INTERNATIONAL JOURNAL OF FOUNDATIONS OF COMPUTER SCIENCE

Volume 22, Number 1, January 2011

CONTENTS

Special Issue — Natural Computing: Theory and Applications

Preface 1
R. Freund, M. Gheorghe, S. Marcus, V. Mitrana and M. J. Perez-Jimmenez

On Strong Reversibility in P Systems and Related Problems 7
O. H. Ibarra

On String Languages Generated by Spiking Neural P Systems with Anti-Spikes 15
K. Krithivasan, V. P. Metta and D. Garg

Computation of Ramsey Numbers by P Systems with Active Membranes 29
L. Pan, D. Díaz-Pernil and M. J. Pérez-Jiménez

P Systems with Proteins on Membranes: A Survey 39
A. Păun, M. Păun, A. Rodríguez-Patón and M. Sidoroff

On a Partial Affirmative Answer for A Păun's Conjecture 55
I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez, M. A. Gutiérrez-Naranjo and M. Rius-Font

P Systems with Active Membranes Working in Polynomial Space 65
A. E. Porreca, A. Leporati, G. Mauri and C. Zandron

On the Power of Families of Recognizer Spiking Neural P Systems 75
P. Sosík, A. Rodríguez-Patón and L. Cienciala

Modeling Diffusion in a Signal Transduction Pathway: The use of Virtual Volumes in P Systems 89
D. Besozzi, P. Cazzaniga, S. Cocolo, G. Mauri and D. Pescini

Log-Gain Stoichiometric Stepwise Regression for MP Systems 97
V. Manca and L. Marchetti

A Simulation Algorithm for Multienvironment Probabilistic P Systems: A Formal Verification 107
M. A. Martínez-del-Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez and F. Sancho-Caparrini

An Overview on Operational Semantics in Membrane Computing 119
R. Barbuti, A. Maggiolo-Schettini, P. Milazzo and S. Tini

Formal Verification of P Systems Using Spin 133
F. Ipate, R. Lefticaru and C. Tudose

Small Universal TVDH and Test Tube Systems 143
A. Alhazov, M. Kogler, M. Margenstern, Y. Rogozhin and S. Verlan

Filter Position in Networks of Substitution Processors Does Not Matter 155
F. A. Montoro, J. Castellanos, V. Mitrana, E. Santos and J. M. Sempere

Functions Defined by Reaction Systems 167
A. Ehrenfeucht, M. Main and G. Rozenberg

P Systems and Topology: Some Suggestions for Research 179
P. Frisco and H. J. Hoogeboom

An Observer-Based De-Quantisation of Deutsch's Algorithm 191
C. S. Calude, M. Cavaliere and R. Mardare

PC Grammar Systems with Clusters of Components 203
E. Csuhaj-Varjú, M. Oswald and Gy. Vaszil

Orthogonal Shuffle on Trajectories 213
M. Daley, L. Kari, S. Seki and P. Sosìk

On the Number of Active Symbols in Lindenmayer Systems 223
J. Dassow and Gy. Vaszil

Positioned Agents in Eco-Grammar Systems 237
M. Langer and A. Kelemenová

Morphic Characterizations of Language Families in Terms of Insertion Systems and Star Languages 247
F. Okubo and T. Yokomori

Power Sums Associated with Certain Recursive Procedures on Words 261
A. Salomaa

Erratum 273

International Journal of Foundations of Computer Science
Vol. 22, No. 1 (2011) 15–27
© World Scientific Publishing Company
DOI: 10.1142/S0129054111007794

World Scientific
www.worldscientific.com

ON STRING LANGUAGES GENERATED BY SPIKING NEURAL P SYSTEMS WITH ANTI-SPIKES

KAMALA KRITHIVASAN*
Department of Computer Science and Engg., Indian Institute of Technology
Chennai, Tamilnadu, India
kamala@iitm.ac.in
http://www.cse.iitm.ac.in/ ∼ kamala/

VENKATA PADMAVATI METTA
Department of Computer Applications, Bhilai Institute of Technology
Durg, Chhattisgarh, India
vmetta@gmail.com

DEEPAK GARG
Department of Computer Science and Engg., Thapar University
Patiala, Punjab, India
deep108@yahoo.com

Received 15 June 2010
Accepted 29 September 2010
Communicated by Rudolf Freund

An Spiking Neural P system with anti-spikes uses two types of objects called spikes and anti-spikes which can encode binary digits in a natural way. The step when system emits a spike or an anti-spike is associated with symbol 1 and 0, respectively. Here we consider these computing devices as language generators. They allow non-determinism between the rules $a^c \to a$ and $a^c \to \overline{a}$, $c \in \mathbb{N}$, thus help to generate languages which cannot be generated using simple SN P systems.

Keywords: Spiking neural P system with anti-spikes; languages.

1991 Mathematics Subject Classification: 22E46, 53C35, 57S20

1. Introduction

Spiking neural P systems (shortly called SN P systems) introduced in [4] are mathematical models inspired by the neurobiological behaviour of neurons sending electrical pulses of identical voltages called spikes to neighbouring neurons through

*Kamala Krithivasan, BSB 353, IIT Madras, Chennai - 600 036, India.

synapses. An SN P system is represented as a directed graph where nodes correspond to the neurons having spiking rules and forgetting rules. The rules involve the spikes present in the neuron in the form of occurrences of a symbol a. The arcs indicate the synapses among the neurons. The spiking rules are of the form $E/a^r \rightarrow a$ and are used only if the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$, where $L(E)$ is the language represented by regular expression E. In this case a^r number of spikes are consumed and one spike is sent out. When neuron σ_i sends a spike, it is replicated in such a way that one spike is immediately sent to all neurons j such that $(i, j) \in syn$, where syn is the set of arcs between the neurons. The transmission of spikes takes no time, the spike will be available in neuron j in the next step. The forgetting rules are of the form $a^s \rightarrow \lambda$ and are applied only if the neuron contains exactly a^s spikes. The rule simply removes s spikes. For all forgetting rules, s must not be the member of $L(E)$ for any firing rule within the same neuron. A neuron is *bounded* if for every firing rule $E/a^r \rightarrow a$, E denotes a finite regular expression. An SN P system is called *bounded* if all the neurons in the system are *bounded.*

SN P system with anti spikes (or SN PA system) introduced in [7], is a variant of an SN P system consisting of two types of objects, spikes (denoted as a) and anti-spikes (denoted as $\overline{a}$). The inhibitory impulses/spikes are represented using anti-spikes. The anti-spikes behave in a similar way as spikes by participating in spiking and forgetting rules. They are produced from usual spikes by means of usual spiking rules; in turn, rules consuming anti-spikes can produce spikes or anti-spikes (here we avoid the rule anti-spike producing anti-spike). There is an additional fact that a and $\overline{a}$ cannot stay together, so annihilate each other. If a neuron has either objects a or objects $\overline{a}$, and further objects of either type(maybe both) arrive from other neurons, such that we end with a^r and $\overline{a}^s$ inside, then immediately an annihilation rule $a\,\overline{a} \rightarrow \lambda$, which is implicit in each neuron, is applied in a maximal manner, so that either a^{r-s} or $\overline{a}^{s-r}$ remain for the next step, provided that $r \geq s$ or $s \geq r$, respectively. This mutual annihilation of spikes and anti-spikes takes no time and the annihilation rule has priority over spiking and forgetting rules, so the neurons always contain either only spikes or anti-spikes. Like in [7], we avoid using rules $\overline{a}^c \rightarrow \overline{a}$, but not the other three types, corresponding to the pairs (a, a), $(a, \overline{a})$, $(\overline{a}, a)$. If we have a rule $E/b^r \rightarrow b'$ with $L(E) = \{b^r\}$, then we write it in the simplified form $b^r \rightarrow b'$.

The initial *configuration* of the system is described by $\mathcal{C} =< n_1, n_2, \cdots, n_m >$ where m is the number of neurons in the system and n_i is the initial number of spikes present in neuron i if $n_i > 0$ or initial number of anti-spikes if $n_i < 0$. A global clock is assumed in SN PA system and in each time unit, each neuron which can use a rule should do it (the system is synchronized), but the work of the system is sequential locally: only (at most) one rule is used in each neuron. Using the rules in this way, we pass from one configuration of the system to another configuration, such a step is called a transition. A computation is a finite or infinite sequences of transitions starting from the initial configuration. A computation halts if it reaches a configuration where no rule can be used. Note that the transition of configuration $\mathcal{C}$ is non-deterministic in the sense that there may be different rules applicable to $\mathcal{C}$.

An SN PA system can be used as a computing device in various ways. Here we use them as language generators. One of the neuron is considered as output neuron and it sends output to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the moments of time when an anti-spike emitted is marked with 0 and no output moments are just ignored. This binary sequence is called the spike train of the system- it might be infinite if the computation does not stop. With halting configurations, we associate a language, the binary strings describing the spike trains.

The complexity of an SN PA system Π is described as $LSNPA_m$ $(rule_k, cons_{p_1,p_2}, forg_{q_1,q_2})$, 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_1 spikes and p_2 anti-spikes and each forgetting rule removing at most q_1 spikes and q_2 anti-spikes. As usual a parameter m, k, p_1, p_2, q_1, q_2 is replaced with $*$ if it is not bounded. If the underlying SN PA systems are finite, we denote the corresponding families of languages by $LFSNPA_m(rule_k, cons_{p_1,p_2}, forg_{q_1,q_2})$.

The power of different variants of SN P systems as language generators are investigated in [3, 2, 1]. It was shown in [3] that some finite languages cannot be generated using simple SN P systems but it was proved in [2] that SN P systems with extended rules can generate the finite languages. SN PA system uses standard rules, adding one symbol at a time, but allows non-determinism between its rules like $a^c \to a$ and $a^c \to \overline{a}$, thus helps to generate languages that cannot be generated by simple SN P systems. In the present paper we address the power of SN PA systems as language generators, in particular, by considering bounded SN PA systems and comparing the languages generated with the results obtained in [6] for standard SN P systems.

Example 1.1
Consider the graphical representation of an SN P system with anti-spikes in Fig.1, the neurons are represented by nodes of a directed graph whose arrows represent the synapses; 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. It is formally denoted as

Π_1=(O, σ_1, σ_2, syn , 2), with

σ_1 = (-1, $\{\overline{a} \to a\}$), σ_2 = (2, $\{a^2/a \to \overline{a}\ ,\ a^2 \to a\}$), syn=$\{(1, 2), (2, 1)\}$.

The evolution of the system Π_1 can be analysed on a transition diagram as that from Fig.1(b) 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 the Fig.1(b), we have also indicated the rules used in each neuron with the following conventions; for each r_{ij} we have

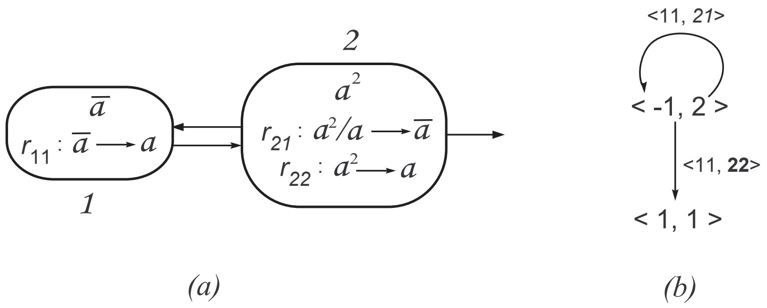

Fig. 1. SN P system with anti-spikes generating 0*1.

written only the subscript ij with *21* written in italics and **22** in bold in order to indicate that an anti-spike is sent to environment at steps when *21* is used and a spike when **22** is used; when a neuron i uses no rule, we write $i0$.

The functioning can easily be followed on this diagram, so that we only briefly describe it. We have two neurons, with labels 1, 2; neuron 2 is the output neuron. Initially neuron 1 has one anti-spike with a rule and neuron 2 has two spikes with two rules and non-determinism between its two rules. So the initial configuration of the system, $\mathcal{C}_0 =< -1, 2 >$.

The two neurons fire in the first step. Neuron 1 the first rule $\overline{a} \to a$ } and sends a spike(1) to neuron 2. Neuron 2 can choose any of its two rules and as long as it uses first rule, one spike is changed into anti-spike which will be sent to environment and neuron 1. In the next step the system will be in the same configuration. At any instance of time, starting from step 1, neuron 2 can choose its second rule, which consumes its two spikes and sends a spike to neuron 1 and environment. In the next step each neuron will have one spike, reaching the configuration $< 1, 1 >$ and the systems halts.

The transition diagram of a finite SN PA system can be interpreted as the representation of a non-deterministic finite automaton, with $\mathcal{C}_0$ being the initial state, the halting configurations being final states and each arrow being marked with 0 if in that transition the output neuron sends an anti-spike and with 1 if it sends a spike. In this way, we can identify the language generated by the system. In the case of finite SN P system Π_1, the language generated is 0^*1.

2. Languages Generated by SN P Systems with Anti-spikes

The following observations show that some finite languages and regular languages which cannot be generated using simple bounded SN P systems proved in [3] can be generated using bounded SN PA systems with one neuron. Here $B = \{0, 1\}$ is the binary set. B^+ is the set of all binary strings formed using the alphabet B.

2.1. *Finite binary languages*

Observation 1 *Languages of the form $L_{k,j} = \{0^k, 10^j\}$, for $k \geq 1$, $j \geq 0$ can be generated by bounded SN PA system.*

An SN PA system generating $L_{k,j} = \{0^k, 10^j\}$ is the following.

Π_1=($\{a, \overline{a}\}$, σ_1=($j + k, R_1$), $\emptyset$, 1), where

$R_1 = \{a^{j+k}/a^k \to a, a^{j+k}/a^{j+1} \to \overline{a}\} \cup \{a^l/a \to \overline{a} | 1 \leq l \leq j + k - 1\}$.

Theorem 1. *If $L = \{x\}$, $x \in B^+$, $|x|_1 = r \geq 0$, then $L \in LFSNPA_1(rule_{|x|}, cons_{1,0}, forg_{0,0})$, where $|x|$ is the length of the string x and $|x|_1$ is the number of occurrences of symbol 1 in x.*

Proof. Let us consider the string $x = 0^{n_1}10^{n_2} \cdots 0^{n_r}10^{n_{r+1}}$, for $n_j \geq 0$, $1 \leq j \leq r + 1$ (if $x = 0^{n_1}$, then $r = 0$). The SN P system from Fig.2 generates the string x. The output neuron initially contains $|x|$ spikes. The second set of rules $a^{|x|-(\sum_{i=1}^{j} n_i + j - 1)}/a \to a$ sends a spike(1) at $\sum_{i=1}^{j} n_i + j - 1$, $1 \leq j \leq r$ places where as the first set of rules allows an anti-spike to be sent out at other places. Depending upon the number of spikes available, a unique rule is used in each step to generate either spike or anti-spike, resulting $|x|$ rules. In the case $r = 0$, the system

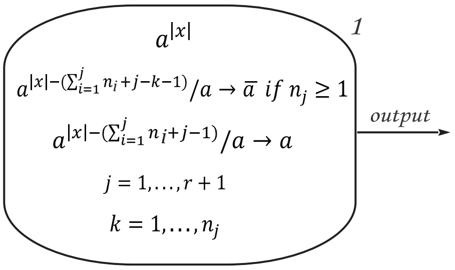
1
$a^{|x|}$
$a^{|x|-(\sum_{i=1}^{j} n_i + j - k - 1)}/a \to \bar{a}$ if $n_j \geq 1$
$a^{|x|-(\sum_{i=1}^{j} n_i + j - 1)}/a \to a$
$j = 1, \ldots, r + 1$
$k = 1, \ldots, n_j$
output

Fig. 2. SN P system with anti-spikes generating a singleton language.

cannot use the second set of rules as $|x| = n_1$. The first set of rules are used for $j = 1$ and $k = 1$ to n_1, outputting the string 0^{n_1}. □

Bounded SN P systems with standard rules cannot generate all binary finite languages, but with anti-spikes help in this respect.

Theorem 2. *$LFSNPA_1(rule_*, cons_{*,*}, forg_{*,*}) = BFIN$, $BFIN$ is the family of finite languages over binary alphabet.*

Proof. The inclusion $LFSNPA_1(rule_*, cons_{*,*}, forg_{*,*}) \subseteq BFIN$ can be easily proved. 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/anti-spikes present in the system at the beginning. Thus, the generated strings have a bounded length.

To prove the opposite inclusion $BFIN \subseteq LFSNPA_1(rule_*, cons_{*,*}, forg_{*,*})$, let us take a finite language, $L = \{x_1, x_2, \cdots, x_m\} \subseteq B^*, m \geq 1$, and let $x_j = 0^{s_{j,1}}10^{s_{j,2}} \cdots 10^{s_{j,r_j+1}}$ for $r_j \geq 0$, $s_{j,l} \geq 0$, $1 \leq l \leq r_j + 1$, $1 \leq j \leq m$.

Let $\mid x_j \mid = n_j$, $1 \leq j \leq m$ and $\alpha_j = \sum_{i=1}^{j} n_i$, $1 \leq j \leq m$

An SN PA system which generates the language L is the following.

Π=($\{a, \overline{a}\}$, $\sigma_1, \phi, 1$), $\sigma_1 = (\alpha_m, R_1)$

R_1=($\{a^{\alpha_m}/a^{\alpha_m-(\alpha_j-1)} \rightarrow b \mid b = \overline{a}$ if $s_{j,1} \geq 1$ and $b = a$ if $s_{j,1} = 0, 1 \leq j \leq m\}$ $\cup\{a^{\alpha_j-1-(\sum_{i=1}^{l} s_{i,l}+l-k-1}/a \rightarrow \overline{a} \mid s_{j,1} \geq 2, s_{j,l} \geq 1, 2 \leq l \leq r_j + 1, 1 \leq k \leq s_{j,l}, 1 \leq j \leq m\}$ $\cup\{a^{\alpha_j-1-(\sum_{i=1}^{l} s_{i,l}+l-1}/a \rightarrow a \mid 1 \leq l \leq r_j, 1 \leq j \leq m\}$ $\cup\{a^{\alpha_{j-1}} \rightarrow \lambda \mid 2 \leq j \leq m\}$)

Initially, only a rule $a^{\alpha_m}/a^{\alpha_m-(\alpha_j-1)} \rightarrow b$ can be used, and in this way it non-deterministically choose the string x_j to generate and output spike/anti-spike depending on the first bit of x_j. The neuron is left with $\alpha_j - 1$ spikes. The rules for generating the remaining bits are similar to rules of SN PA system in Theorem 1. After generating x_j, α_{j-1} spikes remain in the neuron, and are forgotten using the rule $a^{\alpha_{j-1}} \rightarrow \lambda$.

We observe 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 $k \neq j$. □

2.2. *Regular binary languages*

We now pass to investigating the relationships with the family of regular languages over the binary alphabet. It was proved in [6] that 0^*1 cannot be generated by any bounded SN P system. But in example 1.1 we have constructed SN PA system generating the language 0^*1.

Theorem 3. *$LFSNPA_*(rule_*, cons_{*,*}, forg_{*,*}) = BREG$, $BREG$ is the family of regular binary languages.*

Proof. The inclusion $LFSNPA_*(rule_*, cons_{*,*}, forg_{*,*}) \subseteq BREG$ follows from the fact that for each finite SN PA system, we can construct the corresponding transition diagram associated with the computations of the SN PA system and then interpret it as the transition diagram of a finite automaton (with an arc labeled by 1 when the output neuron sends a spike and labeled by 0 when the output neuron sends an anti-spike) as already done in the example of Section 1.

To prove the opposite inclusion that if $L \subseteq B^*$, $L \in BREG$, then $L \in LFSNPA_*(rule_*, cons_{*,*}, forg_{*,*})$, we consider the right-linear grammar $G = (N, T, S, P)$ such that $L = L(G)$ and having the following properties.

1. $N = \{A_1, A_2, \cdots, A_n\}$, $n \geq 1$ and $S = A_n$.
2. The rules in P are of the form $A_i \rightarrow 0A_j \mid 1A_j \mid 0 \mid 1$ where $i, j \in \{1, 2, \cdots, n\}$.

We construct the following SN P system:

Π=($\{a, \overline{a}\}$, σ_1, $\sigma_2, \cdots, \sigma_{n+1}$, *syn* , $n + 1$), with

$\sigma_i = (1, \{a \rightarrow a,\ a \rightarrow \overline{a}\ \})$, $i = 1, 2, \cdots, n$,

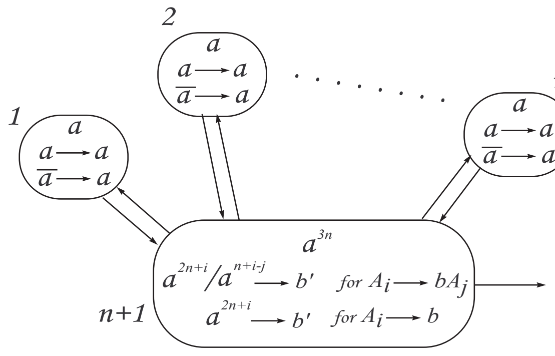
1
2
n
n+1
a
$a \rightarrow a$
$\overline{a} \rightarrow a$
a^{3n}
$a^{2n+i}/a^{n+i-j} \rightarrow b'$ for $A_i \rightarrow bA_j$
$a^{2n+i} \rightarrow b'$ for $A_i \rightarrow b$

Fig. 3. The SN PA system from the proof of Theorem 3.

$\sigma_{n+1} = (3n, \{a^{2n+i}/a^{n+i-j} \to b' \mid A_i \to bA_j \in P\} \cup \{a^{2n+i} \to b' \mid A_i \to b \in P\})$ where $b \in \{0,1\}$ and $b' = a$ if $b = 1$ and $b' = \overline{a}$ if $b = 0$,

syn=$\{(1, n+1), (n+1, 1), (2, n+1), (n+1, 2), \cdots, (n, n+1), (n+1, n)\}$.

For easier understandability, the system is also given graphically in Fig.3. The output neuron σ_{n+1} fires in the first step by a rule $a^{2n-j} \to b'$ (or $a^{3n} \to b'$) associated with a rule $A_n \to bA_j$ (or $A_n \to b$) from P, produces either a spike or an anti-spike depending upon whether $b = 1$ or $b = 0$ and receives n spikes from its neighbouring n neurons. The neurons 1 to n are meant to continuously load the neuron $n+1$ with n spikes, provided that they receive spike or an anti-spike from the output neuron.

Assume in some step t, the rule $a^{2n+i}/a^{n+i-j} \to b'$, for $A_i \to bA_j$ or $a^{2n+i} \to b'$ for $A_i \to b$ is used, for some $1 \leq i \leq n$, and n spikes are received from other neurons. If the first rule is used, then $n+i-j$ spikes are consumed and $n+j$ spikes remain in the output neuron. Then in the step $t+1$, we have $2n+j$ spikes in neuron σ_{n+1}, and a rule for $A_j \to bA_l$ or $A_j \to b$ can be used. In this step also the output neuron receives n spikes from its neighbouring neurons. In this way, the computation continues, unless the second rule is used.

If the second rule is used, then all spikes of the output neuron are consumed sending a spike or an anti-spike to other n neurons and n spikes are received from them. Then in the next step the output neuron again receives n spikes, but no rule is used, so no spike is produced. So it stops loading the other n neurons and the computation halts. In this way, all strings in L can be generated. □

The power of SN PA systems goes beyond the regular languages. We first illustrate this assertion with an example from Fig.4, that generates the language $L(\Pi) = \{0^n1^n \mid n \geq 1\}$; observe that the system is not finite due to the rule $(aa)^+a/a^2 \to \overline{a}$ in the neuron 3 and the output is delayed for two steps.

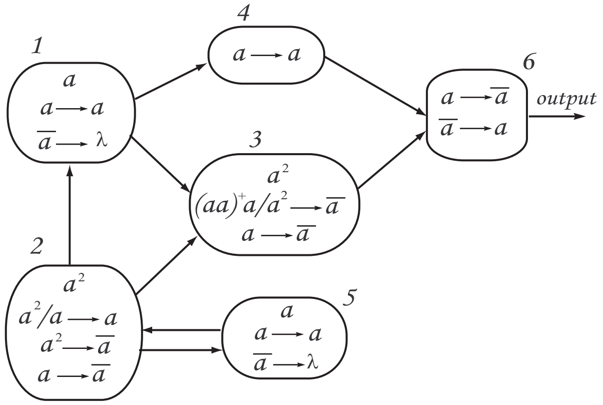

Fig. 4. An SN P system with anti-spikes generating a context free language.

The reader can check that in $n \geq 0$ steps when neuron 2 uses the first rule $a^2/a \to a$, the neuron 3 accumulates $2n+2$ spikes and neuron 1 sends a spikes to neuron 4, which in turn sends a spike to output neuron, which uses its first rule and sends an anti-spike(0) to environment. At any step $n \geq 1$, when the neuron 2 uses the rule $a^2 \to \overline{a}$, the spike from neuron 1 and anti-spike from neuron 2 will annihilate each other in neuron 3, remaining again with $2n+2$ spikes. Neuron 2 receives a spike from neuron 5 where as neuron 5 receives an anti-spike from neuron 2. In the same step spike from neuron 1 is also sent to neuron 4. In the $n+1$ step neuron 1 and 5 forget their anti-spikes received from neuron 2. Neuron 4 sends a spike to neuron 6. Neuron 2 uses its third rule $a \to \overline{a}$ by sending an anti-spike to neuron 3 and 5. Neuron 3 is left with $2n+1$ spikes and neuron 5 with an anti-spike which will be forgotten in the next step. In the $n+2$ step, the neuron 6 outputs a spike (that means total of $n+1$ spikes) and neuron 3 starts firing the as number spikes present becomes odd, and the rule $(aa)^+a/a^2 \to \overline{a}$ repeatedly used until one spike remains; this last spike is used by the second rule $a \to \overline{a}$. These $n+1$ anti-spikes are converted into spikes and sent to environment by the output neuron 6. Actually, much more complex languages can be generated by SN PA systems. The previous construction can be extended to non-context-free language like $\{0^n1^n0^n/n \geq 1\}$.

2.3. *A characterization of recursively enumerable languages*

A characterization of recursively enumerable (RE) languages is possible in terms of languages generated by SN PA systems. Here 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, l_h is the halt label (assigned to instruction HALT), and I is the set of instructions labelled in a one-to-one manner by the labels from H.

Theorem 4. *For every alphabet* $V = \{a_1, a_2, \cdots, a_s\}$ *there is a morphism* $h : V^* \to B^*$ *such that for each language* $L \subseteq V^*$, $L \in RE$, *there is an SN PA system* Π *such that* $L = h^{-1}(L(\Pi))$.

Proof. We follow here the same idea as in the proof of Theorem. 9 from [3] adapted to the case of anti-spikes.

The morphism is defined as follows:

$h(a_i) = 0^i 1$, for $i = 1, 2, \cdots, s$,

For a string $x \in V^*$, let us denote by $val_s(x)$, the value in base $s+1$ of x.(We use base $s+1$ in order to consider the symbols of $a_1, a_2, \cdots, a_s$ as digits $1, 2, \cdots s$, thus avoiding the digit 0 in the left hand of the string). We extend this notation in the natural way to the set of strings. Now consider a language $L \subseteq V^*$. Obviously $L \in RE$ iff $val_s(L)$ is 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 [5]. Let M be such a register machine that is $N(M) = val_s(L)$.

We construct an SN PA system Π performing the following operations (σ_{c0} and σ_{c1} are two distinguished neurons of Π, which are empty in the initial configuration):

1. Output i anti-spikes, for some $1 \leq i \leq s$, and at the same time introduce the number i in neuron σ_{c0} ; in the construction below, a number n is represented in a neuron by storing there $2n$ spikes, hence the previous task means introducing $2i$ spikes in neuron σ_{c0}.
2. When this operation is finished, output a spike hence up to now we have produced a string $0^i 1$.
3. Multiply the number stored in neuron σ_{c1} (initially, we have here number 0) by $s+1$, then add the number from neuron σ_{c0} ; specifically, if neuron σ_{c0} holds $2i$ spikes and neuron σ_{c1} holds $2n$ spikes, $n \geq 0$; then we end this step with $2(n(s+1)+i)$ spikes in neuron σ_{c1} and no spike in neuron σ_{c0} : In the meantime, the system outputs no spike/anti-spike.
4. Repeat from step 1, or, non-deterministically, stop the increase of spikes from neuron σ_{c1} and pass to the next step.
5. After the last increase of the number of spikes from neuron $c1$ we have got $val_s(x)$ for a string $x \in V^+$ such that the string produced by the system up to now is of the form $0^{i_1} 1 \lambda^{j_1} 0^{i_2} 1 \lambda^{j_2} \cdots 0^{i_m} 1 \lambda^{j_m}$, for $1 \leq i_l \leq s$ and $j_l \geq 1$, for all $1 \leq l \leq m$. λ is a symbol for no output, which is ignored. i.e., $h(x) = 0^{i_1} 1 0^{i_2} 1 \cdots 0^{i_m} 1$. We now start to simulate the work of the register machine M in recognizing the number $val_s(x)$. During this process, we output no spike, but the computation halts if (and only if) the machine M halts, i.e., when it accepts the input number, which means that $x \in L$.

From the previous description of the work of Π, it is clear that the computation halts after producing a string of the form $y = 0^{i_1} 1 \lambda^{j_1} 0^{i_2} 1 \lambda^{j_2} \cdots 0^{i_m} 1 \lambda^{j_m} \lambda^k$ as above, if and only if $x \in L$. Moreover, it is obvious that $x = h^{-1}(y)$: we have $h^{-1}(y) = a_{i_1} \cdots a_{i_m}$.

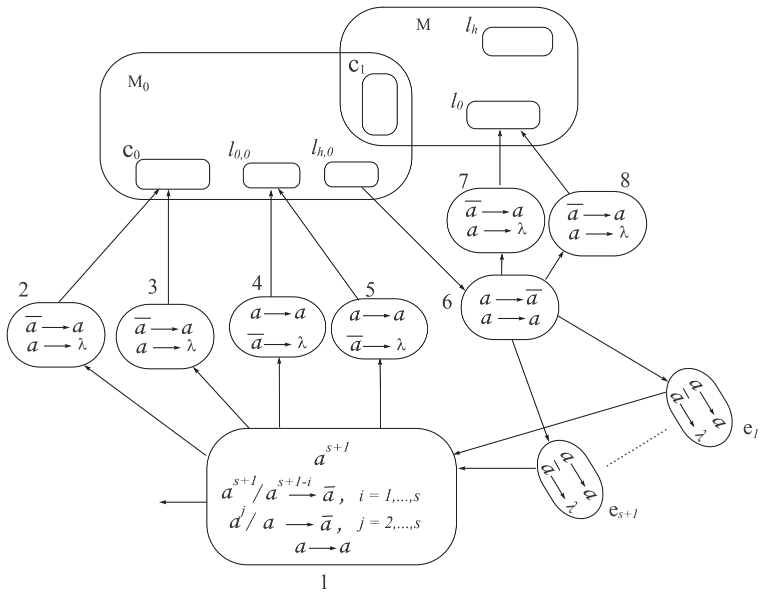
M
l_h
c_1
M_0
l_0
c_0
$l_{0,0}$
$l_{h,0}$
7
8
$\overline{a} \rightarrow a$
$a \rightarrow \lambda$
2
3
4
5
6
$a \rightarrow \overline{a}$
$a \rightarrow a$
$\overline{a} \rightarrow \lambda$
$a \rightarrow \overline{a}$
$\overline{a} \rightarrow \lambda$
e_1
a^{s+1}
$a^{s+1}/a^{s+1-i} \rightarrow \overline{a},\ i = 1,\ldots,s$
$a^j/a \rightarrow \overline{a},\ j = 2,\ldots,s$
$a \rightarrow a$
e_{s+1}
1

Fig. 5. The structure of the SN PA system from the proof of Theorem 5.

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 PA 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 $s + 1$ of the contents of neuron $c1$ followed by adding a number between 1 and s is done by a register machine (with the numbers stored in neurons $c0$, $c1$ 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 PA system. All other modules of the construction (introducing a number of spikes in neuron $c0$, 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 $c1$, 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 Fig.5, 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 $s+1$ spikes in neuron 1 and fires by using some rule $a^{s+1}/a^{s+1-i} \rightarrow \overline{a}$; $1 \leq i \leq s$, then in next $i - 1$ steps, it uses its second rule producing a total of number i anti-spikes and the last spike is used by the third rule producing spike, hence the first letter a_i of the generated string. In each step, when neuron 1 is producing an anti-spike, 2 spikes are sent to the neuron $c0$ through the neurons 2 and 3, accumulating a total of $2i$ spikes and the step when the output neuron produces a spike, it is ignored by neuron 2 and 3 and two spikes are sent to neuron $l_{0,0}$; 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 $c1$ with $s+1$ and adding i. When this process halts, neuron $l_{h,0}$ is activated (this neuron will get two spikes in the end of the computation and

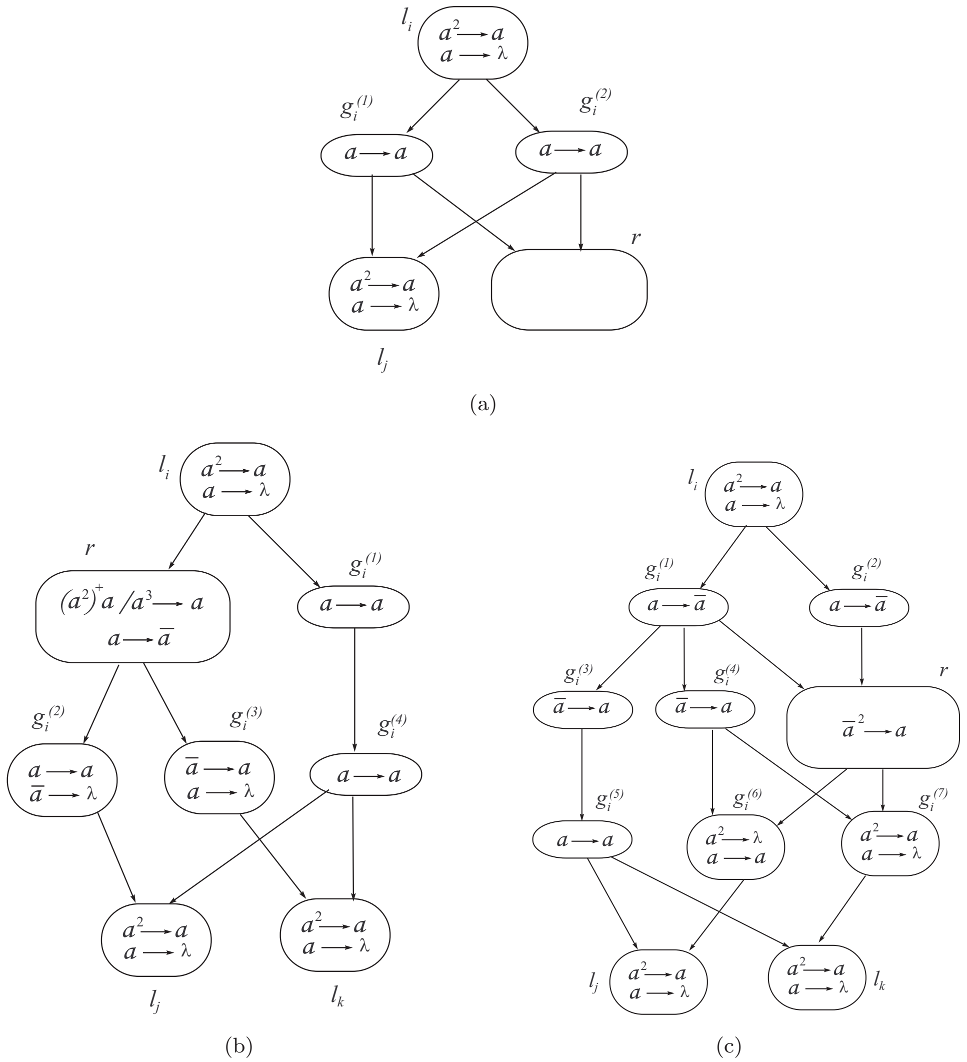

Fig. 6. (a) Module ADD (simulating $l_i : (ADD(r), l_j)$) for M and M_0, Module SUB (simulating $l_i : (SUB(r), l_j; l_k)$) (b) for machine M and (c) for machine M_0.

will spike), and in this way one spike is sent to neuron 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 in $L(M)$. In the first case, neuron 6 uses the rule $a \to a$; which makes neurons $e_1, \cdots, e_{s+1}$ spike; these neurons send $s+1$ spikes to neuron 1, like in the beginning of the computation. In the latter case, neuron 6 uses the rule $a \to \overline{a}$; which in turn activates the neuron 7 and 8, they activate l_0 by sending two spikes to it, thus starting the simulation of the register machine M. The computation of Π stops if and only if $val_s(x)$ is accepted by M. In order to complete the proof we need to show how the two register machines are simulated, using the common neuron $c1$ but without mixing the computations. To this aim, we consider the modules ADD and SUB from Fig.6. Neurons are associated with each label of the machine (they fire if they have two spikes inside) and with each register (with $2n$ spikes representing the number n from the register), there also are additional neurons with labels ${g_1}^i$, $i \geq 1$ it is important to note that all these additional neurons have distinct labels.

The simulation of $(ADD(r), l_j)$ (add 1 to register r and then go to the instruction with label l_j) instruction is easy, we just add two spikes to the respective neuron; no rule is needed in the neuron Fig. $6(a)$. The $(SUB(r), l_j, l_k)$ (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) instructions of machines M and M_0 are simulated by modules as in Fig. 6(b) and 6(c), respectively. Note that the rules for M fire for a content of the neuron r described by the regular expression $(a^2)^*a$ and the rules for M_0 fire for a content of the neuron r described by the regular expression $(\overline{a})^2$. This ensures the fact that the rules of M_0 are not used instead of those of M or vice versa. With these explanations, the reader can check that the system Π works as requested. $\square$

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

3. Conclusion

We have investigated here the power of SN PA systems with standard rules as language generators. We have proved characterizations of finite and regular languages over binary alphabet. We can extend the proofs to any alphabet by considering the morphisms. We have also proved a characterization of recursively enumerable languages. Here we ignored the no output steps. Finding representations of languages over three letter alphabet: no output, producing spikes and producing anti-spikes remains as a research topic.

References

[1] Gh. Păun, M.J. Pérez-Jiménez and G. Rozenberg: Spike trains in spiking neural P systems, *Int. J Found Comput Sci*, **17** 4 2006, 975–1002.

[2] H. Chen, M. Ionescu, T-O. Ishdorj, A. Păun, Gh. Păun and M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: universality and languages, *Nat Comput*, **7** 2008, 147–166.

[3] H. Chen, R. Freund, M. Ionescu, Gh. Păun and M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems, *Fundamenta Informaticae* **75** 14 2007, 141–162.

[4] M. Ionescu, Gh. Păun and T. Yokomori: Spiking Neural P Systems, *Fundamenta Informaticae* **71** 2-3 2006, 279–308.

[5] M. Minsky: *Computation finite and infinite machines*, Prentice Hall, Englewood Cliffs, NJ, (1967).

[6] O.H. Ibarra and S. Woodworth: Characterizations of some classes of spiking neural P systems, *Nat Comput*, **7** 2008, 499–517.

[7] P. Linqiang and Gh. Păun: Spiking Neural P Systems with Anti-Spikes, *Int. J. of Computers, Communications and Control*, **4** (2009) 3 273–282.