Spiking neural P systems with extended rules: universality and languages

Haiming Chen · Mihai Ionescu · Tseren-Onolt Ishdorj · Andrei Păun · Gheorghe Păun · Mario J. Pérez-Jiménez

Abstract We consider spiking neural P systems with rules allowed to introduce zero, one, or more spikes at the same time. The motivation comes both from constructing small universal systems and from generating strings; previous results from these areas are briefly recalled. Then, the computing power of the obtained systems is investigated, when considering them as number generating and as language generating devices. In the first case, a simpler proof of universality is obtained, while in the latter case we find characterizations of finite and recursively enumerable languages (without using any squeezing mechanism, as it was necessary in the case of standard rules). The relationships with regular languages are also investigated.

Keywords Membrane computing · Spiking neural P systems · Turing computability · Universality · Chomsky hierarchy

H. Chen
Computer Science Laboratory, Institute of Software, Chinese Academy of Sciences,
100080 Beijing, China
e-mail: chm@ios.ac.cn

M. Ionescu
Universitat Rovira i Virgili, Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
e-mail: armandmihai.ionescu@urv.net

T.-O. Ishdorj · G. Păun · M. J. Pérez-Jiménez
Department of Computer Science and AI, University of Sevilla, Avda Reina Mercedes s/n,
41012 Sevilla, Spain
e-mail: marper@us.es

T.-O. Ishdorj
Computational Biomodelling Laboratory, TUCS, Abo Akademi University, Turku 20520, Finland
e-mail: tishdorj@abo.fi; tseren@yahoo.com

1 Introduction

We combine here two ideas recently considered in the study of the spiking neural P systems (in short, SN P systems), namely the *extended* rules from (Păun and Păun to appear) and the *string generation* from Chen et al. (2006). Spiking neural P systems were recently introduced in Ionescu et al. (2006) as an tempt to incorporate in membrane computing [see a comprehensive introduction in Păun (2002) and current information in the webpage http://www.psystems.disco.unimib.it ideas from neural computing by spiking [as found, for instance, in Maass (2002) and Maass and Bishop (1999)].

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (arcs of the graph), under the control of firing rules. 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 the string corresponding to the sequence of spikes in the spike train. This sequence is a binary one, with 0 associated with a step when no spike is emitted and 1 associated with a step when a spike is emitted.

The case of SN P systems as number generators was investigated in several papers, starting with (Ionescu et al. 2006), where it is proved that such systems are Turing complete [hence also universal SN P systems exist, because the proof is constructive; universality in a rigorous framework was investigated in Păun and Păun (to appear)]. In turn, the string case is investigated in Chen et al. (2006), where representations of finite, regular, and recursively enumerable languages were obtained, but also finite languages were found which cannot be generated in this way.

Here we consider an extension of the rules, already used in Păun and Păun (to appear), namely we allow rules of the form $E/a^c \rightarrow a^p$, with the following meaning: if the content of the neuron is described by the regular expression E, then c spikes are consumed and p are produced and sent to the neurons to which there exist synapses leaving the neuron where the rule is applied. Thus, these rules cover and generalize at the same time both spiking rules and forgetting rules as considered so far in this area—with the mentioning that we do not also consider here a delay between firing and spiking, because in the proofs we never need such a delay.

As expected, this generalization allows much simpler constructions for the proof of Turing completeness in the case of considering SN P systems as number generators (we treat this issue in Sect. ``Extended SN P systems as number generators''). More interesting is the case of strings produced by SN P systems with extended rules: we associate a symbol b_i to a step when the system sends i spikes into the environment,

A. Păun
Department of Computer Science, Louisiana Tech University, RustonPO Box 10348, LA 71272, USA
e-mail: apaun@latech.edu

A. Păun
Faculdad de Informatíca, Universidad Politécnica de Madrid – UPM, Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain

G. Păun (✉)
Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 Bucharest, Romania
e-mail: george.paun@imar.ro; gpaun@us.es

with two possible cases—b_0 is used as a separated symbol, or it is replaced by λ (sending no spike outside is interpreted as a step when the generated string is not grown). The first case is again restrictive: not all minimal linear languages can be obtained, but still results stronger than those from Chen et al. (2006) can be proved in the new framework because of the possibility of removing spikes under the control of regular expressions—see Sect. ‘‘Languages in the restricted case’’. The freedom provided by the existence of steps when we have no output makes possible direct characterizations of finite and recursively enumerable languages [not only representations, as obtained in Chen et al. (2006) for the standard binary case]—Sect. ‘‘Languages in the non-restricted case’’.

2 Formal language theory prerequisites

We assume the reader to be familiar with basic language and automata theory, e.g., from Rozenberg and Salomaa (1997) and Salomaa (1973), so that we introduce here only some notations and notions used latter in the paper.

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 non-empty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}*, \{a\}^+$. If $x = a_1, a_2, \ldots, a_n$, $a_i \in V$, $1 \leq i \leq n$, then $mi(x) = a_n, \ldots, a_2a_1$.

A morphism $h : V_1^* \rightarrow V_1^*$ such that $h(a) \in \{a, \lambda\}$ for each $a \in V_1$ is called a projection, and a morphism $h : V_1^* \rightarrow V_2^*$ such that $h(a) \in V_2 \cup \{\lambda\}$ for each $a \in V_1$ is called a weak coding.

If $L_1, L_2 \subseteq V^*$ are two languages, the left and right quotients of L_1 with respect to L_2 are defined by $L_2 \backslash L_1 = \{w \in V^* \mid xw \in L_1 \text{ for some } x \in L_2\}$, and, respectively, $L_1 / L_2 = \{w \in V^* \mid wx \in L_1 \text{ for some } x \in L_2\}$. When the language L_2 is a singleton, these operations are called left and right derivatives, and denoted by $\partial_x^l(L) = \{x\} \backslash L$ and $\partial_x^r(L) = L/\{x\}$, respectively.

A Chomsky grammar is given in the form $G = (N,T,S,P)$, where N is the non-terminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the finite set of rules. For regular grammars, the rules are of the form $A \rightarrow aB, A \rightarrow a$, for some $A, B \in N, a \in T$.

We denote by *FIN*, *REG*, *CF*, *CS*, *RE* the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages; by *MAT* we denote the family of languages generated by matrix grammars without appearance checking. The family of Turing computable sets of numbers is denoted by *NRE* (these sets are length sets of RE languages, hence the notation).

Let $V = \{b_1, b_2, \ldots, b_m\}$, for some $m \geq 1$. For a string $x \in V^*$, let us denote by $val_m(x)$ the value in base $m + 1$ of x (we use base $m + 1$ in order to consider the symbols $b_1, \ldots, b_m$ as digits 1, 2,..., m, thus avoiding the digit 0 in the left hand of the string). We extend this notation in the natural way to sets of strings.

All universality results of the paper are based on the notion of a register machine. 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 (labeling 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 instructions are of the following forms:

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j,l_k non-deterministically chosen),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (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),
- l_h : $\mathtt{HALT}$ (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 at that time is said to be generated by M. (Without loss of generality we may assume that in the halting configuration all other registers are empty; also, we may assume that register 1 is never subject of SUB instructions, but only of ADD instructions.) It is known [see, e.g., Minsky (1967)] that register machines generate all sets of numbers which are Turing computable.

A register machine can also be used as a number accepting device: we introduce a number n in some register r_0, we start working with the instruction with label l_0, and if the machine eventually halts, then n is accepted (we may also assume that all registers are empty in the halting configuration). Again, accepting register machines characterize *NRE*.

Furthermore, register machines can compute all Turing computable functions: we introduce the numbers n_1,...,n_k in some specified registers r_1,...,r_k, we start with the instruction with label l_0, and when we stop (with the instruction with label l_h) the value of the function is placed in another specified register, r_t, with all registers different from r_t being empty. Without loss of generality we may assume that $r_1, \ldots, r_k$ are the first k registers of M, and then the result of the computation is denoted by $M(n_1, \ldots, n_k)$.

In both the accepting and the computing case, the register machine can be *deterministic*, i.e., with the ADD instructions of the form $l_i : (\mathtt{ADD}(r), l_j)$ (add 1 to register r and then go to the instruction with label l_j).

In the following sections, when comparing the power of two language generating/ accepting devices the empty string λ is ignored.

3 Spiking neural P systems with extended rules

We directly introduce the type of SN P systems we investigate in this paper; the reader can find details about the standard definition in Ionescu et al. (2006), Păun and Pérez-Jiménez (2006) and Chen et al. (2006), etc.

An *extended spiking neural P system* (abbreviated as *extended SN P system*), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

where

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ 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 σ_i;

(b) R_i is a finite set of *rules* of the form $E/a^c \rightarrow a^p$, where E is a regular expression over a and $c \geq 1$, $p \geq 0$, with the restriction $c \geq p$;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i,j) \in syn, 1 \leq i,j \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the*output neuron*(σ_{i_0}) of the system.

A rule $E/a^c \rightarrow a^p$ is applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule can *fire*, and its application means consuming (removing) c spikes (thus only $k-c$ remain in σ_i) and producing p spikes, which will exit immediately the neuron. A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

Note that we do not consider here a delay between firing and spiking (i.e., rules of the form $E/a^c \rightarrow a^p; d$, with $d \geq 0$; when using such a rule, the spikes are sent to the receiving neurons after d steps and during these steps the sending neuron is idle, no rule is used in it and no spike can enter it), because we do not need this feature in the proofs below, but such a delay can be introduced in the usual way. (As a consequence, here the neurons are always open.)

If a rule $E/a^c \rightarrow a^p$ has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a^p$. If all rules $E/a^c \rightarrow a^p$ have $L(E) = \{a^c\}$, then we say that the system is *finite*.

The spikes emitted by a neuron σ_i go to all neurons σ_j such that $(i,j) \in syn$, i.e., if σ_i has used a rule $E/a^c \rightarrow a^p$, then each neuron σ_j receives p spikes.

If several rules can be used at the same time, then the one to be applied is chosen non-deterministically.

During the computation, a configuration of the system is described by the number of spikes present in each neuron; thus, the initial configuration is described by the numbers $n_1, n_2, \ldots, n_m$.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule can be used. With any computation (halting or not) we associate a *spike train*, the sequence of symbols 0 and 1 describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0 (note that at this stage we ignore the number of spikes emitted by the output neuron into the environment in each step, but this additional information will be considered below).

As the result of a computation, in Ionescu et al. (2006) and Păun and Pérez-Jiménez (2006) one considers the distance between two consecutive steps when there are spikes which exit the system, with many possible variants: taking the distance between the first two occurrences of 1 in the spike train, between all consecutive occurrences, considering only alternately the intervals between occurrences of 1, etc. For simplicity, we consider here only the first case mentioned above: we denote by $N_2(\Pi)$ the set of numbers generated by an SN P system in the form of the number of steps between the first two steps of a computation when spikes are emitted into environment, and by $Spik_2SN^eP_m(rule_k, cons_p, prod_q)$ the family of sets $N_2(\Pi)$ generated by SN P systems with at most m neurons, at most k rules in each neuron, consuming at most p and producing at most q spikes. Any of these parameters is replaced by * if it is not bounded. (The superscript e points out the fact that we work with *extended* rules. When using only standard rules, this superscript is omitted, together with $prod_q$, which is by definition always $prod_1$.)

An SN P system can also be used in the *accepted* mode: a number is introduced in the form of the distance between two spikes entering the system and it is accepted if the computation eventually halts. Moreover, we can have *computing* SN P systems: a number is introduced as the distance between two input spikes and the result of the computation is a number provided as the distance between two output spikes.

Following (Chen et al. 2006) we can also consider as the result of a computation the spike train itself, thus associating a language with an SN P system. Specifically, like in Chen et al. (2006), we can consider the language $L_{bin}(\Pi)$ of all binary strings associated with halting computations in Π: the digit 1 is associated with a step when one or more spikes exit the output neuron, and 0 is associated with a step when no spike is emitted by the output neuron. We denote $B = \{0,1\}$.

Because several spikes can exit at the same time, we can also work on an arbitrary alphabet: let us associate the symbol b_i with a step when the output neuron emits i spikes. We have two cases: interpreting b_0 (hence a step when no spike is emitted) as a symbol or as the empty string. In the first case we denote the generated language by $L_{res}(\Pi)$ (with ''res'' coming from ''restricted''), in the latter one we write $L_{\lambda}(\Pi)$.

The respective families are denoted by $L_{\alpha}SN^{e}P_{m}(rule_k, cons_p, prod_q)$, where $\alpha \in \{bin, res, \lambda\}$ and parameters m,k,p,q are as above. We omit the superscript e and the parameter $prod_q$ when working with standard rules.

We recall from Chen et al. (2006) the following results

Theorem 3.1 (i) *There are finite languages (for instance,* $\{0^k, 10^j\}$, *for any* $k \geq 1, j \geq 0$*) which cannot be generated by any SN P system, but for any* $L \in FIN$, $L \subseteq B^+$, *we have* $L\{1\} \in L_{bin}SNP_1(rule_*, cons_*)$, *and if* $L = \{x_1, x_2, \ldots, x_n\}$, *then we also have* $\{0^{i+3}x_i \mid 1 \leq i \leq n\} \in L_{bin}SNP_*(rule_*, cons_1)$.

(ii) *The family of languages generated by finite non-extended SN P systems is strictly included in the family of regular languages over the binary alphabet, but for any regular language* $L \subseteq V^*$ *there is a finite SN P system* Π *and a morphism* $h : V^* \rightarrow B^*$ *such that* $L = h^{-1}(L(\Pi))$.

(iii) $L_{bin}SNP_*(rule_*, cons_*) \subset REC$, *but for every alphabet* $V = \{a_1, a_2, \ldots, a_k\}$ *there are a morphism* $h_1 : (V \cup \{b, c\})^* \rightarrow B^*$ *and a projection* $h_2 : (V \cup \{b, c\})^* \rightarrow V^*$ *such that for each language* $L \subseteq V^*$, $L \in RE$, *there is an SN P system* Π *such that* $L = h_2(h_1^{-1}(L(\Pi)))$.

These results show that the language generating power of non-extended SN P systems is rather eccentric; on the one hand, finite languages (like $\{0, 1\}$) cannot be generated, on the other hand, we can represent any RE language as the direct morphic image of an inverse morphic image of a language generated in this way. This eccentricity is due mainly to the restricted way of generating strings, with one symbol added in each computation step, and this naturally suggests the idea of extended rules, with the possibility of having λ as output in steps when no spike exits the system. As we will see below, this possibility considerably enlarges the generated families of languages.

4 Extended SN P systems as number generators

Because non-extended SN P systems—without delay; see Ibarra et al. (2006)—are already computationally universal, this result is directly valid also for extended systems. However, the construction on which the proof is based is much simpler in the extended case and it is also instructive for the way the small universal systems are found, that is why we briefly present it.

Theorem 4.1 $NRE = Spik_2SN^eP_*(rule_4, cons_5, prod_2)$.

Proof The proof of the similar result from Ionescu et al. (2006) is based on constructing an SN P system Π which simulates a given register machine M. The idea is that each register r has associated a neuron σ_r, with the value n of the register represented by $2n$ spikes in neuron σ_r. Also, each label of M has a neuron in Π, which is ''activated'' when receiving two spikes. We do not recall other details from Ionescu et al. (2006), and we pass directly to presenting—in Figs. 1–3—modules for simulating the ADD and the SUB instructions of M, as well as an OUTPUT module, in the case of using extended rules.

Because the neurons associated with labels of ADD and SUB instructions have to produce different numbers of spikes, in the neurons associated with ''output'' labels of instructions we have written the rules in the form $a^2 \rightarrow a^{\delta(l)}$, with $\delta(l) = 1$ for l being the label of a SUB instruction and $\delta(l) = 2$ if l is the label of an ADD instruction.

Because l_i precisely identifies the instruction, the neurons $c_{i\alpha}$ are distinct for distinct instructions. However, an interference between SUB modules appears in the case of instructions SUB which operate on the same register r: synapses $(r, c_{is}), (r, c_{i's}), s = 4, 5$, exist for different instructions $l_i : (\texttt{SUB}(r), l_j, l_k)$, $l_{i'} : (\texttt{SUB}(r), l_{j'}, l_{k'})$. Neurons $\sigma_{c_{i'4}}, \sigma_{c_{i'5}}$ receive 1 or 2 spikes from σ_r even when simulating the instruction with label l_i, but they are immediately forgotten (this is the role of rules $a \rightarrow \lambda, a^2 \rightarrow \lambda$ from neurons $\sigma_{c_{i4}}, \sigma_{c_{i5}}$ from Fig. 2).

The task of checking the functioning of the modules from Figs. 1–3 is left to the reader. □

5 Small universal SN P systems

In both the generating and the accepting case, SN P systems are computationally complete, they compute the Turing computable sets of numbers. Like the proof of the previous theorem, all the proofs from Ionescu et al. (2006), Păun and Pérez-Jiménez (2006) are also based on simulating register machines. Because there are universal register machines, in this way we can also find universal SN P systems. For instance, universal register machines are constructed in Korec (1996), with the universality defined as follows. Let $(\varphi_0, \varphi_1, \ldots)$

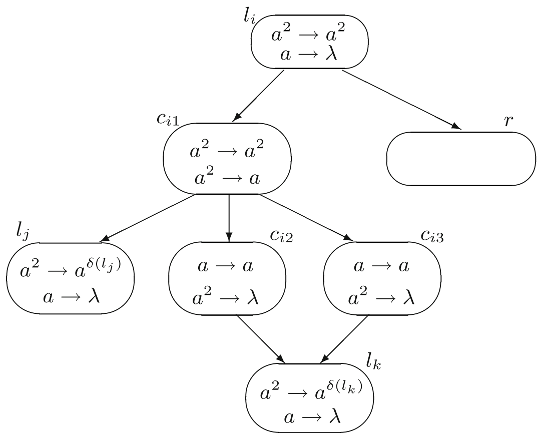


Fig. 1 Module ADD, for simulating an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$

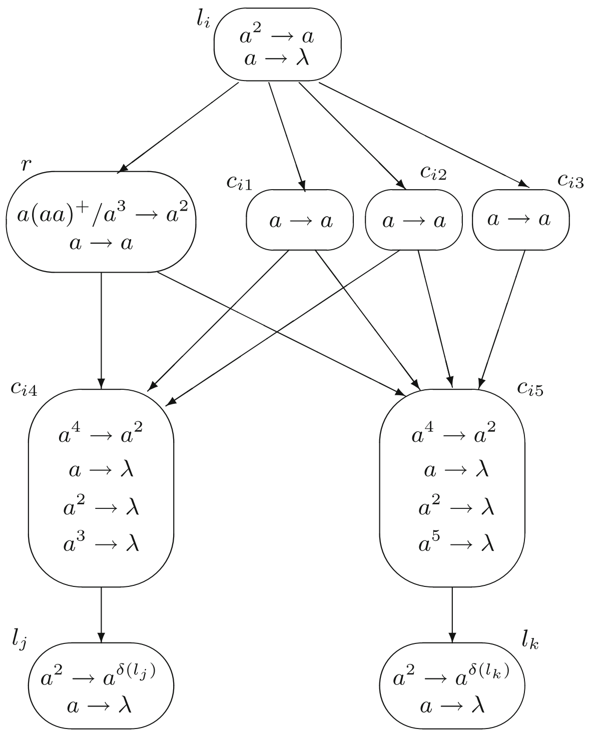


Fig. 2 Module SUB, for simulating an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$

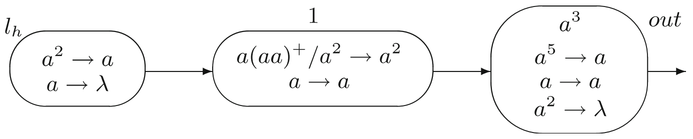


Fig. 3 Module OUTPUT

be a fixed admissible enumeration of the set of 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 $\varphi_x(y) = M_u(g(x), y)$. In Korec (1996), the input is introduced in registers 1 and 2, and the result is obtained in register 0 of the machine. By simulating the universal machine U_{32} from Korec (1996) (it has 8 registers and, in our setup, 23 instructions), with a special care paid to saving neurons always when possible, we get the following results:

Theorem 5.1 Păun and Păun (to appear) *There is a universal standard SN P system with 84 neurons and one with extended rules having 49 neurons.*

In the case when we consider an SN P system as a number generating device, we say that an SN P system Π_u is universal if, given a fixed admissible enumeration of the unary partial recursive functions, $(\varphi_0, \varphi_1, \ldots)$, there is a recursive function g such that for each natural number x, if we input the number $g(x)$ in Π_u, by ‘‘reading’’ the sequence $10^{g(x)-1}1$

from the environment, the set of numbers generated by the system is equal to $\{n \in \mathbf{N} \mid \varphi_x(n)$ is defined$\}$. Otherwise stated, after introducing the ''code'' $g(x)$ of the partial recursive function φ_x in a specified neuron, the system generates (hence halts sometimes after sending two spikes out) all numbers n for which $\varphi_x(n)$ is defined.

The universal system proceeds then as follows:

1. Read the string $10^{g(x)-1}1$ from the environment and load $2g(x)$ spikes in neuron σ_1.
2. Load neuron σ_2 non-deterministically with an arbitrary natural number n (in the case of using restricted rules this means to introduce $2n$ spikes in neuron σ_2 and in the case of extended rules means to introduce $6n$ spikes); at the same time, output the spike train $10^{n-1}1$ (hence the number n).
3. Check whether the function φ_x is defined for n. To this aim, start the register machine U_{32} from Korec (1996), with $g(x)$ in register 1 and n in register 2. If the computation in U_{32} halts, then also the computation in our SN P system halts, hence n is introduced in the set of generated numbers.

By implementing this strategy, we get the following results:

Theorem 5.2 Păun and Păun (to appear) *There is a universal number generating SN P system with standard rules having 76 neurons and one with extended rules having 50 neurons.*

As one can see, the extended rules are always useful, the number of neurons is much smaller for this case in each theorem.

6 Languages in the restricted case

We pass now to considering extended SN P systems as language generators, starting with the restricted case, when the system outputs a symbol in each computation step.

In all considerations below, we work with the alphabet $V = \{b_1, b_2, \ldots, b_s\}$, for some $s \geq 1$. By a simple renaming of symbols, we may assume that any given language L over an alphabet with at most s symbols is a language over V. When a symbol b_0 is also used, it is supposed that $b_0 \notin V$.

6.1 A characterization of FIN

As we have seen before, SN P systems with standard rules cannot generate all finite languages, but extended rules help in this respect.

Lemma 6.1 $L_\alpha SN^e P_1(rule_*, cons_*, prod_*) \subseteq FIN, \alpha \in \{res, \lambda\}$.

Proof In each step, the number of spikes present in a system with only one neuron decreases by at least one, hence any computation lasts at most as many steps as the number of spikes present in the system at the beginning. Thus, the generated strings have a bounded length. □

Lemma 6.2 $FIN \subseteq L_\alpha SN^e P_1(rule_*, cons_*, prod_*), \alpha \in \{res, \lambda\}$.

Proof Let $L = \{x_1, x_2, \ldots, x_n\} \subseteq V^*, n \geq 1$, be a finite language, and let$x_i = x_{i,1} \ldots x_{i,r_i}$ for $x_{i,j} \in V$, $1 \leq i \leq n$, $1 \leq j \leq$ $r_i = |x_i|$. Denote $l = \max\{r_i \mid 1 \leq i \leq n\}$. For $b \in V$, define $index(b) = i$ if $b = b_i$. Define $\alpha_j = ls\sum_{i=1}^{j} |x_i|$, for all $1 \leq j \leq n$.

An SN P system that generates L is shown in Fig. 4.

Initially, only a rule $a^{\alpha_n+ls}/a^{\alpha_n-\alpha_j+s} \to a^{index(x_{j,1})}$ can be used, and in this way we non-deterministically choose the string x_j to generate. This rule outputs the necessary number of spikes for $x_{j,1}$. Then, because $\alpha_j + (l-1)s$ spikes remain in the neuron, we have to continue with rules $a^{\alpha_j+(l-t+1)s}/a^s \to a^{index(x_{j,t})}$, for $t = 2$, and then for the respective $t = 3,4,\ldots,r_j-1$; in this way we introduce $x_{j,t}$, for all $t = 2,3,\ldots,r_j-1$. In the end, the rule $a^{\alpha_j+(l-r_j+1)s} \to a^{index(x_{j,r_j})}$ is used, which produces x_{j,r_j} and concludes the computation.

It is easy to see that the rules which are used in the generation of a string x_j cannot be used in the generation of a string x_k with $k \neq j$. Also, in each rule the number of spikes consumed is not less than the number of spikes produced. The system Π never outputs zero spikes, hence $L_{res}(\Pi) = L_\lambda(\Pi) = L$. □

Theorem 6.1 $FIN = L_\alpha\, SN^eP_1(rule_*, cons_*, prod_*), \alpha \in \{res, \lambda\}$.

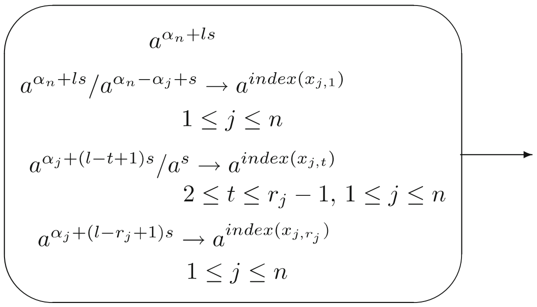
This characterization is sharp in what concerns the number of neurons, because of the following result:

Proposition 6.1 $L_\alpha SN^eP_2(rule_2, cons_3, prod_3) - FIN \neq \emptyset, \alpha \in \{res, \lambda\}$.

Proof The SN P system Π from Fig. 5 generates the infinite language $L_{res}(\Pi) = L_\lambda(\Pi) = b_3^*b_1\{b_1, b_3\}$. □

6.2 Representations of regular languages

Such representations are obtained in Chen et al. (2006) starting from languages of the form $L_{bin}(\Pi)$, but in the extended SN P systems, regular languages can be represented in an easier and more direct way.



a^{α_n+ls}
$a^{\alpha_n+ls}/a^{\alpha_n-\alpha_j+s} \to a^{index(x_{j,1})}$
$1 \leq j \leq n$
$a^{\alpha_j+(l-t+1)s}/a^s \to a^{index(x_{j,t})}$
$2 \leq t \leq r_j - 1,\ 1 \leq j \leq n$
$a^{\alpha_j+(l-r_j+1)s} \to a^{index(x_{j,r_j})}$
$1 \leq j \leq n$

Fig. 4 An SN P system generating a finite language

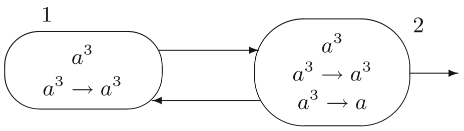


Fig. 5 An SN P system generating an infinite language

Theorem 6.2 *If $L \subseteq V*$, $L \in REG$, then $\{b_0\}L \in L_{res}$ $SN^eP_4(rule_*,cons_*,prod_*)$.*

Proof Consider a regular grammar $G = (N,V,S,P)$ such that $L = L(G)$, where $N = \{A_1,A_2,...,A_n\}$, $n \geq 1, S = A_n$, and the rules in P are of the forms $A_i \rightarrow b_kA_j$, $A_i \rightarrow b_k$, $1 \leq i, j \leq n$, $1 \leq k \leq s$.

Then $\{b_0\}L$ can be generated by the SN P system shown in Fig. 6.

In each step, neuron σ_2 (with the help of neuron σ_1) will send $n + s$ spikes to neuron σ_3, provided that they receive spikes from neuron σ_3. Neuron σ_3 fires in the first step by a rule $a^{2n+s}/a^{2n-j+s} \rightarrow a^k$ (or $a^{2n+s} \rightarrow a^k$) associated with a rule $A_n \rightarrow b_kA_j$ (or $A_n \rightarrow b_k$) from P, produces k spikes and receives $n + s$ spikes from neuron σ_2. In the meantime neuron σ_4 does not spike, hence it produces the symbol b_0, and receives spikes from neuron σ_3, therefore in the second step it generates the first symbol of the string.

Assume in some step t, the rule $a^{n+i+s}/a^{n+i-j+s} \rightarrow a^k$, for $A_i \rightarrow b_kA_j$, or $a^{n+i+s} \rightarrow a^k$, for $A_i \rightarrow b_k$, is used, for some $1 \leq i \leq n$, and $n + s$ spikes are received from neuron σ_2.

If the first rule is used, then k spikes are produced, $n + i - j + s$ spikes are consumed and j spikes remain in neuron σ_3. Then in step $t + 1$, we have $n + j + s$ spikes in neuron σ_3, and a rule for $A_j \rightarrow b_kA_l$ or $A_j \rightarrow b_k$ can be used. In step $t + 1$ neuron σ_3 also receives $n + m$ spikes from σ_2. In this way, the computation continues, unless the second rule is used.

If the second rule is used, then k spikes are produced, all spikes are consumed, and $n + m$ spikes are received in neuron σ_3. Then, in the next time step, neuron σ_3 receives $n + m$ spikes, but no rule can be used, so no spike is produced. At the same time, neuron σ_4 fires using spikes received from neuron σ_3 in the previous step, and then the computation halts.

In this way, all the strings in $\{b_0\}L$ can be generated. □

Corollary 6.1 *Every language $L \in REG, L \subseteq V^*$, can be written in the form $L = \partial^l_{b\_0}(L')$ for some $L' \in L_{res}$ $SN^eP_4(rule_*,cons_*,prod_*)$.*

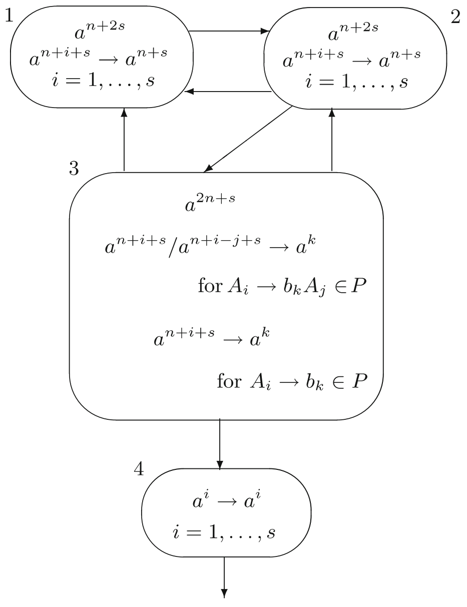

1
a^{n+2s}
$a^{n+i+s} \rightarrow a^{n+s}$
$i = 1, \ldots, s$
2
a^{n+2s}
$a^{n+i+s} \rightarrow a^{n+s}$
$i = 1, \ldots, s$
3
a^{2n+s}
$a^{n+i+s}/a^{n+i-j+s} \rightarrow a^k$
for $A_i \rightarrow b_kA_j \in P$
$a^{n+i+s} \rightarrow a^k$
for $A_i \rightarrow b_k \in P$
4
$a^i \rightarrow a^i$
$i = 1, \ldots, s$

Fig. 6 The SN P system from the proof of Theorem 6.2

One neuron in the previous representation can be saved, by adding the extra symbol in the right hand end of the string.

Theorem 6.3 *If $L \subseteq V^*, L \in REG$, then $L\{b_0\} \in L_{res}\ SN^eP_3(rule_*, cons_*, prod_*)$.*

Proof The proof is based on a construction similar to the one from the proof of Theorem 6.2. Specifically, starting from a regular grammar G as above, we construct a system Π as in Fig. 7, for which we have $L_{res}(\Pi) = L\{b_0\}$. We leave the task to check this assertion to the reader. □

Corollary 6.2 *Every language $L \in REG, L \subseteq V*$, can be written in the form $L = \partial^r_{b_0}(L')$ for some $L' \in L_{res}\ SN^eP_3(rule_*, cons_*, prod_*)$.*

6.3 Going beyond REG

We do not know whether the additional symbol b_0 can be avoided in the previous theorems (hence whether the regular languages can be directly generated by SN P systems in the restricted way), but such a result is not valid for the family of minimal linear languages (generated by linear grammars with only one non-terminal symbol).

Lemma 6.3 *The number of configurations reachable after n steps by an extended SN P system of degree m is bounded by a polynomial $g(n)$ of degree m.*

Proof Let us consider an extended SN P system $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$ of degree m, let n_0 be the total number of spikes present in the initial configuration of Π, and denote $\alpha = \max\{p \mid E/a^c \to a^p \in R_i, 1 \le i \le m\}$ (the maximal number of spikes produced by any of the rules of Π). In each step of a computation, each neuron σ_i consumes some c spikes and produces $p \le c$ spikes; these spikes are sent to all neurons σ_j such that $(i,j) \in syn$. There are at most $m-1$ synapses $(i,j) \in syn$, hence the p spikes produced by neuron σ_i are replicated in at most $p(m-1)$ spikes. We have $p(m-1) \le \alpha(m-1)$. Each neuron can do the same,

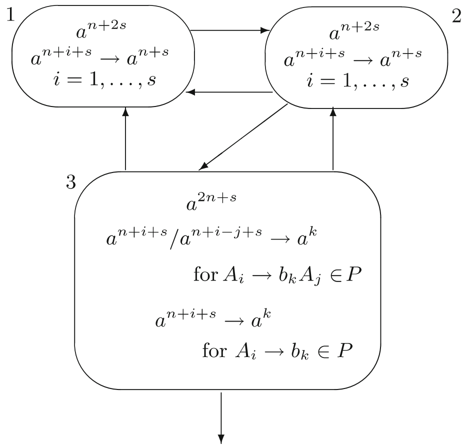


Fig. 7 The SN P system for the proof of Theorem 6.3

hence the maximal number of spikes produced in one step is at most $\alpha(m-1)m$. In n consecutive steps, this means at most $\alpha(m-1)mn$ spikes. Adding the initial n_0 spikes, this means that after any computation of n steps we have at most $n_0 + \alpha(m-1)mn$ spikes in Π. These spikes can be distributed in the m neurons in less that $(n_0 + \alpha(m-1)mn)^m$ different ways. This is a polynomial of degree m in n (α is a constant) which bounds from above the number of possible configurations obtained after computations of length n in Π. $\square$

Theorem 6.4 *If $f{:}V^+ \to V^+$ is an injective function, $card(V) \geq 2$, then there is no extended SN P system Π such that $L_f(V) = \{xf(x) \mid x \in V^+\} = L_{res}(\Pi)$.*

Proof Assume that there is an extended SN P system Π of degree m such that $L_{res}(\Pi) = L_f(V)$ for some f and V as in the statement of the theorem. According to the previous lemma, there are only polynomially many configurations of Π which can be reached after n steps. However, there are $card(V)^n \geq 2^n$ strings of length n in V^+. Therefore, for large enough n there are two strings $w_1, w_2 \in V^+$, $w_1 \neq w_2$, such that after n steps the system Π reaches the same configuration when generating the strings $w_1 f(w_1)$ and $w_2 f(w_2)$, hence after step n the system can continue any of the two computations. This means that also the strings $w_1 f(w_2)$ and $w_2 f(w_1)$ are in $L_{res}(\Pi)$. Due to the injectivity of f and the definition of $L_f(V)$ such strings are not in $L_f(V)$, hence the equality $L_f(V) = L_{res}(\Pi)$ is contradictory. $\square$

Corollary 6.3 *The following languages are not in $L_{res}SN^eP_*(rule_*, cons_*, prod_*)$ (in all cases, $card(V) = k \geq 2$):*

$$L_1 = \{x\,mi(x) \mid x \in V^+\},$$
$$L_2 = \{xx \mid x \in V^+\},$$
$$L_3 = \{x\,c^{val_k(x)} \mid x \in V^+\}, c \notin V.$$

Note that language L_1 above is a non-regular minimal linear one, L_2 is context-sensitive non-context-free, and L_3 is non-semilinear. In all cases, we can also add a fixed tail of any length (e.g., considering $L_1' = \{x\,mi(x)z \mid x \in V^+\}$, where $z \in V^+$ is a given string), and the conclusion is the same—hence a result like that in Theorem 6.3 cannot be extended to minimal linear languages.

7 Languages in the non-restricted case

As expected, the possibility of having intermediate steps when no output is produced is helpful, because this provides intervals for internal computations. In this way, we can get rid of the operations used in Chen et al. (2006) and in the previous sections when dealing with regular and with recursively enumerable languages.

7.1 Relationships with REG

Lemma 7.1 $L_\lambda SN^eP_2(rule_*, cons_*, prod_*) \subseteq REG$.

Proof In a system with two neurons, the number of spikes from the system can remain the same after a step, but it cannot increase: the neurons can consume the same number of spikes as they produce, and they can send to each other the produced spikes. Therefore, the number of spikes in the system is bounded by the number of spikes present at the beginning. This means that the system can pass through a finite number of configurations and these configurations can control the evolution of the system like states in a finite automaton. Consequently, the generated language is regular [see similar reasonings, with more technical details, in Ionescu et al. (2006), Chen et al. (2006)]. □

Lemma 7.2 *$REG \subseteq L_\lambda SN^eP_3(rule_*, cons_*, prod_*)$.*

Proof For the SN P system Π constructed in the proof of Theorem 6.3 (Fig. 7) we have $L_\lambda(\Pi) = L(G)$. □

This last inclusion is proper:

Proposition 7.1 *$L_\lambda\ SN^eP_3(rule_4,cons_4,prod_2)-REG \neq \emptyset$.*

Proof The SN P system Π from Fig. 8 generates the language $L_\lambda(\Pi) = \{b_2^n b_1^{n+1} \mid n \geq 1\}$. Indeed, for a number $n \geq 0$ of steps, neuron σ_2 consumes two spikes by using the rule $(a^2)^+/a^2 \rightarrow a^2$ and receives four from the other two neurons. After changing the parity of the number of spikes (by using the rule $(a^2)^+/a^3 \rightarrow a^2$), neuron σ_2 will continue by consuming four spikes (using the rule $a(a^2)^+/a^4 \rightarrow a$) and receiving only two. When only three spikes remain, the computation stops (the two further spikes received by σ_2 from σ_1 and σ_3 cannot fire again neuron σ_2). □

Corollary 7.1 *$L_\lambda SN^eP_1(rule_*, cons_*, prod_*) \subset L_\lambda SN^eP_2(rule_*, cons_*, prod_*) \subset L_\lambda SN^eP_3(rule_*, cons_*, prod_*)$, strict inclusions.*

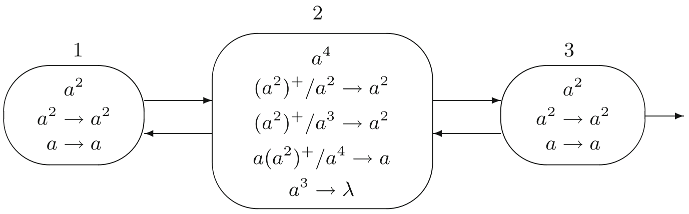
7.2 Going beyond CF

Actually, much more complex languages can be generated by extended SN P systems with three neurons.

Theorem 7.1 *The family $L_\lambda\ SN^eP_3(rule_3,cons_6,prod_4)$ contains non-semilinear languages.*

Proof The system Π from Fig. 9 generates the language.

$$L_\lambda(\Pi) = \{b_4^2 b_2 b_4^{2^2} b_2 \ldots b_4^{2^n} b_2 \mid n \geq 1.\}$$



1
a^2
$a^2 \rightarrow a^2$
$a \rightarrow a$
2
a^4
$(a^2)^+/a^2 \rightarrow a^2$
$(a^2)^+/a^3 \rightarrow a^2$
$a(a^2)^+/a^4 \rightarrow a$
$a^3 \rightarrow \lambda$
3
a^2
$a^2 \rightarrow a^2$
$a \rightarrow a$

Fig. 8 An SN P system generating a non-regular language

We start with $2 + 4{\cdot}2^0$ spikes in neuron σ_1. When moved from neuron σ_1 to neuron σ_3, the number of spikes is doubled, because they pass both directly from σ_1 to σ_3, and through σ_2. When all spikes are moved to σ_3, the rule $a^2 \rightarrow a$ of σ_1 should be used. With a number of spikes of the form $4m + 1$, neuron σ_3 cannot fire, but in the next step one further spike comes from σ_2, hence the first rule of σ_3 can now be applied. Using this rule, all spikes of σ_3 are moved back to σ_1—in the last step we use the rule $a^2 \rightarrow a^2$, which makes again the first rule of σ_1 applicable.

This process can be repeated any number of times. In each moment, after moving all but the last six spikes from neuron σ_1 to σ_3, we can also use the rule $a^6 \rightarrow a^3$ of σ_1, and this ends the computation: there is no spike in σ_1, neuron σ_2 cannot work when having three spikes inside, and the same with σ_3 when having $4m + 3$ spikes.

Now, one sees that σ_3 is also the output neuron and that the number of times of using the first rule of σ_3 is doubled after each move of the contents of σ_3 to σ_1. □

In this proof we made use of the fact that no spike of the output neuron means no symbol introduced in the generated string. If we work in the restricted case, then symbols b_0 are shuffled in the string, hence the non-semilinearity of the generated language is preserved, that is, the result also holds for the restricted case.

7.3 A characterization of RE

If we do not bound the number of neurons, then a characterization of recursively enumerable languages is obtained.

Let us write s in front of a language family notation in order to denote the subfamily of languages over an alphabet with at most s symbols (e.g., $2RE$ denotes the family of recursively enumerable languages over alphabets with one or two symbols).

Lemma 7.3 *$sRE \subseteq sL_\lambda SN^eP_*(rule_{s'}, cons_s, prod_s)$, where $s' = \max(s,6)$ and $s \geq 1$.*

Proof We follow here the same idea as in the proof of Theorem 5.9 from Chen et al. (2006), adapted to the case of extended rules.

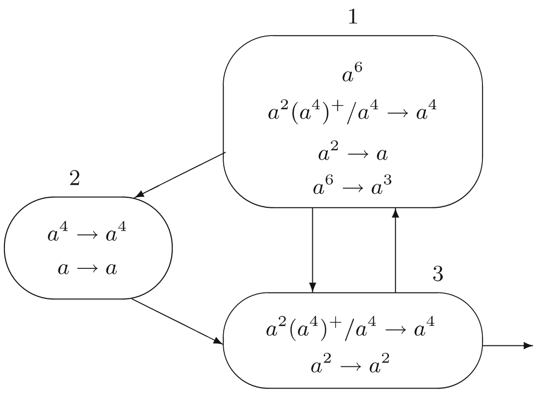


Fig. 9 An SN P system generating a non-semilinear language

Take an arbitrary language $L \subseteq V^*$, $L \in RE$, $card(V) = s$. Obviously, $L \in RE$ if and only if $val_s(L) \in NRE$. In turn, a set of numbers is recursively enumerable if and only if it can be accepted by a deterministic register machine. Let M_1 be such a register machine, i.e., $N(M_1) = val_s(L)$.

We construct an SN P system Π performing the following operations (σ_{c_0} and σ_{c_1} are two distinguished neurons of Π, which are empty in the initial configuration):

1. Output i spikes, for some $1 \leq i \leq s$, and at the same time introduce the number i in neuron σ_{c_0}; in the construction below, a number n is represented in a neuron by storing there $3n$ spikes, hence the previous task means introducing $3i$ spikes in neuron σ_{c_0}.
2. Multiply the number stored in neuron σ_{c_1} (initially, we have here number 0) by $s + 1$, then add the number from neuron σ_{c_0}; specifically, if neuron σ_{c_0} holds $3i$ spikes and neuron σ_{c_1} holds $3n$ spikes, $n \geq 0$, then we end this step with $3(n(s + 1) + i)$ spikes in neuron σ_{c_1} and no spike in neuron σ_{c_0}. In the meantime, the system outputs no spike.
3. Repeat from step 2, or, non-deterministically, stop the increase of spikes from neuron σ_{c_1} and pass to the next step.
4. After the last increase of the number of spikes from neuron σ_{c_1} we have here $val_s(x)$ for a string $x \in V^+$. Start now to simulate the work of the register machine M_1 in recognizing the number $val_s(x)$. The computation halts only if this number is accepted by M_1, hence the string x produced by the system is introduced in the generated language only if $val_s(x) \in N(M_1)$.

In constructing the system Π we use the fact that a register machine can be simulated by an SN P system. Then, the multiplication by $s + 1$ of the contents of neuron σ_{c_1} followed by adding a number between 1 and s is done by a computing register machine (with the numbers stored in neurons $\sigma_{c_0}, \sigma_{c_1}$ introduced in two specified registers); we denote by M_0 this machine. Thus, in our construction, also for this operation we can rely on the general way of simulating a register machine by an SN P system. All other modules of the construction (introducing a number of spikes in neuron σ_{c_0}, sending out spikes, choosing non-deterministically to end the string to generate and switching to the checking phase, etc.) are explicitly presented below.

The overall appearance of Π is given in Fig. 10, where M_0 indicates the subsystem corresponding to the simulation of the register machine $M_0 = (m_0, H_0, l_{0,0}, l_{h,0}, I_0)$ and M_1 indicates the subsystem which simulates the register machine $M_1 = (m_1, H_1, l_{0,1}, l_{h,1}, I_1)$. Of course, we assume $H_0 \cap H_1 = \emptyset$.

We start with spikes only in neuron σ_{d_9}. We spike in the first step, non-deterministically choosing the number i of spikes to produce, hence the first letter b_i of the generated string. Simultaneously, i spikes are sent out by the output neuron, $3i$ spikes are sent to neuron σ_{c_0}, and three spikes are sent to neuron $\sigma_{l_{0,0}}$, thus triggering the start of a computation in M_0. The subsystem corresponding to the register machine M_0 starts to work, multiplying the value of σ_{c_1} with $s + 1$ and adding i. When this process halts, neuron $\sigma_{l_{h,0}}$ is activated, and in this way two spikes are sent to neuron σ_{d_6}.

This is the neuron which non-deterministically chooses whether the string should be continued or we pass to the second phase of the computation, checking whether the produced string is accepted. In the first case, neuron σ_{d_6} uses the rule $a^2 \to a$, which makes neurons $\sigma_{e_1}, \ldots, \sigma_{e_m}$ spike; these neurons send m spikes to neuron σ_{d_9}, like in the beginning of the computation. In the latter case, one uses the rule $a^2 \to a^2$, which activates the neuron $\sigma_{l_{0,1}}$ by sending three spikes to it, thus starting the simulation of the register machine M_1. The computation stops if and only if $val_s(x)$ is accepted by M_1.

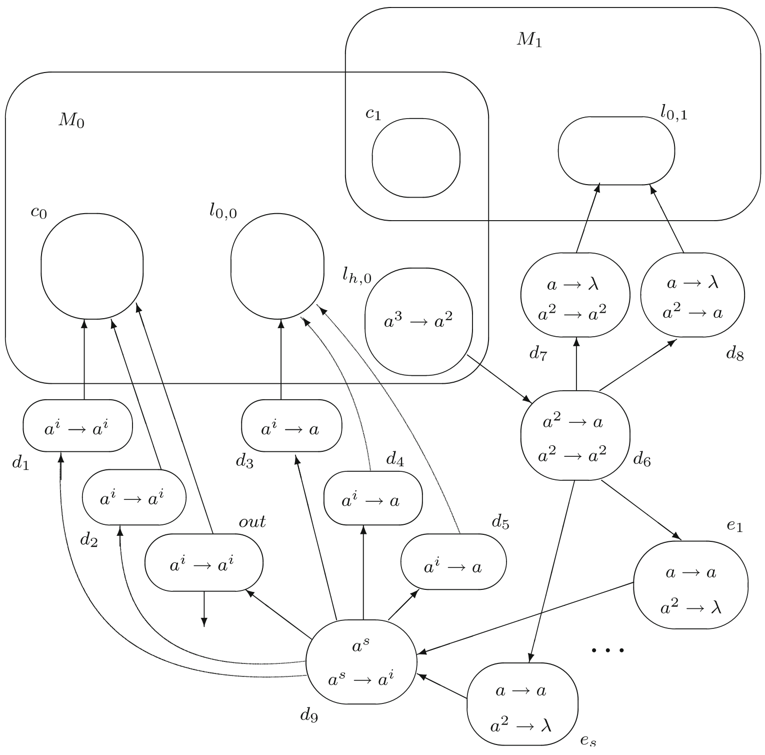


Fig. 10 The structure of the SN P system from the proof of Lemma 7.3

In order to complete the proof we need to show how the two register machines are simulated, using the common neuron σ_{c_1} but without mixing the computations. To this aim, we consider the modules ADD and SUB from Figs. 11–13. Like in Sect. ''Extended SN P systems as number generators'', neurons are associated with each label of the machine (they fire if they have three spikes inside) and with each register (with $3t$ spikes representing the number t from the register); there also are additional neurons with labels c_{il}—it is important to note that all these additional neurons have distinct labels.

The simulation of an ADD instruction is easy, we just add three spikes to the respective neuron; no rule is needed in the neuron—Fig. 11. The SUB instructions of machines M_0,M_1 are simulated by modules as in Figs. 12 and 13, respectively. Note that the rules for M_0 fire for a content of the neuron σ_r described by the regular expression $(a^3)^+a$ and the rules for M_1 fire for a content of the neuron σ_r described by the regular expression $(a^3)^+a^2$. To this aim we use the rule $a^3 \to a^2$ in σ_{l_i} instead of $a^3 \to a$, while in σ_r we use the rule $(a^3)^+a^2/a^5 \to a^4$ instead of $(a^3)^+a/a^4 \to a^3$. This ensures the fact that the rules of M_0 are not used instead of those of M_1 or vice versa. In neurons associated with different labels of M_0,M_1 we have to use different rules, depending on the type of instruction simulated, that is why in Figs. 11–13 we have written again some rules in the form $a^3 \to a^{\delta(l)}$, as in Figs. 1 and 2. Specifically, $\delta(l) = 3$ if l labels an ADD instruction, $\delta(l) = 1$ or $\delta(l) = 2$ if l labels a SUB instruction of M_0 or of M_1, respectively, and, as one sees in Fig. 10, we also take $\delta(l_{h,0}) = 2$.

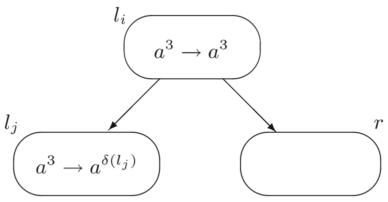


Fig. 11 Module ADD (simulating $l_i : (\mathtt{ADD}(r), l_j)$)

With these explanations, the reader an check that the system Π works as requested, hence $L_\lambda(\Pi) = L$ (in Figs. 12, 13 we have neurons with six rules, that is why $s' = \max(s, 6)$). $\square$

Theorem 7.2 $RE = L_\lambda SN^eP_*(rule_*, cons_*, prod_*)$.

In the proof of Lemma 7.3, if the moments when the output neuron emits no spike are associated with the symbol b_0, then the generated strings will be shuffled with occurrences of b_0. Therefore, L is a projection of the generated language.

Corollary 7.2 *Every language $L \in RE, L \subseteq V*$, can be written in the form $L = h(L')$ for some $L' \in L_{res}SN^eP_*(rule_*, cons_*, prod_*)$, where h is a projection on $V \cup \{b_0\}$ which removes the symbol b_0.*

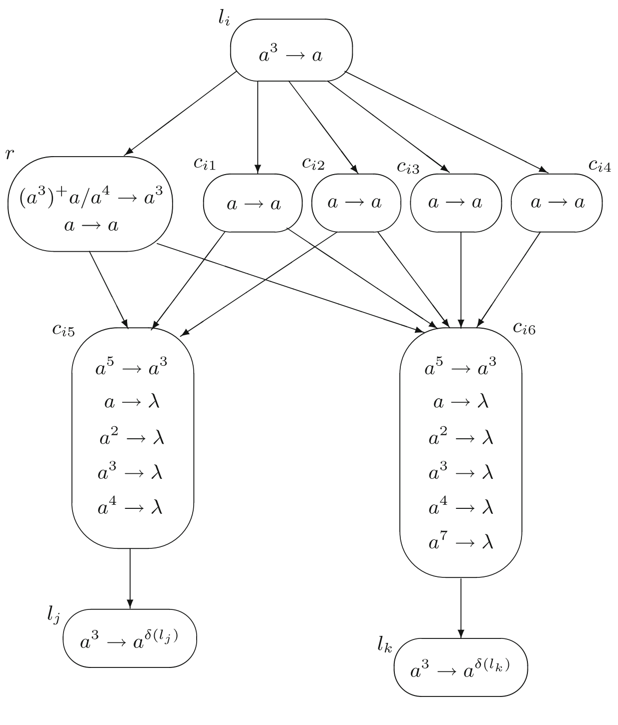


Fig. 12 Module SUB (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$) for machine M_0

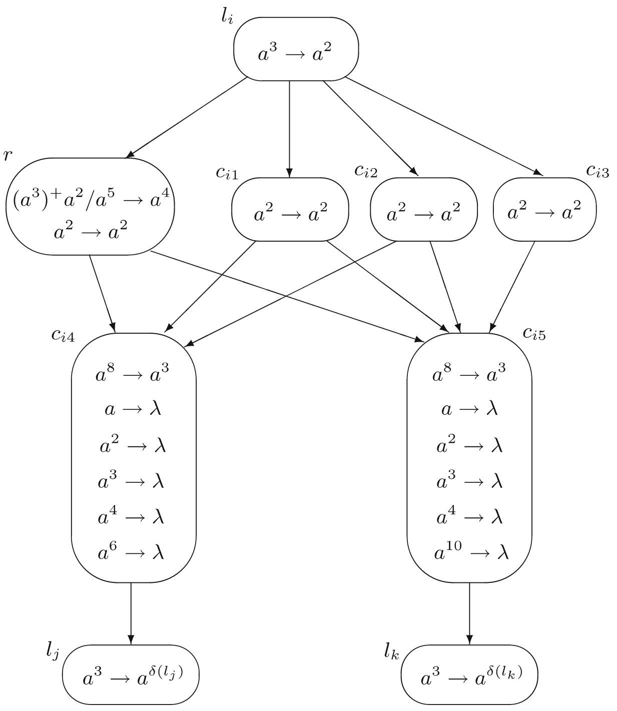


Fig. 13 Module SUB (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$) for machine M_1

8 Final remarks

We have investigated here the power of SN P systems with extended rules (rules allowing to introduce several spikes at the same time) both as number generators and as language generators. In the first case we have provided a simpler proof of a known universality result, in the latter case we have proved characterizations of finite and recursively enumerable languages, and representations of regular languages. Results concerning the size of universal SN P systems with standard rules or with extended rules are also recalled.

Finding characterizations (or at least representations) of other families of languages from Chomsky hierarchy and Lindenmayer area remains as a research topic. It is also of interest to investigate the possible hierarchy on the number of neurons, extending the result from Corollary 7.1, as well as to decrease the number of neurons from universal SN P systems.

Acknowledgements The authors gratefully acknowledge the following (partial) support of their research. H. Chen: Grants numbers 60573013 and 60421001 of the National Natural Science Foundation of China. M. Ionescu: Programa Nacional para la Formación del Profesorado Universitario from the Spanish Ministry of Education. A. Păun: LA BoR RSC grant LEQSF (2004-07)-RD-A-23, and NSF Grants IMR-0414903 and CCF-0523572. Gh. Păun and M.J. Pérez-Jiménez: Project TIN2005-09345-C03-01 of Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and Project of Excellence TIC 581 of Junta de Andalucia.

References

Chen H, Freund R, Ionescu M, Păun Gh, Pérez-Jiménez MJ (2006) On string languages generated by spiking neural P systems. In Proceedings of the fourth brainstorming week on membrane computing, vol. I, Sevilla, pp 169–193 (available at http://psystems.disco.unimib.it)

Ibarra OH, Păun A, Păun Gh, Rodrí guez-Patón A, Sosik P, Woodworth S (2006) Normal forms for spiking neural P systems. In Fourth brainstorming week on membrane computing, Febr. 2006, vol. II, Fenix Editora, Sevilla, pp 105–136

Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fundam Informat 71(2–3):279–308

Korec I (1996) Small universal register machines. Theor Comput Sci 168:267–301

Maass W (2002) Computing with spikes. Special Issue Found Inform Process TELEMATIK 8(1):32–36

Maass W, Bishop C (eds) (1999) Pulsed neural networks. MIT Press, Cambridge

Minsky M (1967) Computation – finite and infinite machines. Prentice Hall, Englewood Cliffs, NJ

Păun Gh (2002) Membrane computing – an introduction. Springer-Verlag, Berlin

Păun A, Păun Gh (to appear) Small universal spiking neural P systems. BioSystems

Păun Gh, Pérez-Jiménez MJ, Rozenberg G (2006) Spike trains in spiking neural P systems. Int J Found Computer Sci 17(4):975–1002

Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages, 3 volumes. Springer-Verlag, Berlin

Salomaa A (1973) Formal languages. Academic Press, New York

The P Systems Web Page: http://psystems.disco.unimib.it