On Trace Languages Generated by (Small) Spiking Neural P Systems

Haiming Chen[1], **Mihai Ionescu**[2], **Andrei Păun**[3], **Gheorghe Păun**[4], **Bianca Popa**[3]

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

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

[3]Department of Computer Science, Louisiana Tech University
Ruston, PO Box 10348, Louisiana, LA-71272 USA
`apaun@latech.edu, bdp010@latech.edu`

[4]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania, and
Department of Computer Science and AI, Univ. of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
`george.paun@imar.ro, gpaun@us.es`

Abstract

We extend to spiking neural P systems a notion investigated in the "standard" membrane systems: the language of the traces of a distinguished object. In our case, we distinguish a spike by "marking" it and we follow its path through the neurons of the system, thus obtaining a language. Several examples are discussed and some preliminary results about this way of associating a language with a spiking neural P system are given, together with a series of topics for further research. For instance, we show that each regular language is the morphic image of a trace language intersected with a very particular regular language, while each recursively enumerable language over the one-letter alphabet is the projection of a trace language.

In all proofs we try to keep the size of used systems (number of neurons, of rules in each neuron, of spikes consumed or removed by rules) as small as possible.

1 Introduction

We continue here the study of the spiking neural P systems (in short, SN P systems) recently introduced in [4], by considering in this framework the idea of following the paths of an object through the system in order to obtain a language. For usual P systems (with symport/antiport rules), this way of generating a language was introduced in [3] and then investigated in a series of papers (see Section 4.4 of [6]).

In short, an SN P 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. One also uses forgetting rules, which remove spikes from neurons. Therefore, the spikes are moved, created, destroyed, but never modified (there is only one type of objects in the system). This makes possible and natural to distinguish one of the spikes present already in the initial configuration of the system and to follow its path through the neurons, recording the labels of the neurons and thus obtaining a string. More precisely, we assume that one of the spikes has a "flag", which remains with this spike as long as the spike is not consumed or forgotten, but the flag passes to the produced spike when the old holder is consumed; the flag disappears when the spike is forgotten or sent out of the system. More precise definitions will be given in Section 2. We only add here the fact that we consider only halting computations, so that the labels of neurons visited by the flag form a string; due to the non-determinism in the SN P system functioning, we get in this way a set of strings, hence a language, associated with the system.

Because the SN P systems are Turing complete as number computing devices (see [4] and [7]) and also complete modulo direct and inverse morphisms in the case when they are used as language generators (see [1]), it is expected that the SN P systems are powerful also as trace languages generators. This expectation is somewhat confirmed below: we both produce a series of examples of systems with an intricate functioning, we give a representation of regular languages (over any alphabet), and we also show that unary recursively enumerable languages have simple representations in terms of SN P systems trace languages. On the other hand, because of the specific way the trace language is generated, there are finite (even singleton) languages which cannot be obtained in this way. However, due to the preliminary stage of our research, we do not have yet a precise estimation of the size of the generated families of languages; many problems in this respect are formulated along the paper.

In all these investigations we try to keep the used system as simple as possible, with respect to the many descriptional complexity measures which can be considered for SN P systems: number of neurons, number of rules in each neuron, etc.

2 Trace Languages Associated with SN P Systems

We assume the reader to be familiar with basic language and automata theory, e.g., from [8], so that we introduce here only some notations used later in the paper.

If $L_1, L_2 \subseteq V^*$ are two languages, the left and right quotients of L_1 with respect to L_2 are defined by $L_2 \backslash L_1 = \{w \in V^* \mid xw \in L_1 \text{ for some } x \in L_2\}$, and respectively $L_1/L_2 = \{w \in V^* \mid wx \in L_1 \text{ for some } x \in L_2\}$. When L_2 is a singleton, $L_2 = \{x\}$, we write $\partial^l_x(L_1)$ for $\{x\}\backslash L_1$ and $\partial^r_x(L_1)$ for $L_1/\{x\}$, and we call these operations left and right derivatives of L_1 with respect to x. We denote by FIN, REG, RE the families of finite, regular, and recursively enumerable languages.

Convention: the empty string is ignored when examining the generative power of any device considered in this paper.

We introduce now the SN P systems in their basic form, then we define the new way of using them, as (trace) language generators; for further details we refer to [4], [7].

A *spiking neural P system* (abbreviated as SN P system), of degree $m \geq 1$, is a construct of the form $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn)$, 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. $syn \subseteq \{1, 2, \ldots, m\} \times \{0, 1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons), with 0 indicating the environment of the system.

Note that, in contrast to [4], we have not indicated here an output neuron, but we have allowed synapses $(i, 0)$ for any neuron σ_i.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains exactly k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a; d$ 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). A spike emitted by a neuron σ_i (is replicated and a copy of it) goes to each neuron σ_j such that $(i, j) \in syn$, as well as to the environment, if $(i, 0) \in syn$. (When a neuron σ_i emits a spike and there is no neuron to receive it, then a synapse $(i, 0)$ is used as an explicit way to point out that the spike is "lost" in the environment.)

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 $a^s \rightarrow \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \rightarrow a; d$ of type (1) has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a; d$.

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, $E_1/a^{c_1} \rightarrow a; d_1$ and $E_2/a^{c_2} \rightarrow a; d_2$, 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 from now on 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.

In the spirit of spiking neurons, see, e.g., [5], as the result of a computation, in [4] and [7] one considers the distance between two consecutive spikes which exit a distinguished neuron of the system. Then, in [1] one considers as the result of a computation the so-called spike train of the computation, the sequence of symbols 0 and 1 obtained by associating 1 with a step when a spike exits the system and 0 otherwise. Languages over the binary alphabet are computed in this way.

Here we consider yet another idea for defining a language, taking into account the traces of a distinguished spike through the system.

Specifically, we distinguish one of the neurons of the system as the *input* one (thus, we add a further component, *in*, to the system description, with $in \in \{1, 2, \ldots, m\}$) and in the initial configuration of the system we "mark" one spike from this neuron – the intuition is that this spike has a "flag" – and we follow the path of this flag during the computation, *recording the labels of the neurons where the flag is present in the end of each step*. Actually, for neuron σ_i we consider the symbol b_i in the trace string. (When presenting the initial configuration of the system, the number n_{in}, of spikes present in the input neuron, is written in the form n'_{in}, to indicate that the marked spike is here.)

The previous definition contains many delicate points which need clarifications – and we use a simple example to do this.

Assume that in neuron σ_i we have three spikes, one of them marked; we write aaa' to represent them. Assume also that we have a spiking rule $aaa/aa \to a; 0$. When applied, this rule consumes two spikes, one remains in the neuron and one spike is produced and sent along the synapses going out of neuron σ_i. Two cases already appear: the marked spike is consumed or not. If not consumed, it remains in the neuron. If consumed, then the flag passes to the produced spike. Now, if there are two or more synapses going out of neuron σ_i, then again we can have a branching: only one spike is marked, hence only on one of the synapses (i, j) we will have a marked spike and that synapse is non-deterministically chosen (on other synapses we send non-marked spikes). If σ_j is an open neuron, then the marked spike ends in this neuron. If σ_j is a closed neuron, then the marked spike is lost, and the same happens if $j = 0$ (the marked spike exits in the environment). Anyway, if the marked spike is consumed, at the end of this step it is no longer present in neuron σ_i; it is in neuron σ_j if $(i, j) \in syn$ and neuron σ_j is open, or it is removed from the system in other cases.

Therefore, if in the initial configuration of the system neuron σ_i contains the marked spike, then the trace can start either with b_i (if the marked spike is not consumed) or with b_j (if the marked spike was consumed and passed to neuron σ_j); if the marked spike is consumed and lost, then we generate the empty string, which is ignored in our considerations. Similarly in later steps.

If the rule used is of the form $aaa/aa \to a; d$, for some $d \geq 1$, and the marked spike is consumed, then the newly marked spike remains in neuron σ_i for d steps, hence the trace starts/continues with b_i^d. Similarly, if no rule is used in neuron σ_i for k steps, then the trace records k copies of b_i. If a forgetting rule is used in the neuron where the marked spike is placed, then the trace string stops (and no symbol is recorded for this step).

Therefore, when considering the possible branchings of the computation, we have to take into account the non-determinism not only in using the spiking rules, but also in consuming the marked spike and in sending it along one of the possible synapses.

The previous discussion has, hopefully, made clear what we mean by *recording the labels of the neurons where the flag is present in the end of each step*, and why we have chosen the end of a step and not the beginning: in the latter case, all traces would start with the same symbol, corresponding to the input neuron, which is a strong – and artificial – restriction.

In the next section, we will illustrate all these points by examining in detail an example.

Anyway, we take into account only halting computations: irrespective whether or not a marked spike is still present in the system, the computation should halt (note that it is possible that the marked spike is removed and the computation still continues for a while – but this time without adding further symbols to the trace string).

For an SN P system Π we denote by $T(\Pi)$ the language of all strings describing the traces of the marked spike in all halting computations of Π. Then, we denote by $TSNP_m(rule_k, cons_p, forg_q)$ the family of languages $T(\Pi)$, 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.

3 Two Examples

We consider here two examples, both illuminating the previous definitions and relevant for the intricate work of SN P systems as language generators by means of traces; indications on the power of these devices are also obtained in this way.

We start with a system already having a complex behavior, the one whose initial configuration is given in Figure 1 (we denote it by Π_1). This also gives us the opportunity to introduce the way to graphically represent an SN P system: as a directed graph, with the neurons as nodes and the synapses indicated by arrows; in each neuron we give the rules and the spikes present in the initial configuration, with the marked spike indicated by a prime; synapses of the form $(i, 0)$ are indicated by arrows pointing to the environment.

We have three neurons, labeled with 1, 2, 3, with neuron σ_1 being the input one. Each neuron contains three rules, and only neuron σ_1 has two spiking rules, but the non-determinism of the system is considerable, due to the possible traces of the marked spike.

The evolution of the system Π_1 can be analyzed on a transition diagram as that from Figure 2: because the number of configurations reachable from the initial configuration is finite, we can place them in the nodes of a graph, and between two nodes/configurations

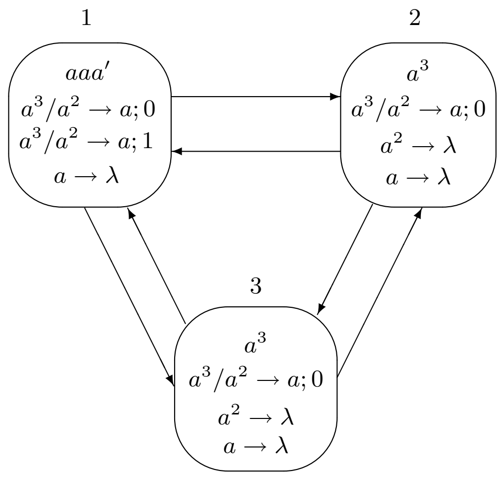

Figure 1: The initial configuration of system Π_1

we draw an arrow if a direct transition is possible between them.

It should be noted in Figure 2 an important detail: when presenting a configuration of the system where there is a marked spike, it is no longer sufficient to indicate only the number of spikes and the open status of neurons, but we also have to indicate the place of the marked spike (if the system still contains such a spike). This is done by priming either the number of spikes from the neuron where the marked spike is, or by priming the number of steps until the neuron is open, in the case when a marked spike was produced by means of a spiking rule with delay.

In Figure 2 we have also indicated on the arrows the symbol introduced in the trace string by that arrow (this is b_j, where σ_j is the neuron where the marked spike arrives in the end of this step). Thus, following the marked arrows, we can construct the language of all traces.

Let us follow on this diagram some of the possible traces. As long as neuron σ_1 uses the rule $a^3/a^2 \to a; 0$, the marked spike circulates among neurons $\sigma_1, \sigma_2, \sigma_3$ and the computation continues. Note that the marked spike can be consumed or not; in the first case it moves to one of the partner neurons, non-deterministically chosen, in the latter case it remains in the neuron where it is placed.

When neuron σ_1 uses the rule $a^3/a^2 \to a; 1$, then the computation passes to the halting phase – in the diagram from Figure 2, we leave the upper level and we pass to the next level of configurations. If the marked spike was in neuron σ_1, it is or not consumed; this is the case when reaching the configurations $\langle 1'/1, 2/0, 2/0\rangle$, $\langle 1/1', 2/0, 2/0\rangle$: all neurons consume two spikes, but neurons σ_2 and σ_3 exchange one spike, hence they end the step with two spikes inside; neuron σ_1 has only one spike inside (it is closed, does not accept spikes from neurons σ_2 and σ_3) and one ready to be emitted, and either one of them can be the marked one.

In each case, after two more steps the computation halts with no spike in the system. Thus, the obtained language is

$$T(\Pi_1) = \{b_1b_1, b_1b_2, b_1b_3\} \cup \{b_1, b_2, b_3\}^*\{b_1b_1b_1, b_1b_1b_2, b_1b_1b_3, b_2b_2, b_2b_3, b_3b_2, b_3b_3\}.$$

We continue with a simpler example, the system Π_2 given in Figure 3. It generates

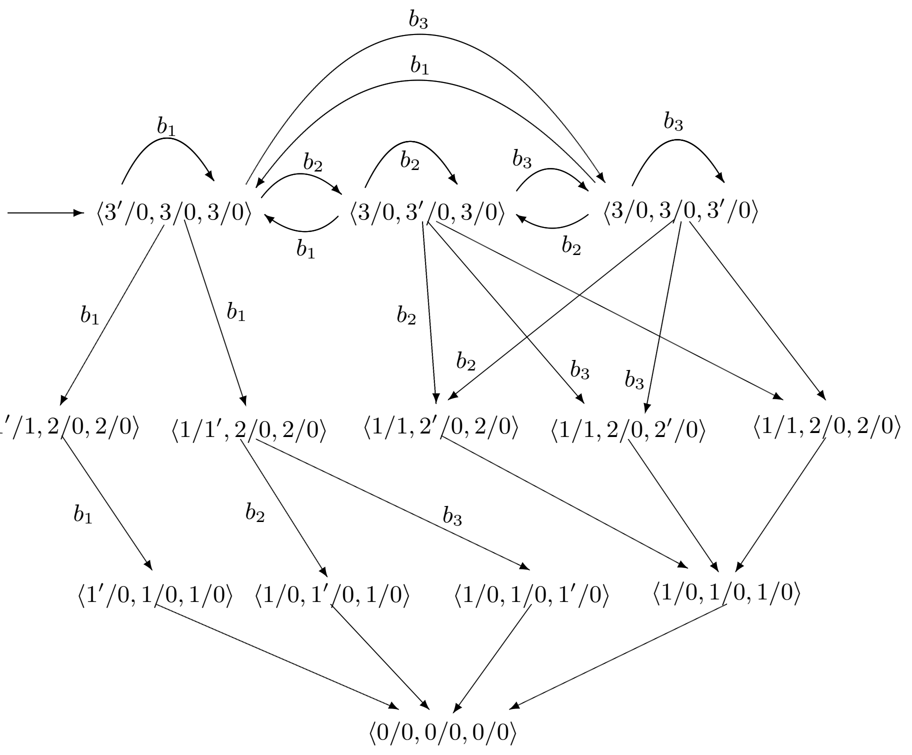
b_3
b_1
b_1
b_3
b_2
b_2
b_3
$\langle 3'/0, 3/0, 3/0 \rangle$
$\langle 3/0, 3'/0, 3/0 \rangle$
$\langle 3/0, 3/0, 3'/0 \rangle$
b_1
b_2
b_1
b_1
b_2
b_2
b_3
b_3
$\langle 1'/1, 2/0, 2/0 \rangle$
$\langle 1/1', 2/0, 2/0 \rangle$
$\langle 1/1, 2'/0, 2/0 \rangle$
$\langle 1/1, 2/0, 2'/0 \rangle$
$\langle 1/1, 2/0, 2/0 \rangle$
b_1
b_2
b_3
$\langle 1'/0, 1/0, 1/0 \rangle$
$\langle 1/0, 1'/0, 1/0 \rangle$
$\langle 1/0, 1/0, 1'/0 \rangle$
$\langle 1/0, 1/0, 1/0 \rangle$
$\langle 0/0, 0/0, 0/0 \rangle$

Figure 2: The transition diagram of system Π_1

the language $T(\Pi_2) = \{b_1^n, b_2^m\}$, for $n, m \geq 1$. In the first step, neuron σ_1 consumes or does not consume the marked spike, thus keeping it inside or sending it to neuron σ_2. One spike remains in neuron σ_1 and one is placed in neuron σ_2. Simultaneously, neurons σ_3 and σ_4 fire, and they spike after $n-1$ and $m-1$ steps, respectively. Thus, in steps n and m, neurons σ_1 and σ_2, respectively, receive one more spike, which is forgotten in the next step together with the spike existing there.

Note that n and m can be equal or different.

4 The Power of SN P Systems as Trace Generators

The following inclusions are direct consequences of the definitions:

Lemma 4.1 *$TSNP_m(rule_k, cons_p, forg_q) \subseteq TSNP_{m'}(rule_{k'}, cons_{p'}, forg_{q'}) \subseteq TSNP_*(rule_*, cons_*, forg_*) \subseteq RE$, for $1 \leq m \leq m'$, $1 \leq k \leq k'$, $1 \leq p \leq p'$, $1 \leq q \leq q'$.*

We pass now to investigating the relationship with the families of languages from Chomsky hierarchy, starting with a counterexample result.

Theorem 4.1 *There are singleton languages not in $TSNP_*(rule_*, cons_*, forg_*)$.*

Proof. Let us consider the language $L = \{b_1 b_2 b_1 b_3\}$. In order to generate it, at least three neurons should be used, with labels 1, 2, 3. Moreover, the synapses $(1,2), (2,1), (1,3)$ are necessary. The existence of the synapses $(1,2), (2,1)$ makes possible also the trace

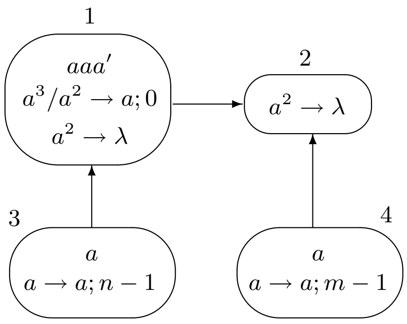

Figure 3: The initial configuration of system Π_2

$b_1b_2b_1b_2$: the marked spike exists after the third step and it can go non-deterministically to each neuron σ_2 and σ_3. Thus, $T(\Pi)$ cannot be a singleton, for an SN P system Π such that $b_1b_2b_1b_3 \in T(\Pi)$. $\square$

Clearly, this reasoning can be applied to any string of the form $w = w_1b_ib_jw_2b_ib_kw_3$ with $j \neq k$, and to any language which contains a string like w but not also the string $w_1b_ib_jw_2b_ib_jw_3$.

Let us mention now a result already proved by the considerations related to the first example from the previous section (this introduces another complexity parameter, of a dynamical nature: the number of spikes present in the neurons during a computation).

Theorem 4.2 *The family of trace languages generated by SN P systems by means of computations with a bounded number of spikes present in their neurons is strictly included in the family of regular languages.*

Proof. The inclusion follows from the fact that the transition diagram associated with the computations of an SN P system which use a bounded number of spikes is finite and can be interpreted as the transition diagram of a finite automaton, as already done in Section 3. The fact that the inclusion is proper is a consequence of Theorem 4.1. $\square$

However, modulo some simple squeezing operations, SN P systems with a bounded number of spikes inside can generate any regular language:

Theorem 4.3 *For each language $L \subseteq V^*, L \in REG$ there is an SN P system Π such that each neuron from any computation of Π contains a bounded number of spikes, and $L = h(\partial_c^r(T(\Pi)))$ for some coding h and symbol c not in V; actually, $T(\Pi) \in TSNP_{m_L}(rule_{k_L}, cons_{p_L}, forg_0)$, for some constants m_L, k_L, p_L depending on language L.*

Proof. Let us assume that $V = \{b_1, b_2, \dots, b_n\}$, and take a regular grammar $G = (N, V, S, P)$ generating the language L. Without loss of the generality, we may assume that each rule $A \to b_iB$ from P has $A \neq B$. If this is not the case with the initial set of rules, then we proceed as follows. Take a rule $A \to b_iA$; we consider a new nonterminal, A', and we replace the rule $A \to b_iA$ with $A \to b_iA'$, then we also add the rule $A' \to b_iA$ as well as all rules $A' \to b_jB$ for each $A \to b_jB \in P$. Let us continue to denote by G the obtained grammar. It is clear that this change does not modify the language generated by G, but it diminishes by one the number of rules having the same nonterminal in both sides. Continuing in this way, we eliminate all rules of this form.

Assume that G contains k nonterminal rules (i.e., rules of the form $A \to b_iB$, with $A, B \in N$; from the previous discussion, we know that $A \neq B$). If $k = 1$, then the grammar has only one nonterminal rule, $S \to b_iB$, for some $B \neq S$, hence $L(G)$ consists only of words of length one or two, and then the statement in the theorem is obvious. Thus, we can assume that $k \geq 2$. We construct the SN P system Π as follows.

The set of neurons is the following:

1. $\sigma_{\langle S\rangle} = (1', \{a \to a; 0\})$,
2. $\sigma_{\langle i,B\rangle} = (0, \{a^j \to a; 0 \mid 1 \leq j \leq k-1\})$, for each rule $A \to b_iB$ of G,
3. $\sigma_{\langle i\rangle} = (0, \{a^j \to a; 0 \mid 1 \leq j \leq k-1\})$, for each rule $A \to b_i$ of G,
4. $\sigma_f = (0, \{a^j \to a; 0 \mid 1 \leq j \leq k-1\})$,
5. $\sigma_{c_i} = (0, \{a \to a; 0\})$, for $i = 1, 2, \ldots, k$,
6. $\sigma_1 = (2, \{a^2/a \to a; 0, \quad a^2/a \to a; 1\})$,
7. $\sigma_2 = (2, \{a^2/a \to a; 0, \quad a \to \lambda\})$,
8. $\sigma_3 = (0, \{a \to a; 0, \quad a^2 \to \lambda\})$.

Among these neurons, we take the following synapses:

$$\begin{aligned} syn &= \{(\langle S\rangle, \langle i,A\rangle) \mid S \to b_iA \in P\} \cup \{(\langle i,A\rangle, \langle j,B\rangle) \mid A \to b_jB \in P, 1 \leq i \leq n\} \\ &\cup \{(\langle i,A\rangle, \langle j\rangle) \mid A \to b_j \in P, 1 \leq i \leq n\} \cup \{(\langle S\rangle, \langle j\rangle) \mid S \to b_j \in P\} \\ &\cup \{(\langle i\rangle, f) \mid 1 \leq i \leq n\} \cup \{(1,2),(2,1),(1,3),(2,3)\} \cup \{(3, c_i) \mid 1 \leq i \leq k\} \\ &\cup \{(c_i, \langle S\rangle),\ (c_i, \langle j,A\rangle),\ (c_i, \langle j\rangle) \mid 1 \leq i \leq k, 1 \leq j \leq n, A \in N\}. \end{aligned}$$

We start with two spikes in each neuron σ_1 and σ_2, and with one spike, marked, placed in neuron $\sigma_{\langle S\rangle}$.

The input neuron spikes in the first step.

The neurons $\sigma_1, \sigma_2, \sigma_3, \sigma_f, \sigma_{c_1}, \ldots, \sigma_{c_k}$ will be discussed separately below.

Among neurons with labels of the form $\langle S\rangle$, $\langle i,A\rangle$, and $\langle j\rangle$, we have synapses only if the labels of these neurons are linked by a rule of the grammar. If there are several rules of the form $A \to b_{i_1}B_1, \ldots, A \to b_{i_r}B_r$, then the spike emitted by each neuron $\sigma_{\langle j,A\rangle}, 1 \leq j \leq n$, goes to all neurons $\sigma_{\langle i_t,B_t\rangle}, 1 \leq t \leq r$, and the marked spike can take any of these choices. Conversely, we can have several (actually, at most $k-1$) synapses coming to the same neuron, if more rules have the same right hand member. Thus, in any neuron we can collect at most $k-1$ spikes in a step. All of them are immediately consumed, hence never the number of spikes from any neuron becomes greater than $k-1$. Clearly, the marked spike can follow a derivation in G.

Because the grammar G can contain cycles (for instance, pairs of rules of the form $A \to b_iB, B \to b_iA$), the system Π can contain pairs of self-sustaining neurons, hence the computation in Π could not stop, even if a derivation in G was correctly simulated (this is due to the fact that the spikes leaving a neuron are replicated to all neurons to which we have a synapse).

In order to prevent such "wrong" evolutions, we use the "halting module", composed of the neurons $\sigma_1, \sigma_2, \sigma_3, \sigma_f, \sigma_{c_1}, \ldots, \sigma_{c_k}$: neurons σ_1, σ_2 exchange spikes for an arbitrary number of steps; when σ_1 uses the rule $a^2/a \to a; 1$, their interplay stops, but neuron σ_3 fires and sends a spike to all neurons $\sigma_{c_i}, 1 \leq i \leq k$. These neurons will send spikes to all neurons $\sigma_{\langle j,A\rangle}$, $\sigma_{\langle j\rangle}$, and $\sigma_{\langle S\rangle}$ of the system, thus halting their evolution (they do not have rules for handling more than $k-1$ spikes). The computation halts.

What is not ensured yet by the construction of Π is the fact that the computation is not halted as above before finishing the derivation in G which we want to simulate, that is, with the marked spike present inside the system. This aspect is handled by the squeezing mechanism: we take the alphabet

$$U \;=\; \{b_{\langle i,A\rangle} \mid 1 \leq i \leq n, A \in N\} \cup \{b_{\langle i\rangle} \mid 1 \leq i \leq n\},$$

the symbol $c = b_f$, and the coding $h : U^* \longrightarrow V^*$ defined by $h(b_{\langle i,A\rangle}) = b_i$ for $1 \leq i \leq n, A \in N$, and $h(b_{\langle i\rangle}) = b_i$ for $1 \leq i \leq n$.

The right derivative with respect to b_f selects from $T(\Pi)$ only those traces for which the marked spike reaches the "final" neuron σ_f, which is accessible only from a "terminal" neuron $\sigma_{\langle i\rangle}$, hence the marked spike has followed a complete derivation in G. Then, the coding h renames the symbols, thus delivering exactly the strings of $L(G)$.

Clearly, we can determine the constants m_L, k_L, p_L as in the statement of the theorem, depending on the size of grammar G, and this observation completes the proof. $\square$

A related result can be obtained by slightly changing the previous proof.

Theorem 4.4 *For each regular language $L \subseteq V^*$ there is an SN P system Π such that each neuron from any computation of Π contains a bounded number of spikes, and $L = h(T(\Pi) \cap U_1^* U_2)$, for some coding h and alphabets U_1, U_2.*

As expected, also non-regular languages can be generated – of course, by using systems whose computations are allowed to use an unbounded number of spikes in the neurons. We skip the proof of the next result.

Theorem 4.5 *$TSNP_{12}(rule_2, cons_2, forg_1) - REG \neq \emptyset$.*

Actually, languages "far from being regular" can be also generated:

Theorem 4.6 *Every unary language $L \in RE$ can be written in the form $L = h(L') = (b_1^* \backslash L') \cap b_2^*$, where $L' \in TSNP_*(rule_2, cons_3, forg_3)$, and h is a projection.*

Proof. This result is a consequence of the fact that SN P systems can simulate register machines. Specifically, as proved in [4], starting from a register machine M (we do not need precise definitions and notations; details about register machines can be found in many books), we construct an SN P system Π which halts its computation with $2n$ spikes in a specified neuron σ_{out} if and only if n can be generated by the register machine M; in the halting moment, a neuron σ_{l_h} of Π associated with the label of the halting instruction of M gets two spikes and fires. The neuron σ_{out} contains no rule used in the simulation of M (the corresponding register is only incremented, but never decremented – see the details of the construction from [4]).

Now, consider a language $L \subseteq b_2^*, L \in RE$. There is a register machine M such that $n \in N(M)$ if and only if $b_2^n \in L$. Starting from such a machine M, we construct the system Π as in [4], having the properties described above. We append to the system Π six more neurons, as indicated in Figure 4. There is a marked spike in neuron σ_1, and it will stay here during all the simulation of M. In the moment when neuron σ_{l_h} of Π spikes, its spike goes both to neuron σ_{out} and to neuron σ_1.

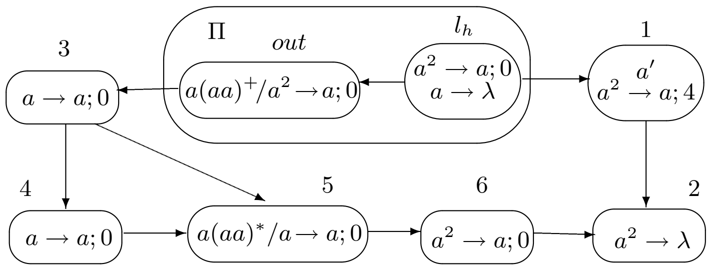

Figure 4: The SN P system from the proof of Theorem 4.6

Neurons $\sigma_3, \sigma_4, \sigma_5, \sigma_6$ send a spike to neuron σ_2 only when neuron σ_{out} has finished its work (this happens after n steps of using the rule $a(aa)^+/a^2 \to a; 0$, for $2n$ being the contents of neuron σ_{out} in the moment when neuron σ_{l_h} spikes).

The marked spike leaves neuron σ_1 four steps after using the rule $a^2 \to a; 4$, hence five steps after the spiking of neuron σ_{l_h}. This means that the marked spike waits in neuron σ_2 exactly n steps. When the spike of neuron σ_6 reaches neuron σ_2, the two spikes present here, the marked one included, are forgotten.

Thus, the traces of the marked spike are of the form $b_1^r b_2^n$, for some $r \geq 1$ and $n \in N(M)$. By means of the left derivative with the regular language b_1^* we can remove prefixes of the form b_1^k and by means of the intersection with b_2^* we ensure that the maximal prefix of this form is removed. Similarly, the projection $h : \{b_1, b_2\}^* \longrightarrow \{b_1, b_2\}^*$ defined by $h(b_1) = \lambda$, $h(b_2) = b_2$, removes all occurrences of b_1. Consequently, $L = (b_1^* \backslash T(\Pi)) \cap b_2^* = h(T(\Pi))$.

The system Π from [4] (Theorem 7.1 there) has the rule complexity described by $rule_2, cons_3, forg_3$; one sees that the construction from Figure 4 does not increase these parameters. $\square$

In what concerns the number of neurons, we do not have a bound for the language L' from the previous theorem.

Corollary 4.1 *Each family $TSNP_*(reg_k, cons_p, forg_q)$ with $k \geq 2, p \geq 3, q \geq 3$, is incomparable with each family FL which (i) contains the singleton languages, (ii) is closed under left quotients with regular languages and intersection with regular languages, or under projections, and (iii) does not contain all unary recursively enumerable languages.*

5 Final Remarks

We have considered here the possibility of generating a language by means of a spiking neural P system by following the traces of a distinguished spike during a halting computation. The power of such devices seems to be rather large – Theorem 4.6 – but we do not have a precise characterization of the obtained families of languages. Theorem 4.6

also raises the natural question whether RE languages over arbitrary alphabets can be represented starting from trace languages of SN P systems.

The main difficulty in handling the trace languages of SN P systems is, in the previous setup, the non-determinism of the marked spike evolution. A possible way to better control the marked spike is to consider spiking rules of the forms $E'/a^c \rightarrow a; d$ and $E/a'a^c \rightarrow a'; d$, with the meaning that the prime indicates whether or not the rule consumes the marked spike: this is not the case when the prime is attached to E, but this happens if the prime marks one of the consumed spikes, and hence also the produced spike. This removes the non-determinism induced by the use of the marked spike when applying the rules, but still non-determinism remains in what concerns the choice of synapses towards neighboring neurons (and this was the basis of the easy counterexample from Theorem 4.1). How also this non-determinism can be removed remains as a research topic.

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 4th author was supported by Project TIN2005-09345-C04-01 of the Ministry of Education and Science of Spain, cofinanced by FEDER funds.

References

[1] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez. On string languages generated by spiking neural P systems. Submitted, 2006.

[2] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth. Normal forms for spiking neural P systems. Submitted, 2006.

[3] M. Ionescu, C. Martin-Vide, A. Păun, Gh. Păun. Membrane systems with symport/antiport: (unexpected) universality results. In *Proc. 8th International Meeting of DNA Based Computing* (M. Hagiya, A. Ohuchi, eds.), Japan, 2002, 151–160.

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

[5] W. Maass. Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.

[6] Gh. Păun. *Membrane Computing – An Introduction*. Springer, Berlin, 2002.

[7] 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 [9]).

[8] G. Rozenberg, A. Salomaa, eds. *Handbook of Formal Languages*. Springer, Berlin, 1997.

[9] The P Systems Web Page. `http://psystems.disco.unimib.it`.