended by allowing more output neurons and then simulating a v-output register machine, producing in this way sets of vectors of natural numbers.

Theorem 2. $PsC_2Spik_{tot}P_*^{nsyn}(gene) = PsRE.$

5 Sequential spiking neural P systems with two components

In this section, we restrict the model to operate in a sequential manner. Before considering the power of sequential SN P systems with two components, we first recall some results from [4] on the power of the sequential SN P systems with one component.

1. Sequential SN P systems with general neurons are universal.
2. Sequential SN P systems with unbounded neurons are universal.
3. Strongly sequential SN P systems with general neurons are universal.
4. Strongly sequential SN P systems with unbounded neurons are not universal.

The paper [4] makes use of delayed rules to achieve synchronization. Here the synchronization can be achieved by switching between the components and hence delayed rules are not required. Here we prove that two component strongly sequential SN P systems with standard unbounded neurons without any delay are computationally complete.

Theorem 3. $NC_2Spik_2P_*^{sseq}(unb) = NRE.$

Proof. Given some register machine M generating a set $N(M)$, we can simulate M with a strongly sequential unbounded SN P Π having two components which generates the set $C_2N_{gen}^{seq}(\Pi) = \{x \mid x \in N(M)\}$. The SN P Π's initial configuration will again start with the initial configuration for each module along with a single spike in neuron σ_{l_0}.

To create a strongly sequential unbounded SN P generating exactly $N(M)$ use the same ideas and methods given in Theorem 1. The ADD module is the same as the one shown in Fig. 1 but we remove the rule $2 : a \to \lambda$ from the neuron σ_r so that the subsystem becomes unbounded. Since no rules from σ_r are fired in the ADD module, the subsystem works correctly even in the sequential mode.

The new subtraction module is shown in Fig. 4. It is initiated with a single spike in neuron σ_{l_i} which immediately sends a spike to neurons σ_r, $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$

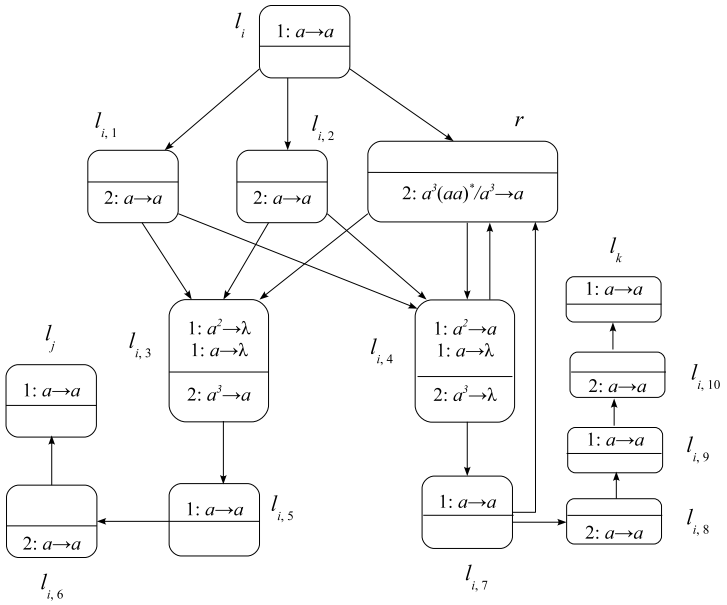

Fig. 4. Strongly sequential unbounded two component SN P SUB module

at time $t+1$ (where t is the time the initial spike is sent to neuron σ_{l_i}). If the value in the register r is not zero then the three neurons non-deterministically spike during the next three steps (time $t+2$, $t+3$, and $t+4$). This causes neuron $\sigma_{l_{i,3}}$ to spike and neuron $\sigma_{l_{i,4}}$ to forget sequentially during the following two time steps (time $t+5$ and $t+6$). Since no rules are enabled in the second component, the system switches to the first component. In the step $t+7$, neuron $\sigma_{l_{i,5}}$ fires and sends a spike to $\sigma_{l_{i,6}}$. Since neuron σ_r sends spikes to all neurons $\sigma_{l_{i'},3}$ and $\sigma_{l_{i'},4}$ where $l_{i'} : (SUB(r), l_{j'}, l_{k'})$, these neurons receive a single spike during the computation of instruction l_i. These spikes must be forgotten before the next instruction executes. Here, the system switches to the second component and fires the rule in the neuron $\sigma_{l_{i,6}}$ only after all spikes are removed from the neurons $\sigma_{l_{i'},3}$ and $\sigma_{l_{i'},4}$ using their forgetting rule $a \to \lambda$ present in their first components. When the neuron $\sigma_{l_{i,6}}$ fires, it initiates the instruction module l_j.

If σ_r does not contain any spikes to begin with (this corresponds to the case when register r is empty), then the neuron σ_r does not fire and the neurons $\sigma_{l_{i,3}}$ and $\sigma_{l_{i,4}}$ receive two spikes each. Since no rules are enabled in the second component, the system switches to the first component. This causes neurons $\sigma_{l_{i,3}}$ to forget and $\sigma_{l_{i,4}}$ to spike sequentially during the following two time steps (time $t+4$ and $t+5$). The spike from $\sigma_{l_{i,4}}$ goes simultaneously to neurons σ_r and $\sigma_{l_{i,7}}$ in time step $t+6$. Neuron $\sigma_{l_{i,7}}$ sends a spike to σ_r and $\sigma_{l_{i,8}}$ in time step $t+7$. Since no rules are enabled in the first component, the system switches to the second component. Now the neuron σ_r has three spikes. The neurons σ_r and

$\sigma_{l_i,8}$ fire sequentially in the next two steps $t+8$ and $t+9$. Thus the contents of σ_r is cleared indicating that r remains zero, as required. The spike from σ_r goes to $\sigma_{l_i,3}$, $\sigma_{l_i,4}$ and all neurons $\sigma_{l_{i'},4}$, where $l_{i'} : (SUB(r), l_{j'}, l_{k'})$. The neurons $\sigma_{l_i,9}$ and $\sigma_{l_i,10}$ with rules in different components ensures that the spikes in $\sigma_{l_i,3}$, $\sigma_{l_i,4}$ and all $\sigma_{l_{i'},3}$ and $\sigma_{l_{i'},4}$ are forgotten before enabling the instruction module l_k as the spike received by $\sigma_{l_i,8}$ (from $\sigma_{l_i,7}$) percolates through $\sigma_{l_i,9}$ and $\sigma_{l_i,10}$ to l_k.

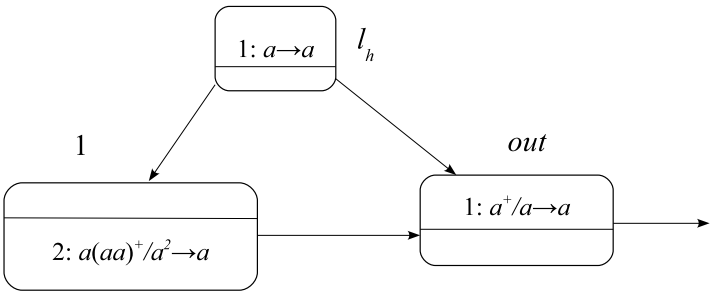

Fig. 5. Strongly sequential unbounded two component SN P output module

To simulate $l_h : (HALT)$, we create the module given in Fig. 5. When neuron l_h receives a spike, it fires and sends a spike to neurons σ_1 and σ_{out} with the system in the first component (it will switch to the first component even otherwise as only rules in the first component are enabled and we are working in the terminating mode). Let t be the moment when neuron l_h fires. Suppose the number stored in the register 1 of M is n.

At step $t+1$, neuron σ_{out} fires for the first time sending its spike to the environment. The number of steps from this spike to the next one is the number computed by the system. Since no rule is enabled in the first component, the system switches to the second component. Now the neuron σ_1 spikes during the next n steps. The neuron σ_{out} will become active only after $2n$ spikes are removed from σ_1. So at time $t+n+1$, the system again switches to the first component and the neuron σ_{out} fires for the second time. The interval between the two spikes emitted by σ_{out} is $(t+n+1)-(t+1)=n$, which is the number stored in the register 1 of M. The system halts after $n-1$ steps with all neurons empty except neuron σ_1 which contains a spike. □

Theorem 3 can easily be extended by allowing more output neurons and then simulating a v-output register machine, producing in this way sets of vectors of natural numbers.

Theorem 4. $PsC_2Spik_2P_*^{sseq}(unb) = PsRE$.

One more observation is that the module given in Fig. 4 works even if the system is asynchronous. It is now possible to construct a new system with ADD module shown in Fig. 1 without the rule $2 : a \to \lambda$ in the neuron σ_r, the SUB

module given in Fig. 4 and the FIN module given in Fig. 3 without the rule $2 : a \to \lambda$ in the neuron σ_1 which would be unbounded and work correctly in the case of an asynchronous system. Hence, we have the following two theorems.

Theorem 5. $NC_2Spik_{tot}P_*^{nsyn}(unb) = NRE.$

Theorem 6. $PsC_2Spik_{tot}P_*^{nsyn}(unb) = PsRE.$

Finally, the system constructed in Section 4 with the FIN module in Fig. 5 would work for sequential systems. Hence, we have the following two theorems.

Theorem 7. $NC_2Spik_2P_*^{sseq}(gene) = NRE.$

Theorem 8. $PsC_2Spik_2P_*^{sseq}(gene) = PsRE.$

6 Conclusion and discussion

The usual SN P systems operate in a maximally parallel manner. This model was shown to be computationally complete even with a variety of additional restrictions on the rule types [8, 11]. In this paper, we introduced a spiking neural P system with cooperating rules. Computational completeness has been proved for asynchronous as well as sequential cooperating SN P systems with two components using unbounded as well as general neurons working in the terminating mode. This suggests that cooperating SN P systems are indeed more powerful by offering seamless synchronization without the use of any delays. Further work would include the construction of small universal systems. It would also be interesting to consider the languages generated by these systems using different number of components. Further, this paper considers only the terminating mode, which is known to be more powerful than others in the case of CD grammar systems. A discussion on if the same result holds for cooperating SN P systems working in other models would be worthwhile.

Acknowledgements The work was supported by EU project Development of Research Capacities of the Silesian University in Opava (CZ.1.07/2.3.00/30.0007) and European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

1. Cavaliere, M., Ibarra, O.H., Păun, Gh., Egecioglu, Ö., Ionescu, M., Woodworth, S.: Asynchronous spiking neural P systems, *Theoretical Computer Science*, **410** (24-25), 2352–2364 (2009).
2. Csuhaj-Varjú, E., Dassow, J.: On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics (EIK)*, **26**, 49–63 (1990).
3. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, Gh.: Grammar Systems. A Grammatical Approach to Distribution and Cooperation, Gordon and Breach, London, (1994).

4. Ibarra, O.H., Woodworth, S., Yu, F., Păun, A.: On spiking neural P systems and partially blind counter machines, *Natural Computing* **7**(1), 3–19 (2008).
5. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems, *Fundamenta Informaticae*, **71**, 279–308 (2006).
6. Meersman, R., Rozenberg, G.: Cooperating grammar systems, Proceedings of Mathematical Foundations of Computer Science, LNCS **64**, 364–374 (1978).
7. Minsky, M.: Computation – Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, NJ, (1967).
8. Pan, L., Păun, Gh.: Spiking Neural P Systems: An Improved Normal Form, *Theoretical Computer Science*, **411**, 906–918 (2010).
9. Păun, Gh., Rozenberg, G., Salomaa, A. (eds): Handbook of Membrane Computing, Oxford University Press, Oxford (2010).
10. Rozenberg, G., Salomaa, A. (eds): Handbook of Formal Languages. 3 volumes, Springer, Berlin, (1997).
11. Zeng, X., Zhang, X., Pan, L.: Homogeneous Spiking Neural P Systems, *Fundamenta Informaticae*, **97**, 1–20 (2009).