ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **15**, Number 4, 2012, 301–318

Control Languages Associated with Spiking Neural P Systems

Ajeesh RAMANUJAN, Kamala KRITHIVASAN

Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai - 36
E-mail: `ajeeshramanujan@gmail.com, kamala@iitm.ac.in`

Abstract. We consider labeled spiking neural P systems, which are usual spiking neural P systems with a label associated with every rule; the labels are symbols of a given alphabet or can be λ (empty). The rules used in a transition should have either the empty label or the same label from the chosen alphabet. In this way, a string is associated with each halting computation, called the *control word* of the computation. The set of all control words associated with computations in a given spiking neural P system form the *control language* of the system.

We study the family of control languages of spiking neural P systems in comparison with the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages. In the restricted case when in each step at least one rule with a non-empty label is used, every regular language is a control language, there are context-sensitive non-context-free languages of this type, but not all context-free languages are control languages of a spiking neural P system. All languages that are accepted by labeled spiking neural P systems are context-sensitive. If transitions with all rules labeled with λ are allowed, then each recursively enumerable language can be the control word of a spiking neural P system.

1. Introduction

Spiking neural P systems (shortly called SN P systems) are distributed parallel computing devices introduced in [1] as a class of membrane systems (also known as P systems) [3, 2] inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons. In SN P systems, the neurons are placed in the nodes of a directed graph, called the synapse graph. Every neuron may contain a number of copies of a single object called the spike (denoted with a), and also a number of firing and forgetting rules. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses which are accumulated at the target neurons. The application of each rule is determined by checking the contents of the neuron against a regular expression associated with the rule. In each step, if a neuron can use more than one rule, then one of them is non-deterministically chosen and applied. Thus, the rules are used in a sequential manner in each neuron, but neurons function in a parallel manner. More information about the SN P systems can be found in the Handbook [4] and in the membrane computing website [17].

SN P systems can be used as language generating devices in various ways – see, *e.g.*, [7, 8, 9, 10, 11]. In all these papers (and several others), words are associated with the sequence of spikes emitted by the system (this sequence is called spike train), that is only binary words are obtained.

In this paper, strings over arbitrary alphabets are obtained, in the form of control words associated with computations in a spiking neural P system whose rules are labeled. The label of a rule can either be a symbol in a given alphabet, or it can be empty (denoted by λ). In each step, besides rules with label λ only rules with the same non-empty label can be used (that is, a computation step is described either by a symbol in the case when at least one rule has a non-empty label, or by λ, in the case when all used rules have the empty label).

This idea was initially considered in [13] and then investigated in [14] for SN P systems with anti-spikes. We continue here this study for the case when standard SN P systems are used (without anti-spikes).

In some sense, we may assume that the control language associated with an SN P system is *accepted* by this system. Specifically, a string over the label alphabet is assumed to be placed in the environment of the system. This string is read from left to right, one symbol at a single step, in the following sense: if an input symbol is read, then in that computation step we can use only rules labeled with that symbol or with λ; if no symbol is read (the "reading head" can remain still, not moving along the input string), then only rules with the empty label can be used at that step. The string is accepted if and only if the computation of the system reaches a *final configuration* in the moment when the whole the string is read. (The use of a final configuration is another new feature used in this paper.)

In what follows we will use this terminology, calling the control languages *languages accepted* by SN P systems, in the sense specified above. Note however the essential difference between this notion and the languages defined by accepting SN P systems, in the "standard" sense – see references in [4].

The difference between the case when λ moves (when only rules with the empty label are used) are allowed and the case when such moves are not allowed is essential: in the former case all recursively enumerable languages are accepted, while in the latter case only a subset of context-sensitive languages are accepted, without covering the family of context-free languages.

The paper is organized as follows. In Section 2, we provide the necessary automata theory prerequisites. In Section 3, we give the definition of an SN P system as introduced in [1]. In Section 4 we introduce and define the labeled SN P systems and their control/accepted languages. In Section 5, we study the families of languages accepted by labeled SN P systems.

2. Basic Definition

In this section we give some definitions and notations related to automata theory.

Let Σ be a finite set of symbols called an alphabet. A string w over Σ is a sequence of symbols from Σ. λ denotes the empty string. The set of all strings over Σ is denoted by Σ^*. The length of a string $w \in \Sigma^*$ is denoted by $|w|$. A language L over Σ is a set of strings over Σ. The family of finite, regular, context-free, context-sensitive, and recursively enumerable languages are denoted by FIN, REG, CF, CS, and RE respectively.

The regular expressions over a given alphabet Σ are defined as follows: (i) $\emptyset, \lambda$, and each $a \in \Sigma$ are regular expressions representing the regular sets $\emptyset, \{\lambda\}$, and $\{a\}$ respectively; (ii) if E_1 and E_2 are regular expressions over Σ representing the regular sets R_1 and R_2, then $E_1 + E_2, E_1E_2$, and E_1^* are regular expressions over Σ representing the regular sets $R_1 \cup R_2, R_1R_2$, and R_1^* respectively, and (iii) nothing else is a regular expression over Σ. With each regular expression E, we associate a language $L(E)$; such a language is said to be *regular.*

A register machine is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (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:

– $l_i : (ADD(r), l_j)$ (add 1 to register r and then go to the instruction with label l_j),

– $l_i : (SUB(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),

– $l_h : HALT$ (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 0), 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., [12]) that register machines (even with only three registers, but this detail is not relevant in what follows) accept all sets of numbers which are Turing computable.

In this paper all the vectors are row vectors and written in boldface letter. The jth component of a vector $\mathbf{u}$ is denoted by $\mathbf{u_j}$.

3. Spiking Neural P Systems (SN P systems)

In this section we give the definition of an SN P system as introduced in [1].

Definition 1. An *SN P system of degree* $m \geq 1$ is a construct of the form $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$, where:

1. $O = \{a\}$ is the alphabet. a is called the *spike*;
2. $\sigma_1, \sigma_2, \cdots, \sigma_m$ are neurons of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where
   (a) $n_i \geq 0$ is the initial number of spikes contained in the neuron σ_i;
   (b) R_i is a finite set of rules of the forms:
      i. (firing rules) $E/a^c \to a; d$, where E is a regular expression over O, and $c \geq 1, d \geq 0$ are integer numbers;
      ii. (forgetting rules) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \to a; d$, of type (i) from R_i :
3. $syn \subseteq \{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$;
4. $out \in \{1, 2, \cdots, m\}$ indicates the output neuron.

A firing rule $E/a^c \to a; d \in R_i$, can be applied in neuron σ_i if it contains $k \geq c$ spikes, and $a^k \in L(E)$. The application of this rule removes c spikes from σ_i (thus leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$, the the spike is immediately emitted, otherwise it is emitted after d steps. During these d steps, the neuron is closed, and 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), and cannot fire. A forgetting rule $a^s \to \lambda$ can be applied in neuron σ_i if it contains exactly s spikes. The application of this rule simply removes s spikes from σ_i.

A generalization of SN P systems known as extended spiking neural P systems (ESN P system) is introduced in [10]; in these systems the rules are of the form $E/a^c \to a^p; d, p \geq 0, p \leq r$. Such a rule operates in the same manner as before except that after firing it sends p spikes along each outgoing synapse. When $p = 1$, the extended rules reduce to the standard rules and $p = 0$ is a special case of forgetting rules.

Rules with delay zero are written in the form $E/a^c \to a$. A neuron is bounded if in every firing rule $E/a^j \to a; d, E$ denotes a finite regular expression, *i.e.*, all firing rules are of the form $a^i/a^j \to a; d$, where $1 \leq j \leq i$. An SN P system is bounded if all the neurons in the system are bounded.

A configuration of an SN P system $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$ is a tuple $c = (n_1/d_1, n_2/d_2, \cdots, n_m/d_m)$, where $n_i \in \mathbb{Z}^+, 1 \leq i \leq m$, denotes the number of spikes in neuron i and $d_i \in \mathbb{Z}^+, 1 \leq i \leq m$, denotes the delay, *i.e.*, the number of steps until each neuron will be open again. In each time unit (a global clock is assumed), each neuron which can apply at least one rule, has to non-deterministically choose one and apply it. The neurons evolve in parallel, synchronously. Using the rules in this way, we can define transitions among the configurations. Any sequence of transitions starting from the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be applied. The output neuron also sends spikes to the environment. A binary string is associated with a halting computation in the following way: if in a time unit a spike is emitted by σ_{out}, then we write 1; if no spike is emitted by σ_{out}, then we write 0. This binary string is also called the *spike train* of Π. Various numbers can be associated with a spike train, or it can be considered as a string generated by the system. (If an input neuron σ_{in} is also considered, then an SN P system can be used in the accepting mode: a spike train is introduced in σ_{in}, symbol by symbol, and this string is accepted if and only if the computation halts after reading all the string. We will not consider here this kind of accepting systems, but we will deal with control words associated with computations, and we will call such words as being "accepted".)

4. Labeled Spiking Neural P System(LSN P System)

In this section we define the types of SN P systems we investigate in this paper, *i.e.*, the labeled SN P systems.

Consider an SN P system $\Pi = (\{a\}, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$ with m neurons. Assume a total ordering of the neurons. Let the total number of rules be n. Assume a total ordering $d_1, d_2, \cdots, d_n$ of the rules of Π in such a way that the rules in the same neuron are given consecutive numbers. Let $R = \{d_1, d_2, \cdots, d_n\}$ and Σ be a finite alphabet. Consider a function $l : R \to \Sigma \cup \{\lambda\}$, called the labeling function, that assigns a label to every rule in R.

With every rule $d_i, 1 \leq i \leq n$, we associate a vector called modification vector denoted by $\mathbf{m_{d_i}} \in \mathbb{Z}^m$ which denotes the change in the number of spikes in each neuron on the application of the rule d_i. For example consider a rule $d_k : a \to a$ in neuron l. The modification vector corresponding to $d_k, \mathbf{m_{d_k}}$ is vector with a -1 as its lth component since it takes one spike from neuron l, 1 in the ith component for every i such that $(l, i) \in syn$ since the application of the rule transfers one spike to every other neuron connected to it and 0 elsewhere. This is for a rule with zero delay. See [16] for handling modification vectors for rules with nonzero delay.

Definition 2. *A labeled SN P system*, in short, *an LSN P system*, is a construct $N = (\Pi, \Sigma, F, l)$, where

1. Π is an SN P system,
2. Σ is finite alphabet (the input alphabet),
3. F is a finite set of configurations (called final),
4. l is a labeling function.

With such a system, we associate a language as follows.

Given a configuration $(n_1, n_2, \cdots, n_m)$ of Π and a symbol $b \in \Sigma$, we define a transition relation $\Rightarrow$ on configurations as follows: $(n_1, n_2, \cdots, n_m) \overset{b}{\Rightarrow} (n'_1, n'_2, \cdots, n'_m)$, if there exists a set of rules $d_{i_1}, d_{i_2}, \cdots, d_{i_j}, 1 \le i_j \le n, 1 \le j \le m$, $l(d_{i_j})$ is either b or λ, *i.e.* we use only rules with the same label b and rules labeled with λ (we say that the transitions are label restricted), with modification vectors $\mathbf{m_{d_k}}, 1 \le k \le i_j$ and $n'_l = n_l + \sum_{l=1}^{i_j} (\mathbf{m_{d_k}})_l$.

In a label restricted transition, we say that the symbol b is an *input symbol*, and it is "consumed" if at least one rule with label b is used. If all the used rules have the label λ, then the input symbol is not consumed.

A string of input symbols (over Σ) is said to be accepted if all its symbols are consumed and Π reaches a configuration in the set F.

The set of all strings accepted in this way by computations in as LSN P system N is denoted by $L_\lambda(N)$. The subscript indicates the fact that λ steps (all rules applied in one step can have λ labels) are permitted. When only steps where at least one rule with a non-empty label is used, the accepted language is denoted by $L(N)$.

The family of languages $L(N)$ associated with LSN P systems N with at most m neurons is denoted by $LLSNP_m$. In the unrestricted case, the corresponding language family is denoted by $L_\lambda LSNP_m$. If the number of neurons is unbounded, then the subscript m is replaced with $*$. If the SN P system is an ESN P system, the superscript e is added, thus obtaining the families $L^eLSNP_m, L^e_\lambda LSNP_m$, respectively.

Note that in LSN P systems the input and the output neuron plays no role, hence they are omitted. Moreover, whenever possible, we associate directly the labels to rules, writing $b : E/a^c \to a^p; d$ instead of writing $l(E/a^c \to a^p; d) = b$.

We also use the convention that two language processing devices – generating or accepting – are equivalent if they characterize languages which differ at most in the empty string (otherwise stated, the string λ is ignored when comparing two language processing devices).

4.1. The Working of an LSN P System

We can imagine that the input string is placed on an "input tape" in the environment. The string is accessible to all the neurons in the system. There is an input pointer which points to the current symbol to be read. The input pointer is allowed to move only from left to right. The symbol read is available to all the neurons in the system.

Given a string $w = b_1 b_2 \cdots b_k \in \Sigma^*$, an LSN P system processes the string w as follows: It starts from an initial configuration, scans the input string from left to right, symbol by symbol, and moves from one configuration to another one by selecting the rules according to the labeling function. Each neuron can apply a rule labeled with the current input symbol if possible. If a neuron has no rules labeled with the current input symbol, it can use a λ labeled rule, if such a rule is applicable. This process is continued until the string is processed completely. If during a computation step only rules with λ labels are used, then no input symbol will be consumed and the symbol pointed by the input pointer will be available for use in the next step. This is possible only in the unrestricted case. The LSN P system $N = (\Pi, \Sigma, F, l)$ accepts the string

w if there exists a computation $c_0\beta_1c_1\beta_2\cdots\beta_kc_f$ with $c_f \in F$ and $l(\beta_1\beta_2\cdots\beta_k) = w$.

We illustrate the previous discussion with an example.

Example 1. Consider the context-free non-regular language $L_1 = \{b^nc^n \mid n \geq 1\} \cup \{c\}$. We define an LSN P system $N_1 = (\Pi_1, \{b, c\}, (2, 2, 0, 0), l)$ accepting L_1 as follows:

$\Pi_1 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, syn)$ with:

$$\begin{aligned}
&\sigma_1 = (1, \{b : a \to a\}),\\
&\sigma_2 = (1, \{b : a \to a\}),\\
&\sigma_3 = (0, \{c : a^*/a \to a\}),\\
&\sigma_4 = (1, \{c : a \to a\}),\\
&syn = \{(1, 2), (2, 1), (2, 3), (4, 1), (4, 2)\}.
\end{aligned}$$

The LSN P system N_1 is diagrammatically shown in Figure 1.

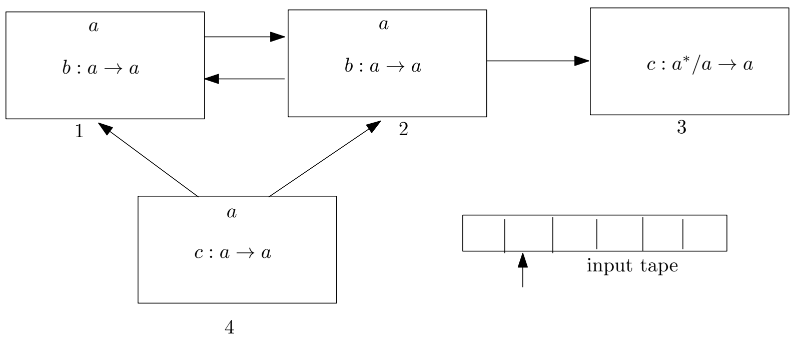


Fig. 1. An LSN P system accepting $\{b^nc^n \mid n \geq 1\} \cup \{c\}$.

The system N_1 processes an input string $bbcc$ as follows: The system starts in the initial configuration $(1, 1, 0, 1)$ and the current input symbol is b. The system uses the rule in σ_1 and σ_2, sending one spike to σ_2, σ_1 and σ_3. We cannot use the rule in σ_3 since the label of the rule is c and the current input symbol is b. So after processing the first b, the system reaches the configuration $(1, 1, 1, 1)$ and the input pointer points to the next symbol. On seeing the second b, the system uses the rule in σ_1 and σ_2, sending one spike to σ_2, σ_1 and σ_3 and the system reaches the configuration $(1, 1, 2, 1)$. Again the system cannot use the rule in σ_3. On seeing the first c, the system uses the rule in σ_3 and σ_4 sending one spike to neurons σ_1 and σ_2 from σ_4 and the spike send from the neuron σ_3 goes to the environment. The system cannot use any rule in σ_1 and σ_2 during this step since the rule in those neurons are labeled with b and the current input symbol is c. So after processing the first c, the system reaches the configuration $(2, 2, 1, 0)$ and the input pointer points to the next symbol. The next input symbol c is processed in the same way the first c is processed during the third step and after processing the second c the system reaches the configuration $(2, 2, 0, 0)$. The string is processed completely and since the system is in a final configuration the input string is accepted by the system. We can see that, if the string is not of the form $b^n c^m$ with $n = m$, the system is not able to apply any rule, hence is not able to reach the final configuration.

We can easily see that the system accepts the language $\{b^n c^n \mid n \geq 1\} \cup \{c\}$.

5. The Power of LSN P Systems

In this section we investigate the relationship between the family of languages accepted by LSN P systems and the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages.

Theorem 1. $FIN \subset LLSNP_1$.

Proof. Let $L \in FIN$. Let $L = \{w_1, w_2, \cdots, w_k\}$. Let $l_i = |w_i|$ and $l = l_1 + l_2 + \cdots + l_k$. Let $w_i = b_{i1} b_{i2} \cdots b_{il_i}$. Define a function $f(b_{ij}) = \sum_{r=1}^{i-1} l_r + j$ that injectively maps the jth symbol of the ith string to an integer between 1 and l. We construct an LSN P system $N = (\Pi, \Sigma, F, l)$ accepting L with one neuron as follows:

$\Pi = (\{a\}, \sigma_1, syn)$ with:

$$\begin{aligned} &\sigma_1 = (a^{l+1}, \{b_{i1} : a^{l+1}/a^{l-f(b_{il_i})+1} \to a \mid 1 \leq i < k\} \\ &\cup \{b_{k1} : a^{l+1}/a \to a\} \\ &\cup \{b_{ij} : a^m/a \to a \mid 1 \leq i \leq k, 2 \leq j \leq l_i, f(b_{i1}) + l_i - j + 1 = m\}), \end{aligned}$$

$F = \{(f(b_{i1}) \mid 1 \leq i \leq k\}$. □

The system starts in the initial configuration a^{l+1}. Suppose that the ith string, say $w_i = b_{i1} b_{i2} \cdots b_{ip} \in L$, is given as input. On seeing the first symbol b_{i1}, the system uses a rule of the type $a^{l+1}/a^{l-f(b_{ip})+1} \to \lambda$ labeled with b_{i1} removing $f(b_{ip})$ spikes from the system. If the last string is given as input, then the rule $a^{l+1}/a \to \lambda$ labeled with b_{k1} is used leaving a^l spikes in the system. The system cannot use any rule of these type from the next step onwards. From the next step onwards, the system uses the other type of rules, whose labels match the next symbols in the input string. When the system processes the string completely, it reaches a final configuration $a^{f(b_{i1})}$. No further rule can be applied and hence the system halts and accepts the input string.

At the price of using arbitrarily many neurons, all regular languages can be accepted.

Theorem 2. $REG \subset LLSNP_*$.

Proof. Let $L \in REG$. Let $D = (Q, \Sigma, \delta, i_0, F_D)$ be a deterministic finite automaton accepting L. Let p be the number of states in D. Rename the states as $q_i, 1 \leq i \leq p$, such that $q_p = i_0$ and redefine the transition rules using the renamed states. Using the modified D we construct an LSN P system $N = (\Pi, \Sigma, F, l)$ as follows:

$\Pi = (\{a\}, \sigma_1, \sigma_2, \cdots, \sigma_p, \sigma_{p+1}, syn)$ with:

$\sigma_i = (2, R_i), 1 \leq i \leq p$, where
$R_i = \{\lambda : a^2/a \to a\}$,

$\sigma_{p+1} = (3p, R_{p+1})$, where
$R_{p+1} = \{b : a^{2p+i}/a^{p+i-j} \rightarrow a \mid (q_i, b, q_j) \in \delta\}$,
$syn = \{(i, p+1), (p+1, i) \mid 1 \leq i \leq p\}$.

F is defined as follows: for every $q_j \in F_D$ add $(2, 2, \cdots, 2, 2p+j)$ to F. □

The SN P system constructed above is shown in Figure 2.

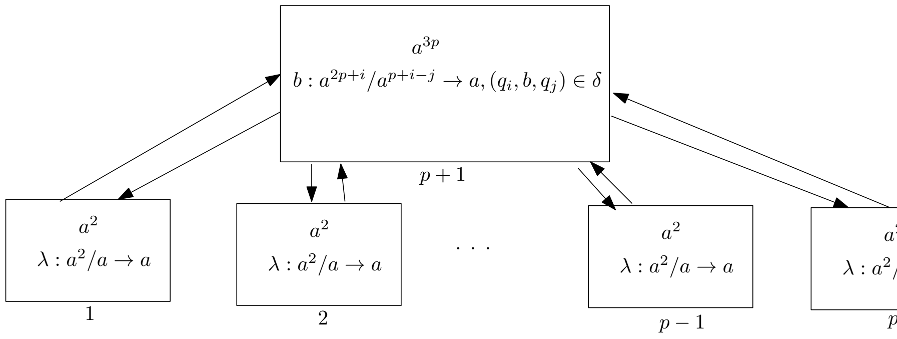

a^{3p}
$b : a^{2p+i}/a^{p+i-j} \rightarrow a, (q_i, b, q_j) \in \delta$
$p+1$
a^2
$\lambda : a^2/a \rightarrow a$
1
2
$\cdots$
$p-1$
p

Fig. 2. DFA to LSN P system construction.

The LSN P system N works as follows: It starts in the initial configuration $(2, 2, \cdots, 2, 3p.)$

The neuron σ_{p+1} fires in the first step by a rule $a^{2p-j} \rightarrow a$ labeled by the first symbol in the input string, associated with the transition rule (q_v, b, q_j) in δ and receives p spikes from its neighboring p neurons. During this step, the first symbol in the input string is consumed and the input pointer advances to the next symbol. The neurons σ_1 to σ_p are meant to continuously load the neuron σ_{p+1} with p spikes. Assume that in some step t, the rule $a^{2p+i}/a^{p+i-j} \rightarrow a$ labeled with the t-th symbol in the input string for $(q_i, b, q_j) \in \delta$ is used for some $1 \leq i \leq p$ and p spikes are received from other neurons. Then $p + i - j$ spikes are consumed and $p + j$ spikes remain in the neuron σ_{p+1}. Then in the step $t + 1$, we have $2p + j$ spikes in neuron σ_{p+1} and a transition rule (q_j, b, q_l) can be used if the next input symbol is b. In this step, the neuron σ_{p+1} receives p spikes from its neighboring neurons and the input symbol that corresponds to the label of the used rule will be consumed and the input pointer advances. In this way the computation proceeds until the input string is processed completely. No further rules can be applied and the computation halts. When the computation halts, the neuron σ_{p+1} contain $2p + k, 1 \leq k \leq p$, spikes and all other neurons contains two spikes each. If $k = i, 1 \leq i \leq p$, for some $q_i \in F$ the string will be accepted. Therefore, $L(N) = L(D)$.

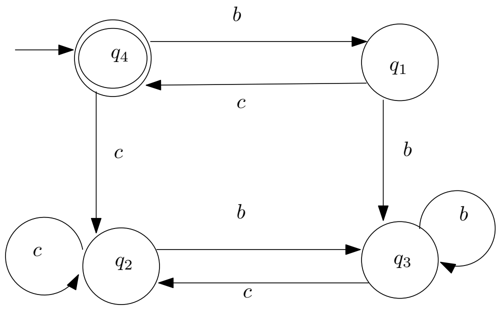
We illustrate the above construction with an example.

Example 2. Consider a DFA $D_1 = (\{q_1, q_2, q_3, q_4\}, \{b, c\}, \delta, q_4, \{q_4\})$, where $\delta =$ $= \{(q_4, c, q_2), (q_4, b, q_1), (q_2, c, q_2), (q_2, b, q_3), (q_3, b, q_3), (q_3, c, q_2), (q_1, b, q_3), (q_1, c, q_4)\}$ which accepts the set of all strings over $\{b, c\}$ in the form $(bc)^n, n \geq 0$. We construct an LSN P system $N = (\Pi_4, \{b, c\}, \{(2, 2, 2, 2, 12), l)$ accepting $L(D_1)$ as follows:

$\Pi_4 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, syn)$, with

$$
\begin{aligned}
&\sigma_1 = (2, \{\lambda : a^2/a \to a\}),\\
&\sigma_2 = (2, \{\lambda : a^2/a \to a\}),\\
&\sigma_3 = (2, \{\lambda : a^2/a \to a\}),\\
&\sigma_4 = (2, \{\lambda : a^2/a \to a\}),\\
&\sigma_5 = (12, \{c : a^{12}/a^6 \to a,\ b : a^{12}/a^7 \to a,\\
&b : a^9/a^2 \to a,\ c : a^9/a \to a, c : a^{10}/a^4 \to a,\ b : a^{10}/a^3 \to a,\\
&b : a^{11}/a^4 \to a,\ c : a^{11}/a^5 \to a\}),\\
&syn = \{(1,5),(5,1),(2,5),(5,2),(3,5),(5,3),(4,5),(5,4)\}.
\end{aligned}
$$

The SN P system Π_4 constructed starting from the DFA D_1 is shown in Figure 3.



DFA D_1 accepting strings over $\{b, c\}^*$ of the form $(bc)^n, n \geq 0$.

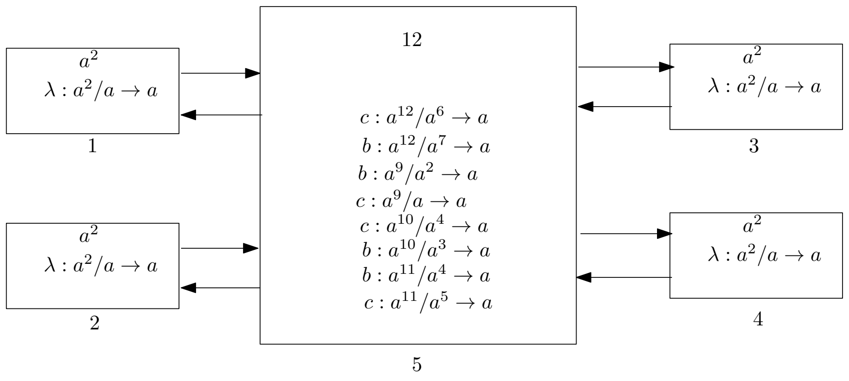


Fig. 3. The LSN P system constructed starting from the DFA D_1.

The LSN P system N works as follows: The system starts in the initial configuration $(2, 2, 2, 2, 12)$.

The SN P system Π_4 accepts the string $bcbc \in L(D_1)$ as follows: Initially, σ_5 contains 12 spikes and neurons σ_1 through σ_4 contains two spikes. The first b is processed by the system using the b labeled rule $b : a^{12}/a^7 \to a$ in σ_5 and λ labeled rules in other neurons. Using those rules one spike is sent to σ_1 through σ_4 from σ_5 and σ_5 also receives a total of four spikes from its neighbors. After this step σ_5 contains nine spikes. The system reaches the configuration $(2, 2, 2, 2, 9)$. In the next step, the first c is processed by using the c labeled rule $c : a^9/a \to a$ in σ_5 and λ labeled rules in other neurons. Using those rules one spike is sent to σ_1 through σ_4 from σ_5 and σ_5 also receives a total of four spikes from its neighbors. After this step σ_5 contains twelve spikes. The system reaches the configuration $(2, 2, 2, 2, 12)$. The next b and c are processed exactly in the same manner in which the system processed the first b and c. After processing the whole string, the system reaches the final configuration $(2, 2, 2, 2, 12)$. Since the input string is processed completely and since the system is in the final configuration the string is accepted by the system.

If the input string is *bbc*, we can easily see that after processing the string completely, the system reaches a configuration $(2, 2, 2, 2, 10)$, which is not a final one. So the string *bbc* is not accepted by the system.

From the construction of Theorem 2, we can see that all the neurons are bounded.

The number of neurons used in this theorem is not bounded. A bound can be obtained if we use extended rules. Specifically, four neurons are enough; to prove this, we use a construction from [15].

Theorem 3. $REG \subset L^eLSNP_4$.

Proof. Let $L \in REG$. Let $D = (Q, \Sigma, \delta, i_0, F_D)$ be a deterministic finite automaton accepting L. Let p be the number of states in D. Rename the states as $q_i, 1 \le i \le p$, such that $q_p = i_0$ and redefine the transition rules using the renamed states. Using the modified D we construct an LSN P system $N = (\Pi, \Sigma, F, l)$ with four neurons as follows:

$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, syn)$ with:

$$\begin{aligned}
&\sigma_1 = (a^{3p}, \{b : a^{2p+i} \to a^j \mid (q_i, b, q_j) \in \delta\}),\\
&\sigma_2 = (0, \{b : a^{2p+i} \to a^j \mid (q_i, b, q_j) \in \delta\}),\\
&\sigma_3 = (a^{3p}, \{\lambda : a^{2p+i} \to a^{2p}, 1 \le i \le p\}),\\
&\sigma_4 = (0, \{\lambda : a^{2p+i} \to a^{2p}, 1 \le i \le p\}),\\
&syn = \{(1, 2), (1, 4), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (4, 3)\}.
\end{aligned}$$

F is defined as follows: for every $q_j \in F_D$ add $(2p + j, 0, 2p + j, 0), (0, 2p + j, 0, 2p + j)$ to F. □

The above SN P system is shown in Figure 4.

The task of checking that $L(D) = L(N)$ is left to the reader.

We illustrate the construction in Theorem 3 by constructing an LSN P system accepting the language $\{(bc)^n \mid n \ge 0\}$ from Example 2; the system is shown in Figure 5, and it works as follows: The system starts in the initial configuration $(12, 0, 12, 0)$. The final configuration is $(12, 0, 12, 0)$. Non-empty strings are accepted by firing the rule labeled with the current input symbol in σ_1, σ_3 and σ_2, σ_4 in alternate steps of the computation; when the string is processed completely, the the system reaches the final configuration $(12, 0, 12, 0)$.

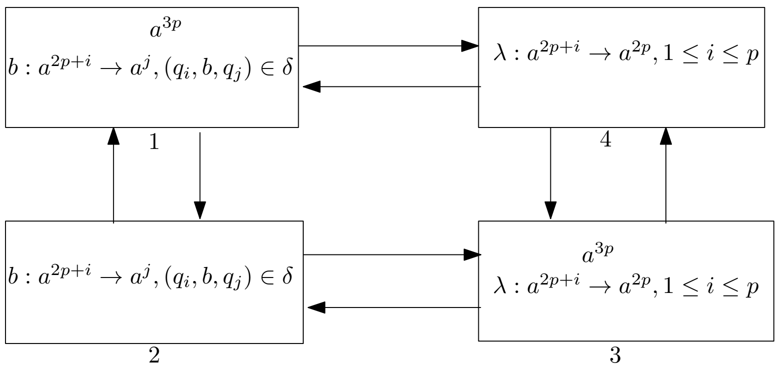

a^{3p}
$b : a^{2p+i} \to a^j, (q_i, b, q_j) \in \delta$
1
$\lambda : a^{2p+i} \to a^{2p}, 1 \le i \le p$
4
$b : a^{2p+i} \to a^j, (q_i, b, q_j) \in \delta$
2
a^{3p}
$\lambda : a^{2p+i} \to a^{2p}, 1 \le i \le p$
3

Fig. 4. DFA to LSN P system construction (Theorem 3).

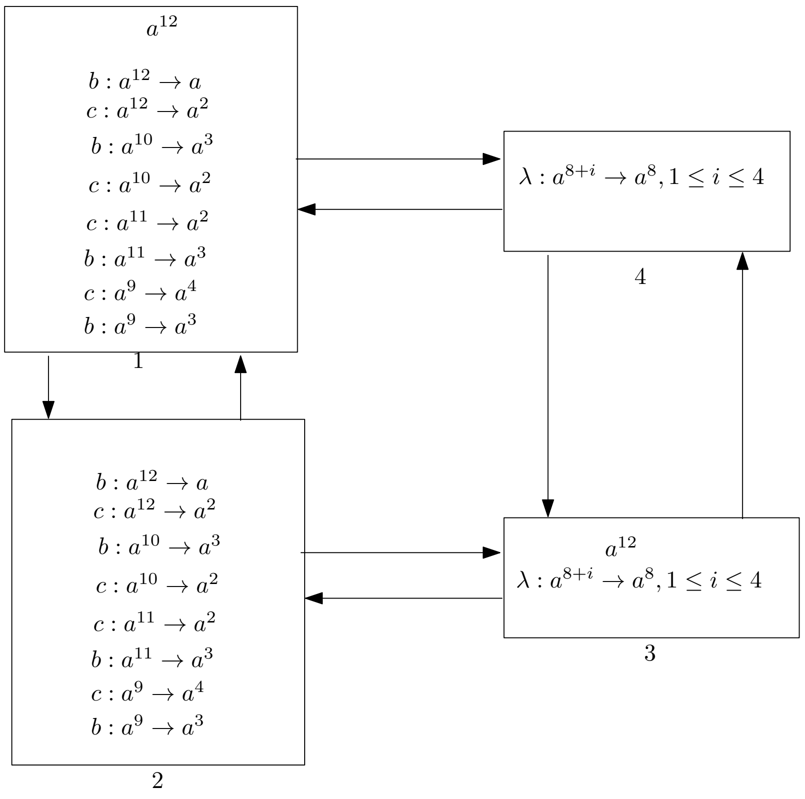


Fig. 5. The LSN P system constructed form the DFA D_1 in Example 2 corresponding to Theorem 3.

Theorem 4. $(CF - REG) \cap LLSNP_* \neq \emptyset$.

Proof. In Example 1, we have shown that the non-regular context-free language $\{b^n c^n \mid n \geq 1\} \cup \{c\}$ is in $LLSNP_*$. □

Theorem 5. $CF - LLSNP_* \neq \emptyset$.

Proof. Consider the context-free language $L = \{ww^R \mid w \in \{b, c\}^*\}$. ($x^R$ is the reversal/mirror image of the string x.) Assume that there exists an LSN P system N with m neurons and n rules that accepts L. Consider a string $uu^R \in L$ and let l be the length of u. After reading l symbols of u, N must be able to reach as many different configurations as there are strings of length l. This must hold since N has to remember the first half of the string uu^R in order to compare it with the second half. Since the alphabet size is two (the argument is applicable to any finite alphabet of cardinality greater than 1), N has to reach at least 2^l different configurations after reading l symbols. If N cannot reach that many configurations, there are two different strings u and u', that leads N to the same configuration. Then, if N accepts uu^R, it will accept also $u'u^R$, which is not in L, a contradiction. So it is required to prove that for sufficiently large l, only less than 2^l configurations are reachable. The proof is as follows: Firing of each rule d_j modifies the configuration by adding a vector $\mathbf{v_j} \in \{0, 1, -r\}^m$ since the rules are of the form $a^r \to a$ or $a^r \to \lambda$. Suppose that the rule $d_i, 1 \leq i \leq n$, is fired $k_i, 1 \leq i \leq n$, times during the whole firing sequence. The configuration of the labeled spiking neural P system gets modified to the configuration $c_0 + \sum_{i=1}^{n} k_i.\mathbf{v_i}$ where $\sum_i k_i = l$. Hence with n rules we can reach at most as many configurations as there are such tuples $(k_1, k_2, \cdots, k_n)$. These n numbers add exactly up to l and therefore $0 \leq k_i \leq l$ for all $i \in \{1, 2, \cdots, n\}$. So there are at most $(l+1)^n$ such tuples. Therefore, for sufficiently large l there are less than 2^l different configurations that are reachable by a spiking neural P system that generates L. This concludes the proof. □

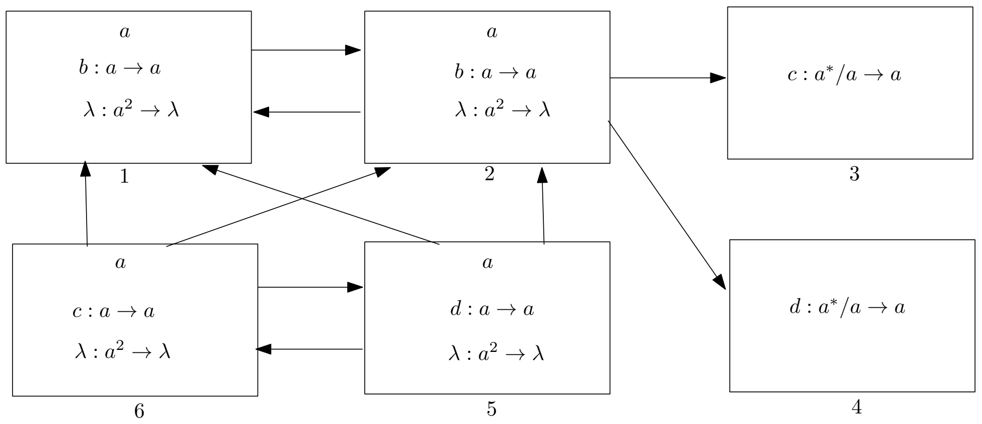
Theorem 6. $LLSNP_* - CF \neq \emptyset$.

Proof. Consider the context-sensitive language $L_2 = \{b^n w \mid n \geq 1, w \in \{c, d\}^*, n_c(w) = n_d(w) = n\}$. ($n_c(w)$ denotes the number of c's in the string w.) We define an LSN P system $N_2 = (\Pi_1, \{b, c, d\}, (0, 0, 0, 0, 0, 0), l)$ accepting L_2 as follows:

$\Pi_1 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, syn)$ with,

$$\begin{aligned}
&\sigma_1 = (1, \{b : a \to a,\ \lambda : a^2 \to \lambda\}),\\
&\sigma_2 = (1, \{b : a \to a,\ \lambda : a^2 \to \lambda\}),\\
&\sigma_3 = (0, \{c : a^*/a \to a\}),\\
&\sigma_4 = (0, \{d : a^*/a \to a\}),\\
&\sigma_5 = (1, \{d : a \to a,\ \lambda : a^2 \to \lambda\}),\\
&\sigma_6 = (1, \{c : a \to a,\ \lambda : a^2 \to \lambda\}),\\
&syn = \{(1,2), (2,1), (2,3), (2,4), (5,1), (5,2), (5,6), (6,1), (6,2), (6,5)\}.
\end{aligned}$$

We can see that $L(N_2) = L_2$ a context-sensitive language which is not context-free. The LSN P system N_2 is diagrammatically shown in Figure 6. The fact that it accepts the mentioned language is left to the reader. □



a
$b : a \to a$
$\lambda : a^2 \to \lambda$
1
a
$b : a \to a$
$\lambda : a^2 \to \lambda$
2
$c : a^*/a \to a$
3
a
$c : a \to a$
$\lambda : a^2 \to \lambda$
6
a
$d : a \to a$
$\lambda : a^2 \to \lambda$
5
$d : a^*/a \to a$
4

Fig. 6. An LSN P system accepting $\{b^n w \mid n \geq 1, w \in \{c, d\}^*, n_c(w) = n_d(w) = n\}$.

Theorem 7. *$LLSNP_* \subset CS$.*

Proof. We show how to recognize a string accepted by an LSN P system with a linear bounded automata. In order to do this, we simulate the computation of the LSN P system by remembering the number of spikes in each neuron after the processing of each symbol in the input string and show that the total number of spikes in the system is linear with respect to the length of the control word.

Consider a language L accepted by an LSN P system N. Let $w = b_1 b_2 \cdots b_k, k \geq 0$ be a string in L. Let the number of neurons be m and the number of rules be n. We build a multi track non-deterministic LBA B which simulates N. In order for B to simulate N, it has to keep track of the number of spikes in each neuron after reading each input symbol. So B's input tape has $m + 1$ tracks. Track 1 holds the input. Tracks 2 to m + 1 hold number of spikes in the neurons. Since B is an LBA, we need to show that the number of spikes in each neuron is bounded linearly with respect to the length of the input string. Consider a neuron, say σ_i and let n_i be the number of spikes initially contained in it. Let $S = d_{1_1} d_{1_2} \cdots d_{1_{i_1}} d_{2_1} d_{2_2} \cdots d_{2_{i_2}} \cdots d_{k_1} d_{k_2} \cdots d_{k_{i_k}}$ be a label sequence corresponding to w. $d_{i_1} d_{i_2} \cdots d_{i_j}$ is the set of rules applied in the ith step. The labels of these rules are either b_i or λ and at least one of the labels is b_i. Thus in the label sequence S, the sub-sequence $d_{j_1} d_{j_2} \cdots d_{j_{i_j}}, 1 \leq j \leq k$, corresponds to the symbol b_j of the input string. Consider a sub-sequence in S having the maximum length and let the length be l. We can see that l is always less than or equal to m, because in a particular step of computation, only one rule in every neuron can be applied. So in a step of computation, the total number of spikes in the neurons σ_i can be at most $n_i + (m-1)$. We can also see that the number of steps in the computation has the same length as the input string. So the total number of spikes in the system after the computation is less than or equal to $\sum_{i=1}^{m} n_i + k \times (m-1)$, which is linear in the length of the input string. So the accepted language is context-sensitive. $\square$

Theorem 8. *$L_\lambda LSNP_* = RE$.*

Proof. We follow the same idea as in the proof of Theorem 9 form [11].

The inclusion $L_\lambda LSNP_* \subset RE$ follows from Church-Turing hypothesis.

The proof of the inclusion $RE \subset L_\lambda LSNP_*$ is as follows:

Let $L \subseteq \Sigma^*$ be a recursively enumerable language. Let $\Sigma = \{b_1, b_2, \cdots, b_l\}$. Define an encoding $e : \Sigma \mapsto \{1, 2, \cdots, l\}$ such that $e(b_i) = i$. We extend the encoding for a string $w = c_1 c_2 \cdots c_k$ as follows: $e(w) = c_1 * (l+1)^{(k-1)} + \cdots + c_{(k-1)} * (l+1)^1 + c_k * (l+1)^0$. We use $l+1$ as the base in order to avoid the digit 0 at the left end of the string.

For any L, there exists a deterministic register machine M which halts after processing the input i_0 placed in its input register if and only if $i_0 = e(w)$ for some $w \in L$.

We use a special symbol \$ which is not an element of the input alphabet to mark the end of the input string. So when the computation starts, we place the string $w\$$ delimited by the special symbol in the environment.

We construct an SN P system Π performing the following operations (σ_{c_0} and σ_{c_1} are two distinguished neurons of Π, where initially σ_{c_0} and σ_{c_1} contains 0 spike). In the register machine, $2n$ spikes contained in a register neuron represent the number n, so $2e(w)$ spikes are going to be generated by M_0 and loaded into M.

1. For some $1 \leq i \leq l$, processing an input symbol $b_i \in \Sigma$ introduces the number i in σ_{c_0}. A number i is represented in a neuron by storing $2i$ spikes. This task is done by introducing $2i$ spikes in σ_{c_0}.

2. Multiply the number stored in σ_{c_1} by $l+1$, then add the number from σ_{c_0}. So, if σ_{c_0} contains $2i$ spikes and σ_{c_1} contains $2j, j \geq 0$ spikes, we end up with $2(j(l+1)+i)$ spikes in σ_{c_1} and no spike in σ_{c_0}. This task is done by simulating a register machine M_0 which does the multiplication by $l+1$ to the content of σ_{c_1} and adding a number between 1 and l.

3. Repeat from step 1, until the input string is processed completely and when finished go to the next step.

4. σ_{c_1} contains a number of spikes equal to $e(w)$ for the input string $w \in \Sigma^+$. We now start to simulate the working of the register machine M in recognizing the number $e(w)$. If the machine halts, then $w \in L$, otherwise the machine goes into an infinite loop.

We are not giving the detailed construction of the SN P system Π. We relay on the fact that a register machine can be simulated by an SN P system as shown in [1]. The overall appearance of Π is given in Figure 7, 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 $M = (m, H, l_0, l_h, I)$ and $H_0 \cap H = \emptyset$.

The final configuration of the system is a row vector with size equal to the total number of neurons in the system *i.e.*, $(11 + m_0 + m - 1$ (since σ_{c_1} is shared) $+3 *$ number of ADD instructions $+ 3 *$ number of SUB instructions for machine $M_0 + 5 *$ number of SUB instructions for machine $M)$ whose components are zero.

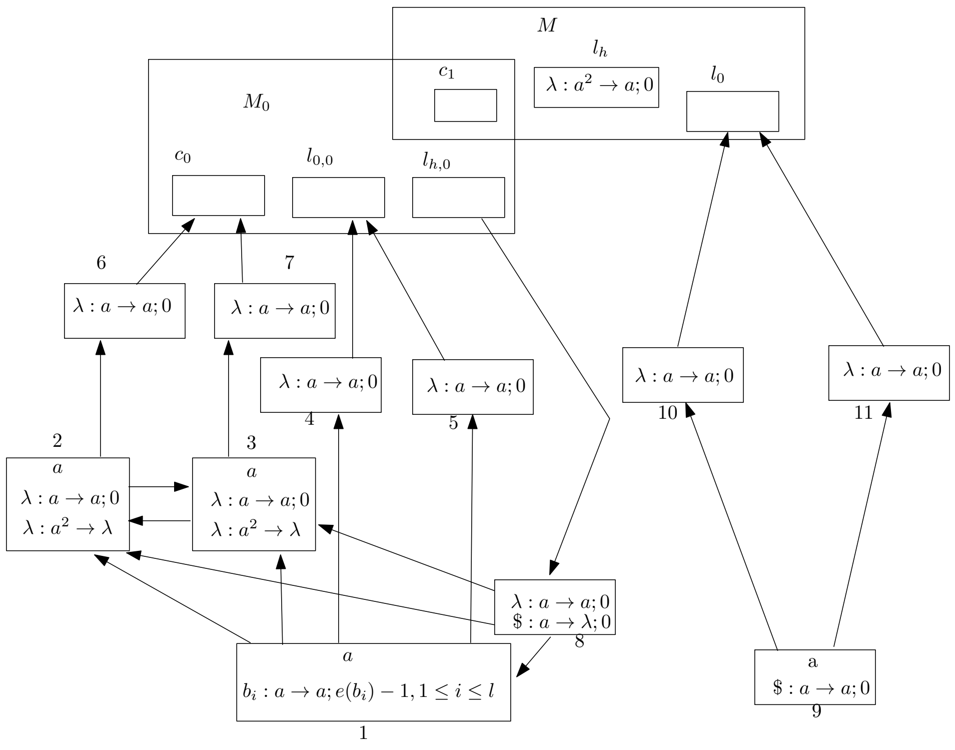


Fig. 7. The structure of the SN P system from the proof of Theorem 8.

We start with one spike in $\sigma_1, \sigma_2, \sigma_3$ and two spikes in σ_{c_1}. So in the first step σ_1, σ_2 and σ_3 spike. As long as σ_2 and σ_3 do not receive a spike from σ_1, they spike and send a spike to each other and two spikes to σ_{c_0}. Using a rule $b_i : a \to a; e(b_i) - 1, 1 \leq i \leq l$ in σ_1, consumes the input symbol b_i pointed by the input pointer and the input pointer advances to the next symbol in the input string and causes a spike to be transferred from σ_1 to $\sigma_2, \sigma_3, \sigma_4$ and σ_5 after a delay of $e(b_i) - 1$ and this causes σ_2 and σ_3 to stop working and σ_4 and σ_5 to load two spikes to $\sigma_{l_{0,0}}$ thereby starting the simulation of the register machine M_0. The subsystem corresponding to the register machine M_0 starts to work, multiplying the value contained in σ_{c_1} with $l+1$ and adding i. When this process halts, $\sigma_{l_{h,0}}$ is activated (this neuron will get two spikes in the end of computation and will spike), and one spike is sent to σ_8. In σ_8, the system uses the first rule if it do not see the end-marker there by sending one spike to each of σ_1, σ_2 and σ_3, so the processing of the input string can be resumed. When the system sees the input end-marker i.e. the input string is processed completely, it uses the second rule in σ_8 to get rid of the spike in σ_8 and at the same time the system uses the rule in σ_9 sending one spike to σ_{10} and σ_{11}, and they activate σ_{l_0} by sending two spikes to it, thus starting the simulation of the register machine M. If the register machine M halts on the input then the neuron σ_{l_h}, that correspond to the label of the halt instruction receives two spikes at the end of the computation. In the next step, the system uses the λ labeled rule $\lambda : a^2 \to a; 0$ in σ_{l_h}, getting rid of the spikes in σ_{l_h}. All the neurons in the system do not have any spike, and since the system is in the final configuration, the string w is accepted by the system.

For the construction of modules M and M_0 we refer to [11]; all rules are labeled with λ. □

6. Conclusion

In this paper we investigated the control words associated with computations of spiking neural P systems. An accepting style was adopted: each step of a computation "consumes" a symbol of an "input" string and the string is accepted when it is completely read and the system reaches a final state. To this aim, labels are associated with the rules of an SN P system, which are symbols of a given alphabet or they are empty. In each transition, only rules with the same label or with the empty label are used. Also λ moves are possible, when no symbol of the input string is read and all used rules have the empty label.

The families of languages accepted in this way are compared with the families of the Chomsky hierarchy. In the case when λ moves are allowed, all recursively enumerable languages can be obtained in this way, in the opposite case all accepted languages are context-sensitive, all regular languages are covered, not all context-free languages, but there are non-context-free context-sensitive languages which can be obtained as control languages of SN P systems.

Several research issues remain to be considered. First, we mention the question whether or not the use of final configurations in defining successful computations makes any difference, or the usual halting condition is sufficient. Then, a systematic comparison of families of languages accepted by various classes of SN P systems should be done (remember that we have standard SN P systems, extended, with or without delay, with or without forgetting rules, synchronous and asynchronous and so on).

A technical question is whether the result in Theorem 8 can be obtained also without using the delay feature (note that in all other theorems, the delay plays no role). We feel that the delay can be avoided.

Finally, we mention the natural issue of extending this way of associating a language with a P system to other classes of P systems, different from SN P systems.

References

[1] Ionescu M., Păun Gh., Yokomori T., *Spiking neural P systems*, Fundamenta Informaticae, **71**, 2–3, pp. 279–308, 2006.

[2] Păun Gh., *Computing with membranes*, Journal of Computer and System Science, **61**(1), pp. 108–143, 2000.

[3] Păun Gh., *Membrane Computing – An Introduction*, Springer-Verlag, Berlin, 2002.

[4] Păun Gh., Rozenberg G., Salomaa A., eds., *The Oxford Handbook of Membrane Computing*, Oxford Univ. Press, 2010.

[5] Pan L., Păun Gh., *Spiking neural P systems with anti-spikes*, Int. J. of Computers, Communications and Control, **4**, pp. 273–282, 2009.

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

[7] Ibarra O. H., Woodworth S., *Characterizing regular languages by spiking neural P systems*, Int. J. of Foundations of Computer Science, **18**(6), pp. 1247–1256, 2007.

[8] Krithivasan K., Metta V. P., Garg D., On string languages generated by spiking neural P systems with anti-spikes, *Int. J. of Foundations of Computer Science*, **22**(1), pp. 15–27, 2011.

[9] Păun Gh., Pérez-Jiménez M. J., Rozenberg G., *Spike trains in spiking neural P systems*, Int. J. of Foundations of Computer Science, **17**(4), pp. 975–1002, 2006.

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

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

[12] Minsky M., *Computation – Finite and infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.

[13] Krithivasan K., Păun Gh., Ramanujan A., *Control words associated with P systems*, *Frontiers of Membrane Computing: Open Problems and Research Topics*, by Gheorghe M., Păun Gh., Pérez-Jiménez M. J. (Eds.), published in the second volume of the *Proceedings of 10th Brainstorming Week on Membrane Computing*, Sevilla, 171–250, 2012.

[14] Ramanujan A., Krithivasan K., *Control Languages of spiking neural P systems*, Submitted.

[15] Păun Gh., Pérez-Jiménez M. J., *Languages and P systems: recent developments*, Manuscript.

[16] Krithivasan K., Ramanujan A., *Matrix Representation of Spiking Neural P Systems with Delay*, *Proceedings of the Twelfth international conference on membrane computing*, France, pp. 283–298, 2011.

[17] The P System Web Page: `http://ppage.psystems.eu`