Small Universal Spiking Neural P Systems with Cooperating Rules

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 paper considers spiking neural P systems (SN P systems) with cooperating rules where each neuron has a finite number of sets of rules. Each set is called a component and each rule from a component has the same component label. 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. By using 59 neurons, a small universal SN P system with two components, working in terminating mode, is constructed for computing functions.

1 Introduction

Cooperating distributed grammar systems (CDGS) were introduced in [1] to model the so–called blackboard type of problem solving architectures. A CD grammar system consists of several component grammars, these are the problem solving agents, which generate a common sentential form by taking turns in the rewriting process. The sentential form represents the blackboard, the current state of the problem, which the agents might modify according to a certain protocol until a terminal string is generated. CD grammar systems with context-free components working in the cooperation protocol called terminal mode (or t-mode) of derivation are more powerful than context-free grammars; they characterize the class of ET0L languages, the languages generated by so-called extended tabled interactionless Lindenmayer systems.

The concept of cooperation and distribution as known from the CD grammar systems is introduced to spiking neural P systems [4]. Spiking neural P systems [2] 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 can be used as a computing device in various ways. Most of the previous research on SN P systems focused on three ways: as number generating/computing devices, as language generators, and as devices for computing functions.

A k-component SN P system with cooperating rules is represented as a directed graph where nodes correspond to the neurons with the initial spikes mentioned in it; the input neuron has an incoming arrow and the output neuron has an outgoing arrow, suggesting their communication with the environment. Each neuron has k components (which can be empty) separated by lines. Each non-empty component has a finite number of spiking and forgetting rules that involve the spikes present in the neuron in the form of occurrences of a symbol a. The arcs indicate the synapses between the neurons. Using spiking rules, the information in a certain neuron can be sent to its neighbouring neurons in the form of spikes, which are accumulated at the target neurons. When we use a forgetting rule in a certain neuron, a specified number of spikes will be removed from the neuron. The application of each rule is determined by the current active component and checking the contents (number of spikes) of the neuron. The rules in the system are uniquely labelled but for the sake of simplicity, we attach only the component label. The rules inside the same component share the same component label. In each time unit, the number of spikes in each neuron contribute to the configuration of the system.

Generally, in an SN P system, a global clock is assumed to mark the time of the whole system. SN P systems work in a synchronous manner, that is, one rule must be applied for each neuron from the list of applicable rules at any step. The different neurons work in parallel. Using the rules in this way, we pass from one configuration of the system to another configuration; such a step is called a transition. A computation is a finite or infinite sequence of transitions starting from the initial configuration. A computation halts if it reaches a configuration where no rule can be used. Note that the transition of configuration C is non-deterministic in the sense that there may be different rules applicable to C.

Spiking neural P systems with cooperating rules are based on cooperation among the components and passing of control between components in each neuron. At any step or in any sequence of steps (depends on mode of application) only one of the components is active for the whole system and one of the enabled rules from this component of each neuron can fire during a step (or steps). After that another (not necessarily different) component of each neuron becomes active. The way of passing active control is called protocol. Similar to the CD grammar systems, series of cooperation protocols among the components in neurons of an SN P system can been considered, where for example any component, once started, has to perform exactly k, at most k, at least k or an arbitrary number of transition steps. In the so-called t-mode, a component may stop working if and only if none of its rules is applicable. In $=1$ mode, the rules in the component of each neuron are used in exactly one step and then control is passed. Selection of the next active component is non-deterministic and only one component generates the output at a step, other components wait for passing control. This paper considers the SN P systems with two components working in the terminating mode.

Cooperating SN P systems are indeed more powerful by offering seamless synchronization without the use of any delays as seen in [4], where 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. In this paper, we take on one of the problems mentioned in [4].

Looking for small universal computing devices is a classical research topic in computer science, see, e.g., [3], and the references therein. This topic has been heavily investigated in the framework of SN P systems [6], where a universal SN P system with standard delayed rules was obtained by using 84 neurons for computing functions, and the system with 76 neurons can generate any set of Turing computable natural numbers. In [9], these results were improved: 67 neurons for standard delayed rules in the case of computing functions, and 63 neurons for standard rules in the case of generating sets of numbers. In this work, we investigate small universal SN P systems with two components (with standard rules, without delay) working in terminating mode. As devices of computing functions, we construct a universal SN P system with two components by using 59 neurons.

2 Universal Register Machines

We assume the reader to be familiar with formal language theory and membrane computing. The reader can find details about them in [8], [7] etc.

We pass now to introducing the universal register machines. Because the register machines used in the following sections are deterministic, we only recall the definition of this type of machines. A deterministic register machine 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. When it is useful, a label can be seen as a state of the machine, l_0 being the initial state, l_h the final/accepting state.
The labelled instructions are of the following forms:

1. l_i: (ADD(r), l_j) (add 1 to register r and then go to the instruction with label l_j),
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 0 (we assume that the registers are always numbered from 0 to $m-1$) at that time is said to be generated by M. It is known (see, [5]) that register machines generate all sets of numbers which are Turing computable.

A register machine can also compute any Turing computable function: we introduce the arguments $n_1, n_2, \ldots, n_k$ in specified registers r_1, $r_2, \ldots, r_k$ (without loss of the generality, we may assume that we use the first k registers), we start with the instruction with label l_0, and if we stop (with the instruction with label l_h), then the value of the function is placed in another specified register, r_t, with all registers different from r_t being empty. The partial function computed in this way is denoted by $M(n_1, n_2, \ldots, n_k)$. In the computing form, it is known (see e.g., [5]) that the deterministic register machines are equivalent with Turing machines.

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

Fig. 1. A universal register machine M_u from Korec [3]

In [3], the register machines are used for computing functions, with universality defined as follows. Let $(\phi_0, \phi_1, \ldots)$ be a fixed admissible enumeration of the unary partial recursive functions. A register machine M_u is said to be universal if there is a recursive function g such that for all natural numbers x, y we have $\phi_x(y) = M_u(g(x), y)$. In [3], several universal register machines are constructed, with the input (the couple of numbers $g(x)$ and y) introduced in registers 1 and 2, and the result obtained in register 0. In the following, we consider the specific universal register machine $M_u = (8, H, l_0, l_h, I)$, with instructions (their labels constitute the set H) given in Fig. 1, which is also the one used in [6] (it has 8 registers numbered from 0 to 7 and 23 instructions).

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

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

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 cell;
   (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 write simply $a^r \to a$);
      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. $in, out \in \{1, 2, 3, \ldots, m\}$ indicates the *input* and *output* neurons, respectively.

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. 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 are 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, at 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 the 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 [2] 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. 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; the number n is accepted if the computation halts.

When both an input and an output neuron are considered, the system can be used as a transducer, both for strings and infinite sequences, as well as for computing numerical functions. Like in the case considered in [6], in order to compute a function $f : N^k \to N$, where N is the set of all non-negative integers, k natural numbers $n_1, \ldots, n_k$ are introduced into the system by "reading" from the environment a binary sequence $z = 10^{n_1-1}10^{n_2-1} \ldots 10^{n_k-1}1$. This means that the input neuron of Π receives a spike at each step corresponding to a digit 1 from string z and no spike otherwise. Note that $k+1$ spikes are exactly inputted; that is, it is assumed that no further spike is coming to the input neuron after the last spike.

We start from the initial configuration and we define the result of a computation as the number of steps between the first two spikes sent out by the output neuron. The result is 0 if no spikes exit the output neuron and the computation halts. The computations and the result of computations are defined in the same way as for usual SN P systems - the time distance between the first two spikes emitted by the system with the restriction that the system outputs exactly two spikes and halts (immediately after the second spike), hence it produces a spike train of the form $0^b10^{r-1}1$, for some $b \geq 0$ and with $r = f(n_1, \dots, n_k)$.

4 Small Universal Computing SN P Systems with Two Components Working in Terminating Mode

We proceed now to constructing a universal SN P system Π_u with cooperating rules, for computing functions. The system has two components and works in the terminating mode. To construct a universal SN P system Π_u, we follow the way used in [6] to simulate a universal register machine M_u.

Before the construction, a modification should be made in M_u because subtraction operation on neurons corresponding to the registers where the result is placed is not allowed in the construction from [2], but the register 0 of M_u is a subject of such operations. That is why a further register has to be added– labelled with 8–and the halt instruction of M_u should be replaced by the following instructions:
l_h: (SUB(0), l_{22}, l'_h), l_{22}: (ADD(8), l_h), l'_h: HALT.
In this way, the obtained register machine M'_u has 9 registers, 24 ADD and SUB instructions and 25 labels.

The usual way of simulating a register machine M'_u by an SN P system is the construction of an SN P system with cooperating rules Π_u, where neurons are associated with each register and with each label of an instruction of the machine. For each register r of M_u, we associate a neuron σ_r. If a register r contains a number n, then the associated neuron σ_r will contain $2n$ spikes. Starting with neurons σ_1 and σ_2 already loaded with $2g(x)$ and $2y$ spikes, respectively, and introducing two spikes in neuron l_0, we can compute in our system Π_u in the same way as the universal register machine M_u from Fig. 1; if the computation halts, then neuron σ_8 will contain $2\phi_x(y)$ number of spikes.

With each label l_i of an instruction in M'_u, we associate a neuron σ_{l_i} and some auxiliary neurons $\sigma_{l_{i,q}}$, $q = 1, 2, \dots$, thus precisely identified by label l_i. Specifically, modules ADD and SUB are constructed to simulate the instructions of M'_u. The modules are given in a graphical form in Figs. 3 and 4. In the initial configuration, all neurons of Π_u are empty. There are two additional tasks to solve: to introduce the mentioned spikes in the neurons σ_{l_0}, σ_1, σ_2, and to output the computed number.

The first task is covered by module INPUT presented in Fig. 2. After receiving the first spike from the environment, neuron σ_{in} starts in the second component and fires. Subsequently, neurons σ_{c_1} and σ_{c_2} send to neuron σ_1 as many pairs of spikes as one more than the number of steps between the first two input spikes, and after that they get "over flooded" by the second input spike and are blocked. In turn, neurons σ_{c_3} and σ_{c_4} start working only after collecting two spikes and stop working after receiving the third spike. No rule in the second component is applicable, so the systems switches to the first component and enables the firing of neuron σ_{c_5}. It sends two spikes to neuron σ_{l_0}, thus starting the simulation of M'_u. At that moment, neurons σ_1 and σ_2 are already loaded using spikes from the neurons σ_{c_1} through σ_{c_4}. Thus, at the end of the INPUT module, some of the neurons are still left with spikes, but this is not a cause for concern as those neurons will be nowhere reused.

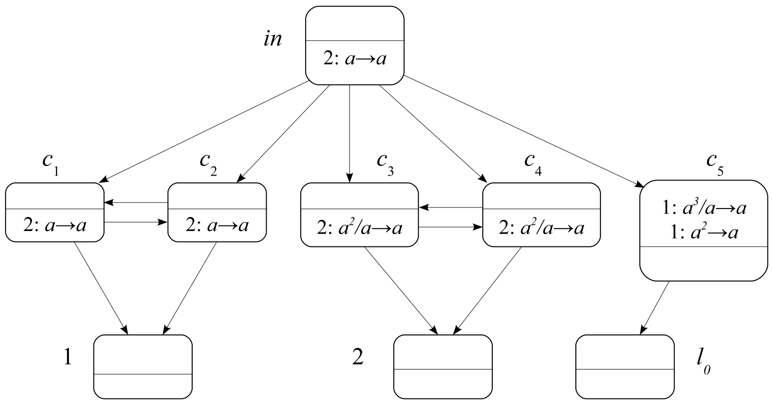

Fig. 2. INPUT module

In general, the simulation of an ADD or a SUB instruction starts by introducing two spikes in the neuron with the instruction label (we say that this neuron is activated). Modules as in Fig. 3 and Fig. 4 are associated with the ADD and the SUB instructions.

Simulating l_i: (ADD(r), l_j) (module ADD in Fig. 3).

Assume that we are in a step t when we have to simulate an instruction l_i: (ADD(r), l_j), with two spikes present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spikes in any other neuron, 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 two spikes inside and now in the first component, neuron σ_{l_i} fires using the rule $a^2/a \to a$ producing a spike. This spike will simultaneously go to neurons σ_r and σ_{l_j}. In step $t+1$, neuron σ_{l_i} fires again using the rule $a \to a$ and sends another spike to σ_r and σ_{l_j}. Note that there was

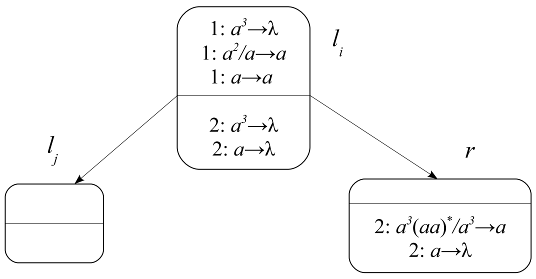

Fig. 3. Deterministic ADD module l_i: (ADD(r), l_j)

no switch in the component as the first component still had a rule applicable. Therefore, from the firing of neuron σ_{l_i}, the system adds two spikes each to neuron σ_r and σ_{l_j} and activates the neuron σ_{l_j}. Consequently, the simulation of the ADD instruction is possible in Π_u.

Another important point to note is that if l_j is also the label for an ADD instruction, then σ_{l_j} will fire in step $t + 1$ itself using the rule $a \to a$. This does not hamper the correctness of the module since the second spike will also reach σ_{l_j} in the next step and another spike will be sent out by using the same rule. If it is a SUB instruction, σ_{l_j} will not fire in step $t+1$ as there is no rule applicable in the first component of the neuron corresponding to a SUB instruction, as we will see in the SUB module simulation.

Simulating l_i: (SUB(r), l_j, l_k) (module SUB in Fig. 4).

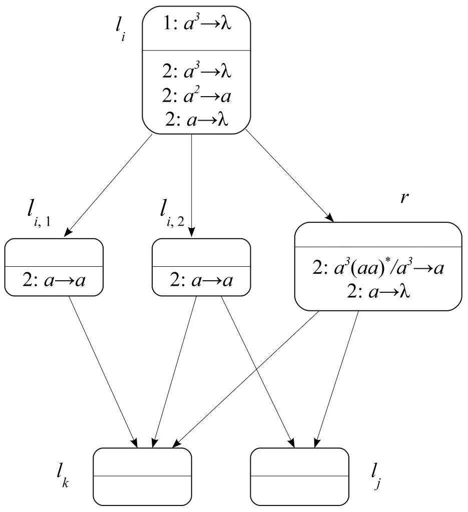

Fig. 4. SUB module: simulation of l_i: (SUB(r), l_j, l_k)

Assume that we are in a step t when we have to simulate an instruction l_i: (SUB(r), l_j, l_k), with two spikes present in neuron σ_{l_i} and no spikes in any other neurons, except in those associated with registers. Let us examine now Fig. 4, starting from the situation of having two spikes in neuron σ_{l_i} and neuron σ_r, which holds a certain number of spikes (this number is twice the value of the corresponding register r). Even if the system is in the first component at the time, it 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. A spike from neuron σ_{l_i} goes immediately to neurons $\sigma_{l_{i,1}}$, $\sigma_{l_{i,2}}$ and σ_r. If σ_r does not contain any spikes to begin with (this corresponds to the case when register r is empty), then in the step $t+1$, the spike sent by σ_{l_i} gets forgotten by virtue of the rule $a \rightarrow \lambda$ and σ_r is again left with no spikes, indicating that it is still zero. At the same time, neurons $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$ send spikes using the rule $a \rightarrow a$. Thus, neurons σ_{l_j} and σ_{l_k} end with 1 and 2 spikes respectively. In the subsequent step $t+2$, σ_{l_j} forgets the spike through the rule $a \rightarrow \lambda$. In the case of the neuron σ_{l_k}, if it corresponds to an ADD instruction, it will fire in the next step since the second component of the neuron corresponding to an ADD instruction has no rule applicable and the system cannot switch over to the first component as σ_{l_j} has an applicable rule. If it corresponds to a SUB instruction, it will fire in the same step and this does not create any issues as the operation is complete and the next one may begin. Thus the neuron σ_{l_k} gets activated, as required by simulating the SUB instruction.

If neuron σ_r has $2n$ spikes to begin with, where $n \geq 1$, then in the step $t+1$, the rule $a^3(aa)^*/a^3 \rightarrow a$ is used in σ_r and $a \rightarrow a$ is used in neurons $\sigma_{l_{i,1}}$ and $\sigma_{l_{i,2}}$, and hence the neurons σ_{l_j} and σ_{l_k} receive 2 and 3 spikes respectively. The neuron σ_r now has 2 spikes lesser than when it started out and hence we have achieved the decrement of the register r by 1. In the subsequent step $t+2$, σ_{l_k} forgets the spikes through the rule $a^3 \rightarrow \lambda$. In the case of the neuron σ_{l_j}, if it corresponds to an ADD instruction, it will fire in the next step since the second component of the neuron corresponding to an ADD instruction has no rule applicable and the system cannot switch over to the first component as σ_{l_k} has an applicable rule. If it corresponds to a SUB instruction, it will fire in the step $t+2$ and this does not create any problems as the operation is complete and the next one may begin. Thus the neuron σ_{l_j} gets activated, as required by simulating the SUB instruction.

Another important point to note is that in our construction, the neuron σ_r is sending a spike. Note that there may be more than a single SUB instruction involving the same register r. In that case, when σ_r sends a spike, it would be sent to not just σ_{l_j} and σ_{l_k} but also to target neurons of all SUB instructions involving r. This is handled since the second components of both ADD and SUB modules have the forgetting rule $a \rightarrow \lambda$. So in the same step where σ_{l_k} forgets its three spikes, those target neurons which received a spike unnecessarily will also forget their spike received from σ_r. Since σ_r does not send spikes when it started out with a zero value, we do not have any problem in that case.

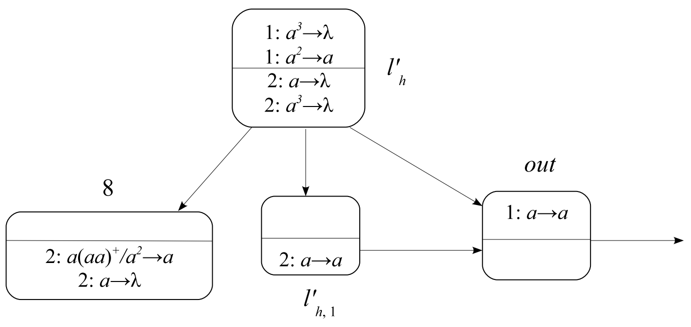

Fig. 5. Module OUTPUT

It is also important to note that the neurons σ_{l_i} associated with ADD instructions are different from those associated with SUB instructions: in the first case it starts firing after receiving either one spike or two spikes, in the latter case the neuron fires only after receiving two spikes.

Having the result of the computation in register 8, which is never decremented during the computation, we can output the result by means of the module OUTPUT from Figure 5. When neuron l'_h receives two spikes, it fires and sends a spike to neurons σ_8, $\sigma_{l'_{h,1}}$ 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 8 of M'_u 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 function of the number computed by the system. Since no rules are enabled in the first component, the system switches to the second component. Now the two neurons σ_8 and $\sigma_{l'_{h,1}}$ spike during the next $n+1$ steps (σ_8 would fire $n+1$ times and $\sigma_{l'_{h,1}}$ would fire for one time). The neuron σ_{out} will become active only after $2n+1$ spikes are removed from σ_8. So at time $t+n+3$, the system again switches to the first component and the neuron σ_{out} fires for the second time. In this way, we get the spike train $..10^{n+1}1$, encoding the number $\phi_x(y)$ as the result of the computation. The overall design of the system is given in Fig. 6.

Thus, the system Π_u has

- 9 neurons for the 9 registers,
- 25 neurons for the 25 labels,
- 28 neurons for 14 SUB instructions,
- 6 neurons in the INPUT module,
- 2 neurons in the OUTPUT module,

which comes to a total of 70 neurons. This number can be slightly decreased, by some "code optimization", exploiting some particularities of the register machine M'_u.

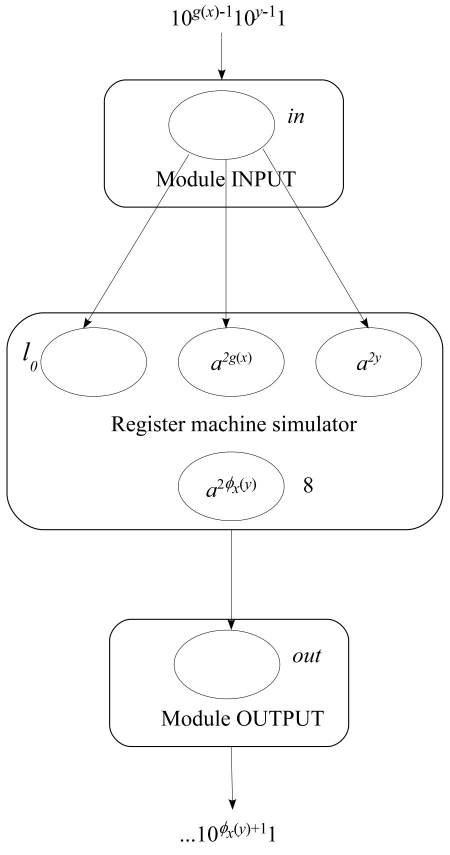

Fig. 6. The general design of the universal SN P system

First, let us observe that the sequence of two consecutive ADD instructions l_{17}: (ADD(2), l_{21}), l_{21}: (ADD(3), l_{18}), without any other instruction addressing the label l_{21}, can be simulated by merging the modules for these two instructions and eliminating the neuron $\sigma_{l_{21}}$, and in this way we save the neuron associated with l_{21}.

If the two SUB instructions address different registers, then they can share one auxiliary neuron, as shown in Fig. 7. The working of any particular instruction is as described above. The only difference is that when one of the instructions executes, a spike is sent to the target neuron of another SUB instruction. Since the second components of both ADD and SUB modules have the forgetting rule $a \to \lambda$, those target neurons which received a spike will forget their spike received from $\sigma_{l_{i_1},2}$.
By using the results as above, the 14 SUB instructions can be classified to four groups:

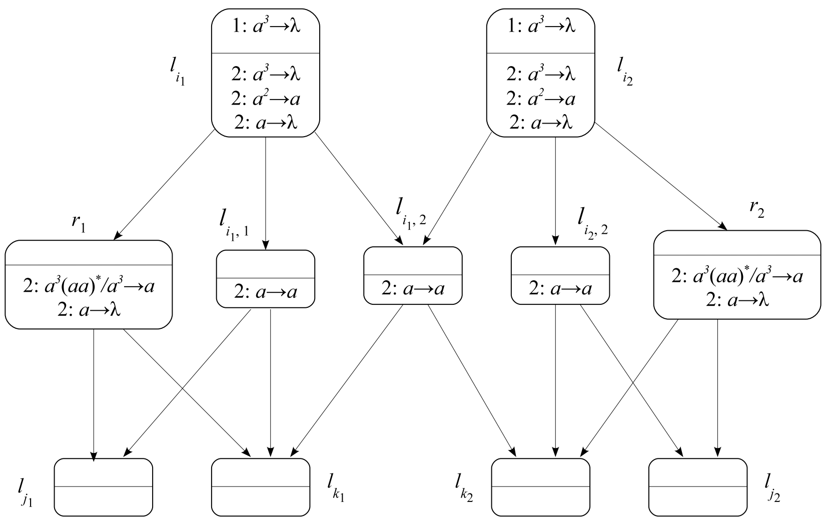

Fig. 7. A module simulating SUB-SUB instructions

1. l_0: (SUB(1), l_1, l_2), l_3: (SUB(5), l_2, l_4), l_4: (SUB(6), l_5, l_3), l_6: (SUB(7), l_7, l_8), l_{10}: (SUB(4), l_0, l_{11}), l_{13}: (SUB(2), l_{18}, l_{19}), l_{15}: (SUB(3), l_{18}, l_{20}), l_{19}: (SUB(0), l_0, l_{18});
2. l_8: (SUB(6), l_9, l_0), l_{11}: (SUB(5), l_{12}, l_{13}), l_{18}: (SUB(4), l_0, l_h), l_h: (SUB(0), l_{22}, l'_h);
3. l_{12} : (SUB(5), l_{14}, l_{15});
4. l_{14} : (SUB(5), l_{16}, l_{17}).

All modules associated with the instructions in each group can share one auxiliary neuron. In this way, 7 neurons are saved from the first group, 3 neurons from the second group.

Merging ADD and SUB instructions does not seem readily possible. This is because the ADD module has no auxiliary neurons and merging the neurons corresponding to the instruction labels is also not possible as the rules in the first component are different and the rules in the second component cannot be omitted. We have already considered the optimization of merging two consecutive ADD instructions into one and this is possible.

Overall 11 neurons are saved, thus an improvement is achieved from 70 to 59 neurons. We state this result in the form of a theorem in order to stress its importance.

Theorem 1. *There exists a universal spiking neural P system with two components working in terminating mode having 59 neurons for computing functions.*

5 Conclusion

Starting from the definition of spiking neural P systems and following the idea of cooperating distributed grammar systems, we have proposed a class of spiking neural P systems with cooperating rules for which we have constructed a small universal computing system with two components working in the terminating mode. The system constructed in this work has 59 neurons. This number can be reduced by using more components, for instance, the number of auxiliary neurons in the SUB module can be brought down to one by using four components. Thus, further work could include smaller universal systems using more components and perhaps working in different modes.

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. Csuhaj-Varju, E., Dassow, J.: On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics (EIK)*, **26**, 49–63 (1990).
2. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems, *Fundamenta Informaticae*, **71**, 279–308 (2006).
3. Korec, I.: Small universal Turing machines, *Theoretical Computer Science*, **168**, 267–301 (1996).
4. Metta, V. P., Raghuraman, S., Krithivasan, K.: Spiking neural P systems with cooperating rules, Conference on membrane computing (CMC 15), August 20 -22, 2014, Prague, Czech Republic.
5. Minsky, M.: Computation – finite and infinite machines, Prentice Hall, Englewood Cliffs, NJ, (1967).
6. Păun, A., Păun, Gh.: Small universal spiking neural P systems, *BioSystems*, **90**(1), 48–60 (2007).
7. Păun, Gh., Rozenberg, G., Salomaa, A. (eds) Handbook of Membrane Computing. Oxford University Press, Oxford (2010).
8. Rozenberg, G., A. Salomaa, A. (eds): Handbook of Formal Languages. 3 volumes, Springer, Berlin, (1997).
9. Zhang, X., Zeng, X., Pan, L.: Smaller universal spiking neural P systems, *Fundamenta Informaticae*, **87**, 117–136 (2008).