On String Languages Generated by Spiking Neural P Systems

Haiming Chen[1], Rudolf Freund[2], Mihai Ionescu[3], Gheorghe Păun[4,5], Mario J. Pérez-Jiménez[5]

[1] Computer Science Laboratory, Institute of Software
Chinese Academy of Sciences
100080 Beijing, China
`chm@ios.ac.cn`

[2] Faculty of Informatics, Technische Universität Wien
Favoritenstraße 9, A-1040 Wien, Austria
`rudi@emcc.at`

[3] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
`armandmihai.ionescu@urv.net`

[4] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania

[5] Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
`gpaun@us.es, marper@us.es`

Summary. We continue the study of spiking neural P systems by considering these computing devices as binary string generators: the set of spike trains of halting computations of a given system constitutes the language generated by that system. Although the work of spiking neural P systems is rather restricted (and this is illustrated by the fact that very simple languages cannot be generated in this framework), regular languages are inverse-morphic images of languages of finite spiking neural P systems, and recursively enumerable languages are projections of inverse-morphic images of languages generated by spiking neural P systems.

1 Introduction

Spiking neural P systems (in short, SN P systems) were recently introduced in [7], and then investigated in [12], [13], and [6], thus incorporating ideas from neural computing by spiking (see, e.g., [4], [8], [9]) in membrane computing [11].

In Section 3 SN P systems will be introduced formally; now, we only mention that such a system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. There can be a delay between firing and producing a spike. Moreover, we also use forgetting rules, which remove spikes from neurons. The system works in a synchronized manner, i.e., in each time unit, each neuron which can use a rule should do it, but the work of the system is sequential in each neuron: only (at most) one rule is used, taking into account all spikes present in the neuron (precise definitions are given in Section 3). One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the spike train of the system – it might be infinite if the computation does not stop.

Several ways to associate sets of numbers with spike trains were investigated in [7] and [12]: considering the distance between the first 2 spikes or between the first k spikes of a spike train, or the distances between all consecutive spikes, taking into account all intervals or only intervals that alternate, etc. Computational completeness results were obtained in all cases: these devices compute exactly the sets of natural numbers computed by Turing machines.

Another attractive possibility is to consider the spike trains themselves as the result of a computation; moreover, we may even consider input neurons and then an SN P system can work as a transducer; details can be found in [13], where this case was investigated.

Several normal form theorems were proved in [6] – removing the delay between firing and spiking, removing the forgetting rules, etc. – but here we work in the general case, without imposing such restrictions.

In the present paper we address a more standard (i.e., language-theoretic) question: which is the power of SN P systems as language generators. The way to associate a language $L(\Pi)$ with an SN P system Π is obvious: we consider as successful only the halting computations of Π, and into $L(\Pi)$ exactly the binary strings describing the spike trains of the halting computations are taken.

Because the strings are generated symbol by symbol from left to right, and, moreover, during each time unit of a computation we wait outside the system for a bit, each string produced in this way has the length equal to the duration of a computation; hence, there are strong restrictions on the computing power of SN P systems (as string generators). This observation will be illustrated by several results below – but the generative capacity of SN P systems is not at all small, "hard" languages can also be generated, and recursively enumerable languages can be characterized as projections of inverse-morphic images of languages generated by SN P systems.

Several natural questions remain to be investigated and some of them are formulated in the paper (e.g., changing the definition of an SN P system or of the generated language, looking for characterizations of recursively enumerable languages starting from fixed spiking neural P systems, etc.).

2 Formal Language Theory Prerequisites

We assume the reader to be familiar with basic language and automata theory, e.g., from [16] and [17], hence, we here introduce only some notations and notions used later 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^+. The length of a string $x \in V^*$ is denoted by $|x|$, and $|x|_a$ is the number of occurrences of the symbol $a \in V$ in the string x.

Regular expressions over an alphabet V can iteratively be obtained by defining (1) $\emptyset$ and a for every $a \in V$ are regular expressions over V, and (2) if E_1 and E_2 are regular expressions, then $(E_1 \cup E_2)$ – union –, $(E_1 \cdot E_2)$ – concatenation, usually only written as (E_1E_2) –, and $(E_1)^*$ – Kleene star – are regular expressions, too. The regular set (language over V) represented by the regular expression E is denoted by $L(E)$ and obtained by replacing a with the corresponding singleton set $\{a\}$ for every $a \in V$. Moreover, for any regular expressions E over V, $(E)^+$ represents the regular set $L(E^+)$ ($= L(EE^*)$).

For a morphism $h : V_1^* \longrightarrow V_2^*$ and a string $y \in V_2^*$ we define $h^{-1}(y) = \{x \in V_1^* \mid h(x) = y\}$; this mapping from V_2^* to the power set of V_1^* is extended in the natural way to languages over V_2 and is called the inverse morphism associated with h. A morphism $h : V_1^* \longrightarrow V_1^*$ is called projection if $h(a) \in \{a, \lambda\}$ for each $a \in V_1$.

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

A Chomsky grammar will be given in the form $G = (N, T, S, P)$, with N being the nonterminal alphabet, T the terminal alphabet, $S \in N$ the axiom, and P the set of rules. The families of finite, regular, context-free, context-sensitive, and recursively enumerable languages are denoted by FIN, REG, CF, CS, and RE, respectively. Below we will also invoke the family MAT, i.e., the family of languages generated by matrix grammars without appearance checking (see [1] for details) and the family REC, i.e., the family of recursive languages (languages whose membership problem is decidable).

In the main results of this paper, i.e., the characterization of recursively enumerable languages, we use the notion of a deterministic register machine. Such a device 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 labeled in a one-to-one manner by the labels from H. The instructions are of the following forms:

- $l_1 : (\texttt{ADD}(r), l_2)$ (add 1 to register r and then go to the instructions with label l_2),

- $l_1 : (\texttt{SUB}(r), l_2, l_3)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M accepts a number n in the following way: we start with number n in a specified register r_0 and all other registers being empty (i.e., storing the number zero), we first apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of the registers); if we reach the halt instruction, then the number n is said to be accepted by M. The set of all numbers accepted by M is denoted by $N(M)$. It is known (see, e.g., [10]) that register machines (even with only three registers, but this detail is not relevant here) accept all sets of numbers which are Turing computable.

A register machine can also compute any Turing computable function: we introduce the arguments in specified registers $r_1, \ldots, r_k$, and start with the instruction with label l_0; 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. In the proof of Theorem 9 we will use also this way of working with register machines.

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

3 Spiking Neural P Systems

We pass directly to considering the device we investigate in this paper; we refer to [7], [12] for motivation and more detailed definitions.

A *spiking neural P system* (abbreviated as 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 following two forms:
      (1) $E/a^c \rightarrow a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
      (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;

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.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a; d$ can be applied. The application of this rule means consuming (removing) c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces a spike after d time units (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t{+}1, t{+}2, \ldots, t{+}d{-}1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost). In the step $t + d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t + d + 1$), but it does not use any rule at this step (the neuron is busy with sending out the spike it has produced d steps before and stored up to now inside).

The rules of type (2) are *forgetting* rules and they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \to a; d$ of type (1) has $E = a^c$, then we will write it in the simplified form $a^c \to a; d$. If for every firing rule $E/a^c \to a; d$ the regular set $L(E)$ is finite, the SN P system is called *finite.*

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used. Since two firing rules, $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and in that case, only one of them is chosen non-deterministically. By definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron, with all neurons being open. During the computation, a configuration is described by both the number of spikes present in each neuron and by the state of the neuron, more precisely, by the number of steps to count down until it becomes open (this number is zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron $i = 1, 2, \ldots, m$ contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps; with this notation, the initial configuration is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$.

Using the rules as described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation.* A computation halts if it reaches a configuration where all neurons are open and 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.

A more precise definition of the notion of a transition can be given, but instead of cumbersome details we will illustrate this notion with an example in the next section.

In the spirit of spiking neurons, in [7] and [12] the distance between two consecutive spikes which exit the system is considered to be the result of a computation, with the many variants suggested in the Introduction. Here we go into a different direction and consider the spike train itself as the result of a computation, thus associating a language with an SN P system; in order to have a language of finite strings, we take into consideration only halting computations.

More formally, let $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$ be an SN P system and let γ be a halting computation in Π, $\gamma = C_0 \Longrightarrow C_1 \Longrightarrow \ldots \Longrightarrow C_k$ (C_0 is the initial configuration, and $C_{i-1} \Longrightarrow C_i$ is the ith step of γ). Let us denote by $bin(\gamma)$ the string $b_1 b_2 \ldots b_k$, where $b_i \in \{0, 1\}$, and $b_i = 1$ if and only if the (output neuron of the) system Π sends a spike into the environment in step i of γ. We denote by B the binary alphabet $\{0, 1\}$, and by $COM(\Pi)$ the set of all halting computations of Π. Moreover, we define the language generated by Π by

$$L(\Pi) = \{bin(\gamma) \mid \gamma \in COM(\Pi)\}.$$

In the next sections we will illustrate these definitions with a series of examples.

The complexity of an SN P system can be described by means of various parameters (number of neurons, of rules, of consumed or forgotten spikes, maximum delay, or outdegree/indegree of the synapse graph, etc.), but here we only consider the basic parameters also used in [7], and we denote by $LSNP_m(rule_k, cons_p, forg_q)$ the family of languages $L(\Pi)$, generated by systems Π with at most m neurons, each neuron having at most k rules, each of the spiking rules consuming at most p spikes, and each forgetting rule removing at most q spikes. As usual, a parameter m, k, p, q is replaced with $*$ if it is not bounded. If the underlying SN P systems are finite, we denote the corresponding families of languages by $LFSNP_m(rule_k, cons_p, forg_q)$.

4 An Illustrative Example

We consider a system with a simple architecture, but with an intricate behavior. This also gives us the opportunity to illustrate the way to graphically represent an SN P system: as a directed graph, with the neurons as nodes and the synapses indicated by arrows; an arrow also exits from the output neuron, pointing to the environment; in each neuron we specify the rules and the spikes present in the initial configuration.

Figure 1 represents the initial configuration of the system Π_1. We have three neurons, labeled with 1, 2, 3, with neuron 3 being the output one. Each neuron contains two rules, with neurons 1 and 2 having the same rules (firing rules which can be chosen in a non-deterministic way, the difference between them being in

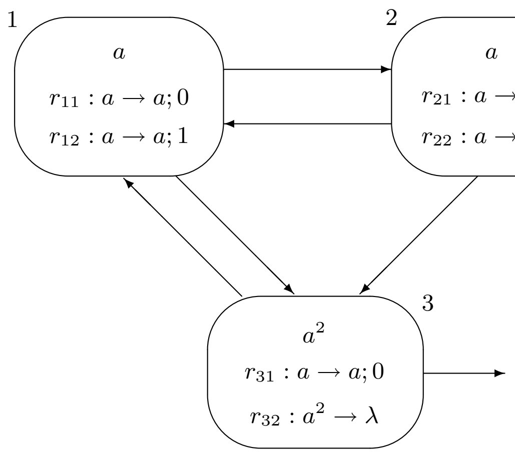

Fig. 1. The initial configuration of system Π_1

the delay from firing to spiking), and neuron 3 having one firing and one forgetting rule. In the figure, the rules are labeled, and these labels are useful below.

The evolution of the system Π_1 can be analyzed on a transition diagram as that from Figure 2: because the system is finite, the number of configurations reachable from the initial configuration is finite, too, hence, we can place them in the nodes of a graph, and between two nodes/configurations we draw an arrow if and only if a direct transition is possible between them. In Figure 2 we have also indicated the rules used in each neuron, with the following conventions: for each r_{ij} we have written only the subscript ij, with *31* being written in italics, in order to indicate that a spike is sent out of the system at that step; when a neuron $i = 1, 2, 3$ uses no rule, we have written $i0$, and when it spikes (after being closed for one step), we write is.

The functioning of the system can easily be followed on this diagram, so that we only briefly describe it. We start with spikes in all neurons. Neurons 1 and 2 behave non-deterministically, choosing one of the two rules. As long as they use the rules $a \to a; 0$, the computation cycles in the initial configuration: neurons 1 and 2 exchange spikes, while neuron 3 forgets its two spikes. If both neurons use the second rule the rule $a \to a; 1$, then both neurons fire, but do not spike immediately, and we reach the configuration $\langle 0/1, 0/1, 0/0\rangle$. In the next step, both neurons spike, and we return to the initial configuration. When only one of neurons 1, 2 uses the rule $a \to a; 0$ (and therefore spikes immediately) and the other one uses the rule $a \to a; 1$ (and fires, but does not spike, hence, is closed and cannot receive spikes), then only one spike arrives in neuron 3, and in the next step neuron 3 as well as the neuron having used the rule $a \to a; 1$ now send out a spike. If neuron 1 has used the rule $a \to a; 0$ and neuron 2 has used the rule $a \to a; 1$, we reach the configuration $\langle 2/0, 0/0, 1/0\rangle$; neurons 1 and 2 now cannot apply a rule

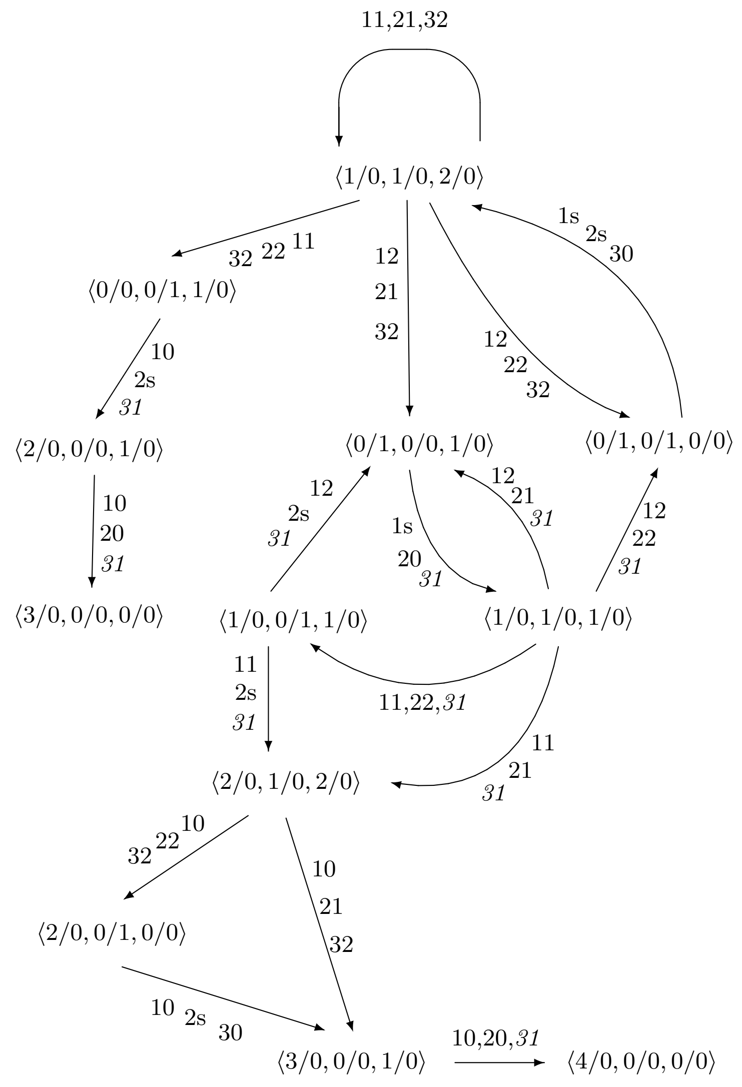

Fig. 2. The transition diagram of system Π_1

anymore, thus after one more spiking of neuron 3 we reach the halting configuration $\langle 3/0, 0/0, 0/0\rangle$. If, conversely, neuron 1 uses the rule $a \rightarrow a; 1$ and neuron 2 uses the rule $a \rightarrow a; 0$, then we first get the configuration $\langle 0/1, 0/0, 1/0\rangle$; in the next step, neuron 1 sends its spike to neurons 2 and 3, while neuron 3 also spikes, and "reloads" neuron 1. Then the computation can either run along the cycles depicted in the central part of the diagram from Figure 2, or it can reach again the initial configuration, or else it can reach the halting configuration $\langle 4/0, 0/0, 0/0\rangle$ from the configuration $\langle 2/0, 1/0, 2/0\rangle$ in two or three steps (as indicated in the bottom part of the diagram).

The transition diagram of a finite SN P system can be interpreted as the representation of a non-deterministic finite automaton, with C_0 being the initial state, the halting configurations being final states, and each arrow being marked with 0 if in that transition the output neuron does not send a spike out, and with 1 if in the respective transition the output neuron spikes; in this way, we can identify the language generated by the system. In the case of the finite SN P system Π_1, the generated language is the following one:

$$L(\Pi_1) = L((0^*0(11 \cup 111)^*110)^*0^*(011 \cup 0(11 \cup 111)^+(0 \cup 0^2)1)).$$

We here do not present further examples, because many of the results in the next section are based on effective constructions of SN P systems.

5 The Generative Power of SN P Systems

As already suggested in the Introduction, the power of SN P systems used as language generators is rather “ex-centric”: “easy” languages cannot be generated, but on the other hand some “hard” languages can be generated.

We start by pointing out an example of the first type.

Theorem 1. *No language of the form $L_{k,j} = \{0^k, 10^j\}$, for $k \geq 1$, $j \geq 0$, can be generated by an SN P system.*

Proof. In order to generate a string 10^j, in the initial configuration the output neuron must contain at least one spike. In such a case, no string of the form 0^k can be generated: if $k = 1$, then we need a forgetting rule $a^r \rightarrow \lambda$ which can be applied at the same time with a spiking rule, and this is not possible, by definition (no forgetting rule can be interchanged with a spiking rule); if $k \geq 2$, then in the first step the output neuron should not use a rule, but this is not allowed by the way of defining the computations in a synchronized way.

The simplest language of the form above is $L_{1,0} = \{0, 1\}$, which does not belong to $LSNP_*(rule_*, cons_*, forg_*)$. The same argument works for any language of the form $\{0^k, 1x\}$, where x is an arbitrary string over the binary alphabet.

This does not at all mean that languages consisting of two words similar to those considered above (one word starting with 1 and one with 0) cannot be generated by our systems. An example is the language $\{100, 01\}$, which is generated by the system Π_2 from Figure 3; for the reader’s convenience, the transition diagram of system Π_2 is given in Figure 4.

We remain in the same area, of finite languages, mentioning the following three results.

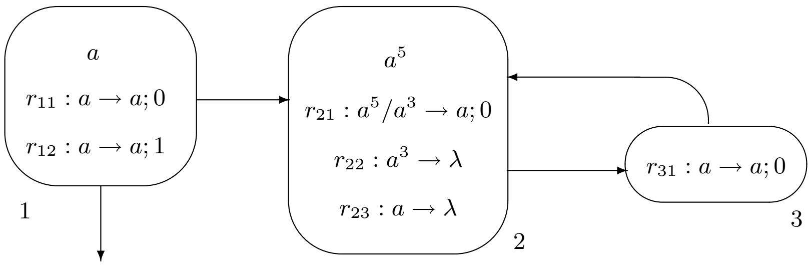

Fig. 3. An SN P system (Π_2) generating the language $\{100, 01\}$

$$\begin{tikzcd}
 & \langle 1/0, 5/0, 0/0\rangle \arrow[dl, "\begin{array}{c}\mathit{11}\\21\\30\end{array}"'] \arrow[dr, "\begin{array}{c}12\\21\\30\end{array}"] & \\
\langle 0/0, 3/0, 1/0\rangle \arrow[d, "\begin{array}{c}10\\22\\31\end{array}"'] & & \langle 0/1, 2/0, 1/0\rangle \arrow[d, "\begin{array}{c}\mathit{1s}\\20\\31\end{array}"] \\
\langle 0/0, 1/0, 0/0\rangle \arrow[d, "\begin{array}{c}10\\23\\30\end{array}"'] & & \langle 0/0, 4/0, 0/0\rangle \\
\langle 0/0, 0/0, 0/0\rangle & &
\end{tikzcd}$$

Fig. 4. The transition diagram of the system Π_2

Theorem 2. *If $L = \{x\}$, $x \in B^+$, $|x|_1 = r \geq 0$, then $L \in LFSNP_2(rule_{r+1}, cons_1, forg_0)$.*

Proof. Let us consider a string $x = 0^{n_1}10^{n_2}\dots 0^{n_r}10^{n_{r+1}}$, for $n_j \geq 0$, $1 \leq j \leq r+1$ (if $x = 0^{n_1}$, then $r = 0$). The SN P system from Figure 5 generates the string x. The output neuron initially contains r spikes. At step 1, the rule $a^r/a \to a; n_1$ can be applied. One spike is removed and at step $n_1 + 1$ one spike is sent out. We continue in this way until using the rule $a/a \to a; n_r$, i.e., until exhausting the spikes; the last spike is sent out at step $\sum_{i=1}^{r} n_i + r$. The second neuron (not having a synapse with the output one) is meant to make the computation last exactly $|x|$ steps. Note that $|x| - (\sum_{j=1}^{r} n_j + r) = n_{r+1}$, therefore the system halts after generating n_{r+1} more occurrences of 0.

In the case $r = 0$, the system contains no rule in neuron 1, but there is one rule in neuron 2, that is why we have $rule_{r+1}$ in the theorem statement.

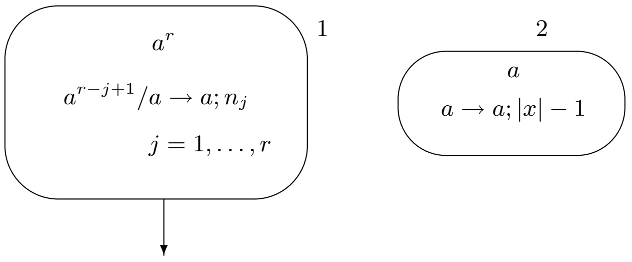

Fig. 5. An SN P system generating a singleton language

Actually, modulo a supplementary final occurrence of 1, any finite language can be generated.

Theorem 3. *If $L \in FIN$, $L \subseteq B^+$, then $L\{1\} \in LFSNP_1(rule_*, cons_*, forg_0)$.*

Proof. Let us assume that $L = \{x_1, x_2, \ldots, x_m\}$, with $|x_j 1| = n_j \geq 2, 1 \leq j \leq m$; denote $\alpha_j = \sum_{i=1}^{j} n_i$, for all $1 \leq j \leq m$. We write $x_j 1 = 0^{s_{j,1}} 1 0^{s_{j,2}} 1 \ldots 1 0^{s_{j,r_j}} 1$, for $r_j \geq 1$, $s_{j,l} \geq 0$, $1 \leq l \leq r_j$.

An SN P system which generates the language $L\{1\}$ is the following:

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \emptyset, 1),\\
\sigma_1 &= (\alpha_m + 1, R_1),\\
R_1 &= \{a^{\alpha_m+1}/a^{\alpha_m+1-\alpha_j} \to a; s_{j,1} \mid 1 \leq j \leq m\}\\
&\quad \cup \{a^{\alpha_j - t+2}/a \to a; s_{j,t} \mid 2 \leq t < r_j - 1, 1 \leq j \leq m\}\\
&\quad \cup \{a^{\alpha_j - r_j + 2} \to a; s_{j,r_j} \mid 1 \leq j \leq m\}.
\end{aligned}$$

Initially, only a rule $a^{\alpha_m+1}/a^{\alpha_m+1-\alpha_j} \to a; s_{j,1}$ can be used, and in this way we non-deterministically chose the string x_j to generate. After $s_{j,1}$ steps, for some $1 \leq j \leq m$, we output a spike, hence, in this way the prefix $0^{s_{j,1}}1$ of the string x_j is generated. Because α_j spikes remain in the neuron, we have to continue with rules $a^{\alpha_j-t+2}/a \to a; s_{j,t}$, for $t = 2$, and then for the respective $t = 3, 4, \ldots, r_j - 1$; in this way we introduce the substrings $0^{s_{j,t}}1$ of x_j, for all $t = 2, 3, \ldots, r_j - 1$. The last substring, $0^{s_{j,r_j}}1$, is introduced by the rule $a^{\alpha_j - r_j+2} \to a; s_{j,r_j}$, which concludes the computation.

We observe that the rules which are used in the generation of a string $x_j 1$ cannot be used in the generation of a string $x_k 1$ with $k \neq j$.

Corollary 1. *Every language $L \in FIN$, $L \subseteq B^+$, can be written in the form $L = \partial_1^r(L')$ for some $L' \in LFSNP_1(rule_*, cons_*, forg_0)$.*

A sort of "mirror result" can be obtained, based on the idea used in the proof of Theorem 2.

Theorem 4. *If $L \in FIN$, $L \subseteq B^+, L = \{x_1, x_2, \dots, x_n\}$, then $\{0^{i+3}x_i \mid 1 \leq i \leq n\} \in LFSNP_*(rule_*, cons_1, forg_0)$.*

Proof. For each $x_i \in L$, $x_i = 0^{n_{i,1}}10^{n_{i,2}}1 \dots 10^{n_{i,r_i}}10^{n_{i,r_i+1}}$, there is a system as in the proof of Theorem 2, consisting of a neuron which outputs spikes in the moments which correspond to the digits 1 of x_i, and a companion neuron which just makes sure that the computation lasts $|x_i|$ steps. We combine such subsystems, each one taking care of one string x_i, into a system as that from Figure 6.

In the bottom of Figure 6 there are the modules (w_i, t_i) for generating x_i, $1 \leq i \leq n$, as in Figure 5, except that there is no spike in the neurons w_i; the corresponding sets of rules are denoted by $R_1, R_2, \dots, R_n$. If there is an i such that $|x_i|_1 = 0$ (note that in this case we have $R_i = \emptyset$), then we set $r_i = 1$ in the construction from Figure 6. The work of the system is triggered by neuron 1, which selects one of its rules to be applied non-deterministically in step 1: if the rule $a \to a; i$ is applied, then the module for generating x_{i+1} is activated, $i = 0, 1, , \dots, n-1$. The neurons c_i, $1 \leq i \leq n$, count the time steps, so that at step i one spike is sent to c_{i+1} and one to c'_i. All these spikes are forgotten, except the one which arrives in c'_i at the same time with the spike emitted by neuron 1, and these spikes load the necessary number of spikes in the corresponding working module w_i, and also send one spike to the timing neuron t_i. As in the proof of Theorem 2, these two neurons ensure the generation of x_i – with $i+3$ occurrences of 0 in the left hand, corresponding to the $i+2$ steps necessary for the spike of neuron 1 to reach the working neurons and to the step necessary for passing a spike from neuron w_i to neuron *out*.

We now pass to investigating the relationships with the family of regular languages, and we start with a result already proved by the considerations above.

Theorem 5. *The family of languages generated by finite SN P systems is strictly included in the family of regular languages over the binary alphabet.*

Proof. The inclusion follows from the fact that for each finite SN P system we can construct the corresponding transition diagram associated with the computations of the SN P system and then interpret it as the transition diagram of a finite automaton (with an arc labeled by 1 when the output neuron spikes and labeled by 0 when the output neuron does not spike) as already done in the example of Section 4. The strictness is a consequence of Theorem 1.

However, each regular language, over any alphabet, not only on the binary one, can be represented in an easy way starting from a language in $LFSNP_*(rule_*, cons_*, forg_*)$.

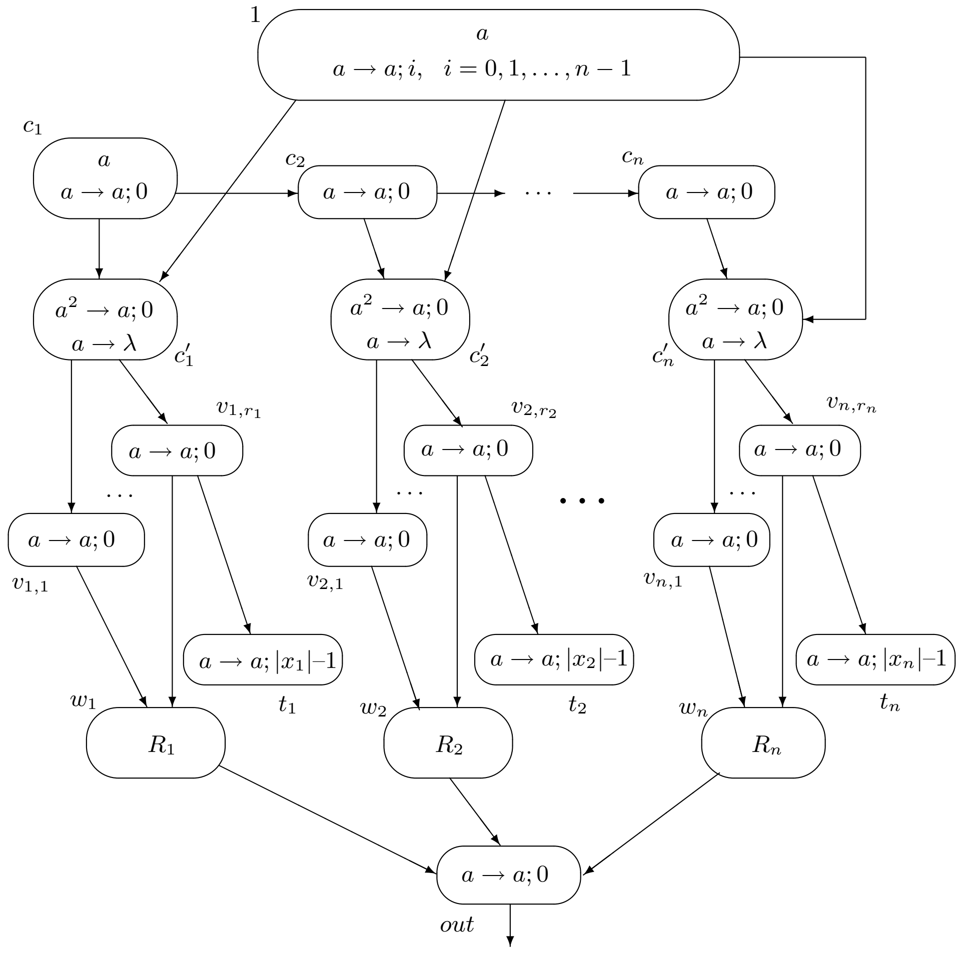

Fig. 6. An SN P system generating a finite set, prefixed by zeroes

Theorem 6. *For any language $L \subseteq V^*$, $L \in REG$, there is a finite SN P system Π and a morphism $h : V^* \longrightarrow B^*$ such that $L = h^{-1}(L(\Pi))$.*

Proof. Let $V = \{a_1, a_2, \ldots, a_k\}$ and let $L \subseteq V^*$ be a regular language. Consider the morphism $h : V^* \longrightarrow B^*$ defined by

$$h(a_i) = 0^{i+1}1, \; 1 \leq i \leq k.$$

The language $h(L)$ is regular. Consider a right-linear grammar $G = (N, B, S, P)$ such that $L(G) = h(L)$ and having the following properties:

1. $N = \{A_1, A_2, \ldots, A_n\}, n \geq 1$, and $S = A_n$,

2. the rules in P are of the forms $A_i \to 0^s 1 A_j$ or $A_i \to 0^s 1$, for $s \in \{2, 3, \ldots, k + 1\}, i, j \in \{1, 2, \ldots, n\}$.

A grammar with these properties can easily be found (the first property is a matter of renaming the nonterminals, the second property is ensured by the fact that the strings of $h(L)$ consist of blocks of the form $0^s 1$ for $2 \leq s \leq k + 1$).

For uniformity, let us assume that there exists a further nonterminal, A_0, and that all terminal rules $A_i \to 0^s 1$ are replaced by $A_i \to 0^s 1 A_0$, hence, all rules have the generic form $A_i \to 0^s 1 A_j$. It is important to note that we always have at least two occurrences of 0 in the rules.

We construct the following SN P system:

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \ldots, \sigma_{n+3}, n + 3),\\
\sigma_1 &= (2, \{a^2/a \to a; 0,\ a^2/a \to a; 1,\ a \to \lambda\}),\\
\sigma_2 &= (2, \{a^2/a \to a; 0,\ a \to \lambda\}),\\
\sigma_i &= (0, \{a \to a; 0\}), i = 3, 4, \ldots, n + 2,\\
\sigma_{n+3} &= (2n, \{a^{n+i}/a^{n+i-j} \to a; s \mid A_i \to 0^s 1 A_j \in P\}\\
&\quad \cup \{a^j \to a; k + 2 \mid 1 \leq j \leq n\}.
\end{aligned}$$

For an easier understandability, the system is also given graphically, in Figure 7.

The output neuron already fires in the first step, by a rule $a^{2n}/a^{2n-j} \to a; s$ associated with a rule $A_n \to 0^s 1 A_j$ from P (where A_n is the axiom of G) and its spike exits the system in step $s + 1$. Because $s \geq 2$, the neuron $n + 3$ is closed at least two steps, hence, no spike can enter from neurons $3, 4, \ldots, n + 2$ in these steps (this is true in all subsequent steps of the computation when rules of G are simulated).

The neurons 1 and 2 are meant to continuously "reload" neuron $n + 3$ with n spikes, through the intermediate "multiplier neurons" $3, 4, \ldots, n + 2$: as long as neuron 1 uses the rule $a^2/a \to a; 0$, neurons 1 and 2 send to each other a spike, returning to the initial state, while neuron 2 also sends a spike to all neurons $3, 4, \ldots, n + 2$. In each step, neuron 1 can however use the rule $a^2/a \to a; 1$. Simultaneously, neuron 2 spikes, but its spike does not enter neuron 1. In the next step, neuron 2 uses its forgetting rule $a \to \lambda$, and receives one spike from neuron 1. This spike is forgotten in the next step, and the work of neurons 1 and 2 ends. In this way, also the reloading of neuron $n + 3$ stops.

Let us now return to the work of neuron $n + 3$ and assume that we have $n + i$ spikes in it, for some $1 \leq i \leq n$ (initially, $i = n$). The only rule which can be used is $a^{n+i}/a^{n+i-j} \to a; s$, for $A_i \to 0^s 1 A_j \in P$. There remain j spikes; if the neuron receives n spikes from neurons $3, 4, \ldots, n + 2$ in step $s + 1$ (the spikes sent earlier are lost), then the output neuron ends the step $s + 1$ with $n + j$ spikes inside. If $j \geq 1$, then the simulation of rules in G can be repeated.

If in the moment when a rule $a^{n+i}/a^{n+i-0} \to a; s$ is applied (i.e., a rule $A_i \to 0^s 1 A_0$ is simulated) the output neuron does not receive further spikes from neurons $3, 4, \ldots, n + 2$, which means that neurons 1, 2 have finished their work, then no

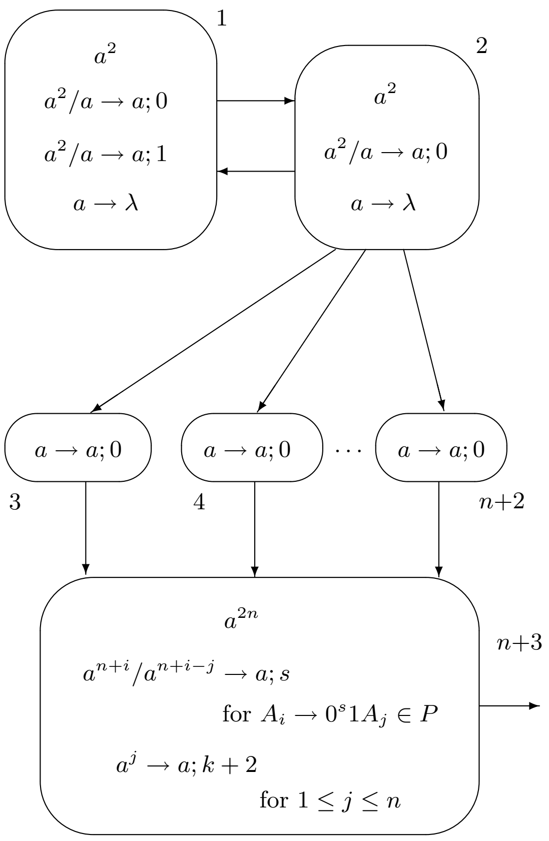

Fig. 7. The SN P system from the proof of Theorem 6

spike remains in the system and the computation halts. The generated string is one from $L(G) = h(L)$.

If the work of neurons 1 and 2 stops prematurely, i.e., in neuron $n+3$ we apply a rule $a^{n+i}/a^{n+i-j} \to a;s$ and no spike comes from neurons $3,4,\ldots,n+2$ in step $s+1$, then the rule $a^j \to a;k+2$ is immediately applied, and the computation stops after producing the string $0^{k+2}1$, as a suffix of the generated string.

Similarly, if after using a rule $a^{n+i}/a^{n+i-0} \to a;s$ we still receive spikes from neurons $3,4,\ldots,n+2$, then in the next step the rule $a^n \to a;k+2$ is used, and again the string $0^{k+2}1$ is introduced, possibly repeated several times before the computation halts.

Therefore, $h(L) \subseteq L(\Pi)$ and $L(\Pi) - h(L) \subseteq B^*\{0^{k+2}1\}^+$. Because a string containing a substring 0^{k+2} is not in $h(V^*)$ and because h is injective, we have $h^{-1}(L(\Pi)) = L$.

As expected, the power of SN P systems goes far beyond the regular languages. We first illustrate this assertion with an example, namely, the system Π_3 from Figure 8, for which we have $L(\Pi_3) = \{0^{n+4}1^{n+4} \mid n \geq 0\}$; observe that due to the rule $a(aa)^+/a^2 \to a; 0$ in neuron 10 this SN P system is not finite.

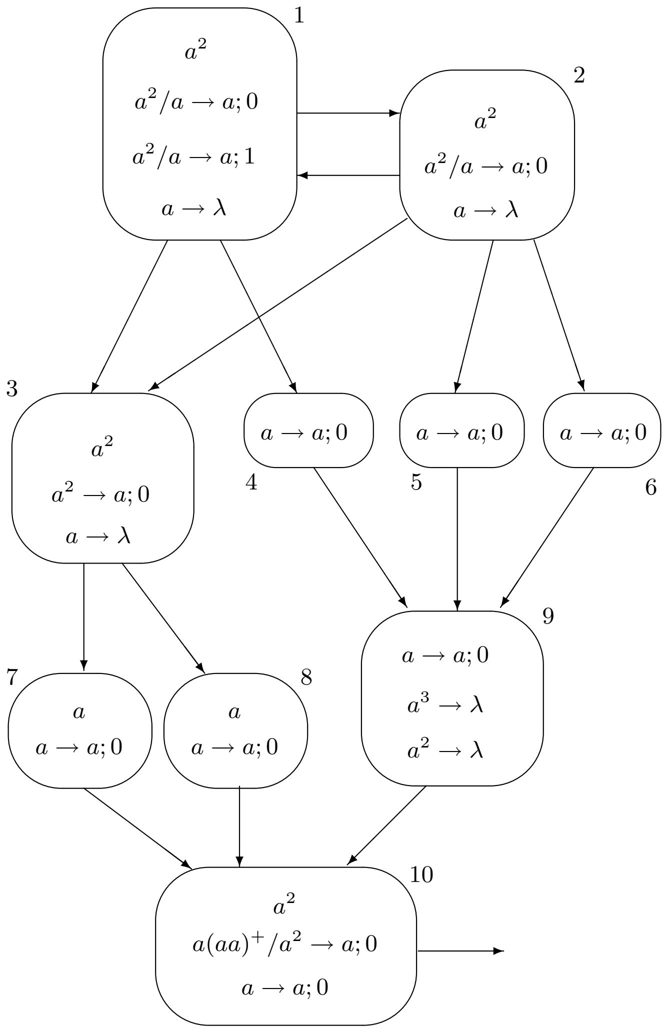

Fig. 8. An SN P system generating a non-regular language

The reader can check that in $n \geq 0$ steps when neuron 1 uses the rule $a^2/a \to a; 0$ the output neuron accumulates $2n + 6$ spikes. When neuron 1 uses the rule

$a^2/a \to a; 1$, one more spike will arrive in neuron 10 (in step $n+4$). In this way, the number of spikes present in neuron 10 becomes odd, and the rule $a(aa)^+/a^2 \to a; 0$ can be repeatedly used until only one spike remains; this last spike is used by the rule $a \to a; 0$, thus $n+4$ occurrences of 1 are produced.

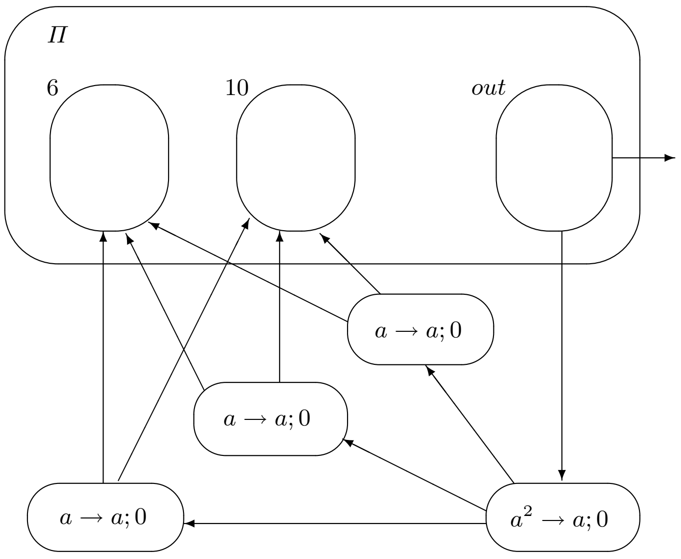

Fig. 9. A halting module (as used in the proof of Theorem 7)

Much more complex languages can be generated. First, the previous construction can be extended to non-context-free languages consisting of strings of the form $0^{n_1}1^{n_2}0^{n_3}$ with a precise relation between n_1, n_2, n_3. Then, languages with non-semilinear blocks in their strings can be generated.

Theorem 7. $LSNP_{22}(rule_3, cons_3, forg_3) - MAT \neq \emptyset$.

Proof. In Figure 14 from [7] one considers an SN P system Π (with 18 neurons, of the complexity described by $rule_3, cons_3, forg_3$) which produces all spike trains of the form $0^k10^{2^n}10w$, for any $n \geq 2$, and some $k \geq 1$ and an infinite binary sequence w. To this system from [7] we add a *halting module* as that suggested in Figure 9, which waits until receiving two spikes from the output neuron of Π, then sends three spikes to neurons 6 and 10 of Π; these neurons play an important rôle in iterating the work of Π, but they cannot handle more than two spikes. In this way, the work of Π is blocked after producing two spikes. (The same effect is obtained if we send three spikes to all neurons of Π, except the output one, so the reader should not mind which is the precise role of neurons 6 and 10 in Π.)

Thus, the obtained system, let us denote it by Π', will halt after sending out two spikes, hence, it generates a language $L(\Pi')$ included in $\{0\}^*\{10^{2^n}1 \mid n \geq 2\}\{0\}^*$ such that

$$((\{0\}^*\{1\})\backslash L(\Pi'))/(\{1\}\{0\}^*) = \{0^{2^n} \mid n \geq 1\}.$$

The family MAT is closed under right and left quotients by regular languages [1] and all one-letter matrix languages are regular [5], therefore $L(\Pi') \notin MAT$.

The strong restriction that in order to produce a string of length n we have to work exactly n steps (and the fact that the workspace used during these steps cannot increase exponentially) directly implies the fact that languages generated by SN P systems are recursive. Indeed, their membership problem can be solved in the following easy way: consider a string x and a system Π; start from the initial configuration of Π and construct the computation tree with $|x| + 1$ levels, then check whether there is a path in this tree which corresponds to a halting computation and which produces the string x. Therefore, we have the following result.

Theorem 8. $LSNP_*(rule_*, cons_*, forg_*) \subset REC$.

We do not know whether this result can be improved to the inclusion

$$LSNP_*(rule_*, cons_*, forg_*) \subset CS.$$

However, a characterization of recursively enumerable languages is possible in terms of languages generated by SN P systems.

Theorem 9. *For every alphabet* $V = \{a_1, a_2, \ldots, a_k\}$ *there are a morphism* $h_1 : (V \cup \{b, c\})^* \longrightarrow B^*$ *and a projection* $h_2 : (V \cup \{b, c\})^* \longrightarrow V^*$ *such that for each language* $L \subseteq V^*$, $L \in RE$, *there is an SN P system* Π *such that* $L = h_2(h_1^{-1}(L(\Pi)))$.

Proof. The two morphisms are defined as follows:

$$\begin{aligned} &h_1(a_i) = 10^i1, \text{ for } i = 1, 2, \ldots, k, \\ &h_1(b) = 0, \\ &h_1(c) = 01, \\ &h_2(a_i) = a_i, \text{ for } i = 1, 2, \ldots, k, \\ &h_2(b) = h_2(c) = \lambda. \end{aligned}$$

For a string $x \in V^*$, let us denote by $val_k(x)$ the value in base $k + 1$ of x (we use base $k+1$ in order to consider the symbols $a_1, \ldots, a_k$ as digits $1, 2, \ldots, k$, thus avoiding the digit 0 in the left hand of the string). We extend this notation in the natural way to sets of strings. Now consider a language $L \subseteq V^*$. Obviously, $L \in RE$ if and only if $val_k(L)$ is a recursively enumerable set of numbers. In turn, a set of numbers is recursively enumerable if and only if it can be accepted by a deterministic register machine. Let M be such a register machine, i.e., $N(M) = val_k(L)$.

We construct an SN P system Π performing the following operations (c_0 and c_1 are two distinguished neurons of Π, the first one being empty and the second one having three spikes in the initial configuration):

1. Output a spike in the first time unit.
2. For some $1 \leq i \leq k$, output no spike for i steps, but introduce the number i in neuron c_0; in the construction below, a number n is represented in a neuron by storing there $3n$ spikes, i.e., the previous task means introducing $3i$ spikes in neuron c_0.
3. When this operation is finished, output a spike (hence, up to now we have produced a string 10^i1).
4. Multiply the number stored in neuron c_1 (initially, we here have number 0) by $k+1$, then add the number from neuron c_0; specifically, if neuron c_0 holds $3i$ spikes and neuron c_1 holds $3m$ spikes, $m \geq 0$, we end this step with $3(m(k+1)+i)$ spikes in neuron c_1 and no spike in neuron c_0. In the meantime, the system outputs no spike (hence, the string was continued with a number of occurrences of 0; this number depends on the duration of the operation above, but it is greater than 1). When the operation is completed, output two spikes in a row (hence, the string is continued with 11).
5. Repeat from step 2, or, non-deterministically, stop the increase of spikes from neuron c_1 and pass to the next step.
6. After the last increase of the number of spikes from neuron c_1 we have got $val_k(x)$ for a string $x \in V^+$ such that the string produced by the system up to now is of the form $10^{i_1}10^{j_1}110^{i_2}10^{j_2}11\ldots110^{i_m}10^{j_m}$, for $1 \leq i_l \leq k$ and $j_l \geq 1$, for all $1 \leq l \leq m$, i.e., $h_1(x) = 10^{i_1}110^{i_2}1\ldots10^{i_m}1$. We now start to simulate the work of the register machine M in recognizing the number $val_k(x)$. During this process, we output no spike, but we output one if (and only if) the machine M halts, i.e., when it accepts the input number, which means that $x \in L$. After emitting this last spike, the system halts. Therefore, the previous string $10^{i_1}10^{j_1}110^{i_2}10^{j_2}11\ldots110^{i_m}10^{j_m}$ is continued with a suffix of the form 0^s1 for some $s \geq 1$.

From the previous description of the work of Π, it is clear that we stop, after producing a string of the form $y = 10^{i_1}10^{j_1}110^{i_2}10^{j_2}11\ldots110^{i_m}10^{j_m}0^s1$ as above, if and only if $x \in L$. Moreover, it is obvious that $x = h_2(h_1^{-1}(y))$: we have $h_1^{-1}(y) = a_{i_1}b^{j_1-1}ca_{i_2}b^{j_2-1}c\ldots a_{i_m}b^{j_m+s-1}c$ (this is the only way to cover correctly the string x with blocks of the forms of $h_1(a_i), h_1(b), h_1(c)$); the projection h_2 simply removes the auxiliary symbols b, c.

Now, it remains to construct the system Π.

Instead of constructing it in all details, we rely on the fact that a register machine can be simulated by an SN P system, as already shown in [7] – for the sake of completeness and because of some minor changes in the construction, we below recall the details of this simulation. Then, we also suppose that the multiplication by $k+1$ of the contents of neuron c_1 followed by adding a number between 1 and l is done by a register machine (with the numbers stored in neurons c_0, c_1 introduced in two specified registers); we denote this machine by M_0. 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 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.

A delicate problem which appears here is the fact that the simulations of both machines M_0 and M have to use the same neuron c_1, but the correct work of the system (the fact that the instructions of M_0 are not mixed with those of M) will be explained below.

The overall appearance of Π is given in Figure 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 indicates the subsystem which simulates the register machine $M = (m, H, l_0, l_h, I)$. Of course, we assume $H_0 \cap H = \emptyset$.

We start with spikes in neurons 6, 7, 8, and 18 (besides the three spikes from neuron c_1), hence, we spike in the first step. As long as neurons 6, 7 do not receive a spike from neuron 8, they spike and send a spike to each other and three spikes to neuron c_0.

If neuron 8 starts by using some rule $a \rightarrow a; i-1$, $1 \leq i \leq k$, then after $i-1$ steps a spike is sent from neuron 8 to all neurons 6, 7 (which stop working), 4, 5 (which load two spikes in neuron $l_{0,0}$, thus starting the simulation of the register machine M_0), and 17; from here, the spike goes to the output neuron 18, which spikes exactly in the moment when the simulation of M_0 starts.

Now, the subsystem corresponding to the register machine M_0 works a number of steps (at least one); after a while the computation in M_0 stops, by activating the neuron $l_{h,0}$ (this neuron will get two spikes in the end of the computation and will spike; see below). This neuron sends a spike to both neurons 9 and 10.

Neuron 9 is the one which non-deterministically chooses to continue the string (the case of using the rule $a \rightarrow a; 0$) or to stop growing the string and to pass to checking whether it is in our language (the case of using the rule $a \rightarrow a; 1$). If both neurons 9 and 10 spike immediately, then neuron 14 fires, but neuron 15 forgets the two spikes.

Neuron 14 sends a spike to the output neuron and one to neuron 16. In the next step, besides sending a spike outside, the system returns neurons 6, 7, 8, and 18 to the initial state (having one spike). This means that a sequence of two spikes are sent out, and the process continues by introducing another substring $0^i 1$ in the string produced by the system.

When neuron 9 uses the rule $a \rightarrow a; 1$, the spikes of neurons 10 and 9 arrive, in this order, in neuron 15, which spikes and sends the spikes to neuron 13. This neuron waits for the two spikes, and, after having both of them, spikes and thus sends two spikes to l_0, the initial label of the register machine M. We start simulating the work of this machine, checking whether or not the number stored in

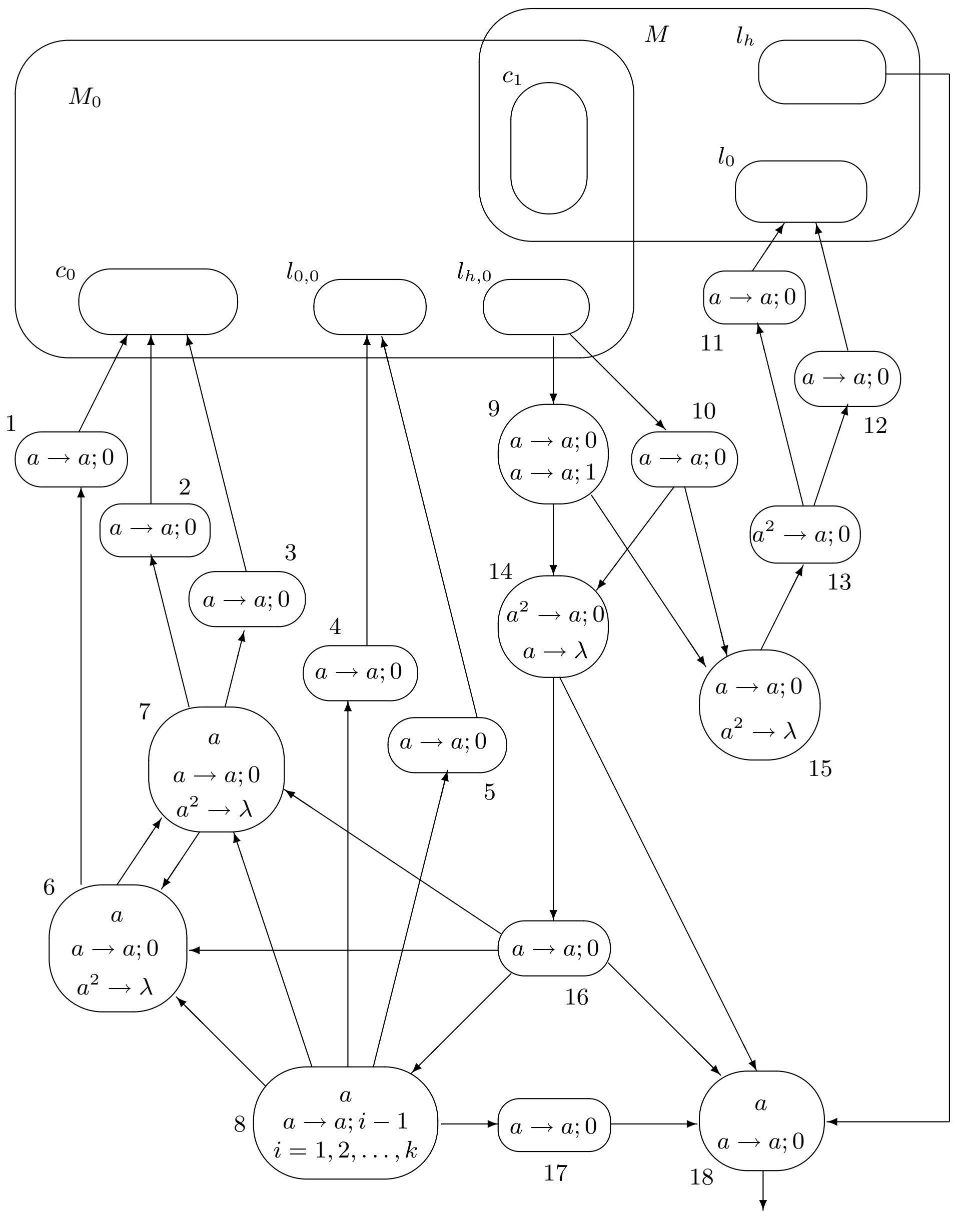

Fig. 10. The structure of the SN P system from the proof of Theorem 9

neuron c_1 belongs to $val_k(L)$. In the affirmative case, neuron l_h is activated, it

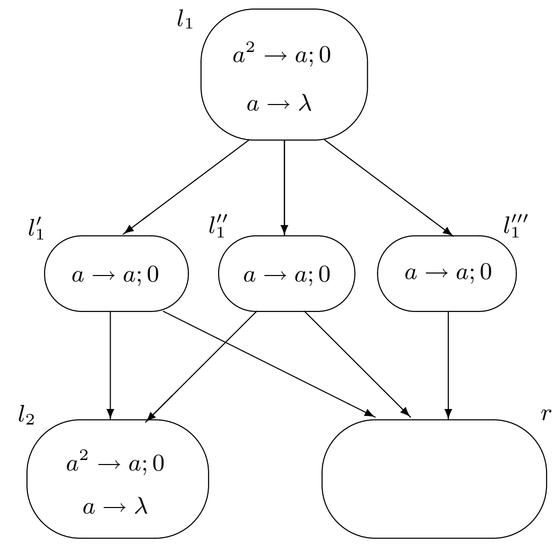

Fig. 11. Module ADD (simulating $l_1 : (\mathtt{ADD}(r), l_2)$)

sends a spike to the output neuron of the system, and the computation stops. If the number stored in neuron c_1 is not accepted, then the computation continues forever, hence, the system does not produce a string.

In order to complete the proof we have to show how the two register machines are simulated, using the common neuron c_1 but without mixing the computations. To this aim, we consider the modules ADD and SUB from Figures 11 and 12. Neurons are associated with each label of the machine (they fire if they have two spikes inside and forget a single spike), with each register (with $3t$ spikes representing the number t from the register), and there also are additional neurons with primed labels – 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. The instructions SUB of machine M_0 are simulated by modules as in the left side of Figure 12 and those of M by modules as in the right hand of the figure. The difference is that the rules for M_0 fire for a content of the neuron described by the regular expression $(a^3)^+a$ while the rules for M fire for a content of the neuron described by the regular expression $(a^3)^+a^2$ (that is why the module for M has two additional neurons, g_1', g_2'). This ensures the fact that the rules of M_0 are not used instead of those of M or vice versa. It is also important to note that the neurons corresponding to the labels of the register machines need two spikes to fire, hence, the unique spike sent by neuron c_1 (and by other neurons involved in subtraction instructions) to

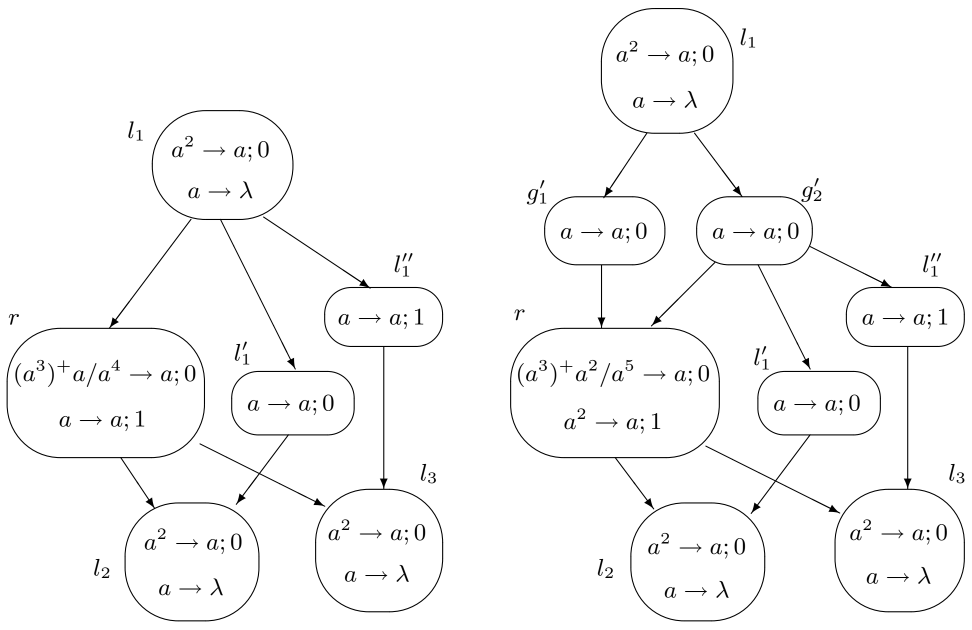

Fig. 12. Module SUB (simulating $l_1 : (\texttt{SUB}(r), l_2, l_3)$)

other neurons than the correct ones identified by the instruction are immediately forgotten.

With these explanations, maybe also following the description of the modules ADD and SUB from [7], the reader can check that the system Π works as requested.

The previous theorem given a characterization of recursively enumerable languages, because the family RE is closed under direct and inverse morphisms.

Corollary 2. *The family $LSNP_*(rule_*, cons_*, forg_*)$ is incomparable with all families of languages FL such that $FIN \subseteq FL \subset RE$ (even if we consider only languages over the binary alphabet) which are closed under direct and inverse morphisms.*

The system Π constructed in the proof of Theorem 9 depends on the language L, while the morphisms h_1, h_2 only depend on the alphabet V. Can this property be reversed, taking the system Π depending only on the alphabet V and the morphisms (or other stronger string mappings, such as a gsm mapping) depending on the language? A possible strategy of addressing this question is to use a characterization of RE languages starting from fixed languages, such as the twin-shuffle language [2] or the copy languages [3].

6 Final Remarks

We have considered the natural question of using spiking neural P systems as language generators, and we have investigated their power with respect to families in the Chomsky hierarchy. Several topics remain to be investigated, mainly, concerning possible changes in the definition of SN P systems, starting with considering different types of rules. For instance, what about using forgetting rules of the form $E/a^r \to \lambda$, with E being a regular expression like in firing rules? Another extension is to have rules of the form $E/a^c \to a^p; d$, with $p \geq 1$ (at least in the output neuron): when we output i spikes in one moment, we can record a symbol b_i in the generated string, and in this way we produce strings over arbitrary alphabets, not only on the binary one.

Another variant of interest is to consider a sort of rough-set-like rule, of the form $(s, S)/a^c \to a^p; d$, with $0 \leq s \leq c \leq S \leq \infty$. The meaning is that if the number of spikes from the neuron is k and $s \leq k \leq S$, then the rule fires and c spikes are consumed. This reminds the lower and the upper approximations of sets in rough sets theory [14], [15] and it is also well motivated from the neurobiology point of view (it reminds the sigmoid function of neuron exciting). It is interesting to note that all rules of the form $a^n/a^c \to a; d$ are of this type: just take $(n, n)/a^c \to a; d$. Thus, all examples and results until Theorem 6, including this theorem, are valid also for SN P systems with rough-set-like rules. Are there such systems able to generate non-regular languages? This is a question worth to be considered.

It is also of interest to see whether or not languages from other families can be represented starting from languages generated by specific classes of SN P systems and using various operations with languages (as we have done here with the regular languages – Theorem 6, and with recursively enumerable languages – Theorem 9). For instance, are there such representations – maybe using other operations and/or restricted/extended variants of SN P systems – for other families of languages from the Chomsky hierarchy, such as CF and CS?

Acknowledgements

The work of the first author was supported by the National Natural Science Foundation of China under Grants numbers 60573013 and 60421001. The work of the last two authors was supported by Project TIN2005-09345-C04-01 of the Ministry of Education and Science of Spain, cofinanced by FEDER funds.

References

1. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.
2. J. Engelfriet, G. Rozenberg: Fixed point languages, equality languages, and representations of recursively enumerable languages. *Journal of the ACM*, 27 (1980), 499–518.

3. R. Freund: Bidirectional sticker systems and representations of RE languages by copy languages. In *Computing with Bio-Molecules. Theory and Experiments* (Gh. Păun, ed.), Springer-Verlag, Singapore, 1998, 182–199.
4. W. Gerstner, W. Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
5. D. Hauschild, M. Jantzen: Petri nets algorithms in the theory of matrix grammars. *Acta Informatica*, 31 (1994), 719–728.
6. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. Submitted, 2006.
7. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
8. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
9. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
10. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
11. Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
12. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, to appear (also available at [18]).
13. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.
14. Z. Pawlak: *Rough Sets. Theoretical Aspects of Reasoning About Data*. Kluwer, Dordrecht, 1991.
15. Z. Pawlak: A treatise on rough sets. In *Transactions on Rough Sets* (J.F. Peters, A. Skowron, eds.), LNCS 3700, Springer-Verlag, Berlin, 2005, 1–17.
16. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.
17. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
18. The P Systems Web Page: `http://psystems.disco.unimib.it`.