Handling Languages with Spiking Neural P Systems with Extended Rules

Haiming CHEN[1], Tseren-Onolt ISHDORJ[3],
Gheorghe PĂUN[2,3], Mario J. PÉREZ-JIMÉNEZ[3]

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

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

[3]Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: tseren@yahoo.com, gpaun@us.es, marper@us.es

Abstract. We consider spiking neural P systems with spiking rules allowed to introduce zero, one, or more spikes at the same time. A tool-kit for computing (some) operations with languages generated by such systems is provided. Computing the union of languages is easy. However, computing the concatenation or the intersection with a regular language is not so easy. A way to compute weak encoding is also provided. The main results of the computing power of the obtained systems are then presented, when considering them as number generating and as language generating devices. In particular, we find direct characterizations of finite and recursively enumerable languages (without using any squeezing mechanism, as it was necessary in the case of restricted rules).

1. Introduction

We combine here two ideas recently considered in the study of the spiking neural P systems (in short, SN P systems) introduced in [3], namely the *extended* rules from [5] and the *string generation* from [1].

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train*. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the string corresponding to the sequence of spikes which exit the output neuron. This sequence is a binary one, with 0 associated with a step when no spike is emitted and 1 associated with a step when a spike is emitted.

The case of SN P systems as number generators was investigated in several papers, starting with [3], where it is proved that such systems are Turing complete (hence also universal, because the proof is constructive; universality in a rigorous framework was investigated in [5]). In turn, the string case is investigated in [1], where representations of finite, regular, and recursively enumerable languages were obtained, but also finite languages were found which cannot be generated in this way.

Here we consider an extension of the rules, already used in [5], namely we allow rules of the form $E/a^c \rightarrow a^p$, with the following meaning: if the content of the neuron is described by the regular expression E, then c spikes are consumed and p are produced and sent to the neurons to which there exist synapses leaving the neuron where the rule is applied (more precise definitions will be given in the next section). Thus, these rules cover and generalize at the same time both spiking rules and forgetting rules as considered so far in this area – with the mentioning that we do not also consider here a delay between firing and spiking, because in the proofs we never need such a delay.

In Section 3 we present constructions of SN P systems for computing some usual operations with languages: union, concatenation, weak coding, intersection with regular languages. Computing the union of languages is easy, but computing the concatenation or the intersection with a regular language is not so easy. A way to compute weak encoding is also provided. The main results of the computing power of these systems are recalled in Section 4. As expected, the use of extended rules allows much simpler constructions for the proof of universality in the case of considering SN P systems as number generators. More interesting is the case of strings produced by SN P systems with extended rules: we associate a symbol b_i to a step when the system sends i spikes into the environment, with two possible cases – b_0 is used as a separated symbol, or it is replaced by λ (sending no spike outside is interpreted as a step when the generated string is not grown). The first case is again restrictive: not all minimal linear languages can be obtained, but still results stronger than those from [1] can be proved in the new framework because of the possibility of removing spikes under the control of regular expressions. The freedom provided by the existence of steps when we have no output makes possible direct characterizations of finite and recursively enumerable languages (not only representations, modulo various operations with languages, as obtained in [1] for the standard binary case).

2. Spiking Neural P Systems with Extended Rules

We assume the reader to be familiar with basic language and automata theory, e.g., from [7] and [8], so that we introduce here only some notations and notions used later in the paper.

If $x = a_1 a_2 \dots a_n$, $a_i \in V$, $1 \leq i \leq n$, then $mi(x) = a_n \dots a_2 a_1$. A morphism $h : V_1^* \longrightarrow V_1^*$ such that $h(a) \in \{a, \lambda\}$ for each $a \in V_1$ is called a projection, and a morphism $h : V_1^* \longrightarrow V_2^*$ such that $h(a) \in V_2 \cup \{\lambda\}$ for each $a \in V_1$ is called a weak coding. If $L_1, L_2 \subseteq V^*$ are two languages, the left and right quotients of L_1 with respect to L_2 are defined by $L_2 \backslash L_1 = \{w \in V^* \mid xw \in L_1 \text{ for some } x \in L_2\}$, and respectively $L_1 / L_2 = \{w \in V^* \mid wx \in L_1 \text{ for some } x \in L_2\}$. When the language L_2 is a singleton, these operations are called left and right derivatives, and denoted by $\partial_x^l(L) = \{x\} \backslash L$ and $\partial_x^r(L) = L / \{x\}$, respectively. We denote by FIN, REG, CF, CS, RE the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages. The family of Turing computable sets of numbers is denoted by NRE (these sets are length sets of RE languages, hence the notation). Let $V = \{b_1, b_2, \dots, b_m\}$, for some $m \geq 1$. For a string $x \in V^*$, let us denote by $val_m(x)$ the value in base $m+1$ of x (we use base $m+1$ in order to consider the symbols $b_1, \dots, b_m$ as digits $1, 2, \dots, m$, thus avoiding the digit 0 in the left hand of the string). We extend this notation in the natural way to sets of strings.

We directly introduce the type of SN P systems we investigate in this paper; the reader can find details about the standard definition in [3], [6], [1], etc.

An *extended spiking neural P system* (abbreviated as *extended SN P system*), of degree $m \geq 1$, is a construct of the form $\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_0)$, where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \dots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the form $E/a^c \rightarrow a^p$, where E is a regular expression over a and $c \geq 1, p \geq 0$, with the restriction $c \geq p$;
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \dots, m\}$ indicates the *output neuron* (σ_{i_0}) of the system.

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

Note that we do not consider here a delay between firing and spiking (i.e., rules of the form $E/a^c \rightarrow a^p; d$, with $d \geq 0$), because we do not need this feature in the proofs below, but such a delay can be introduced in the usual way. (As a consequence, here the neurons are always open.)

If a rule $E/a^c \rightarrow a^p$ has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a^p$.

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

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

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

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

As the result of a computation, in [3] and [6] one considers the distance between two consecutive steps when there are spikes which exit the system, with many possible variants: taking the distance between the first two occurrences of 1 in the spike train, between all consecutive occurrences, considering only alternately the intervals between occurrences of 1, etc. For simplicity, we consider here only the first case mentioned above: we denote by $N_2(\Pi)$ the set of numbers generated by an SN P system in the form of the number of steps between the first two steps of a computation when spikes are emitted into environment, and by $Spik_2SN^eP_m(rule_k, cons_p, prod_q)$ the family of sets $N_2(\Pi)$ generated by SN P systems with at most m neurons, at most k rules in each neuron, consuming at most p and producing at most q spikes. Any of these parameters is replaced by $*$ if it is not bounded.

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

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

The respective families are denoted by $L_\alpha SN^eP_m(rule_k, cons_p, prod_q)$, where $\alpha \in \{bin, res, \lambda\}$ and parameters m, k, p, q are as above.

3. A Tool-Kit for Handling Languages

We present first several extended SN P systems which perform operations with languages in order to let the reader have an idea how these systems work.

For instance, starting with two SN P systems Π_1, Π_2, we look for a system Π which generates the language $L_\lambda(\Pi_1) \diamond L_\lambda(\Pi_2)$, where $\diamond$ is a binary operation with languages.

For the union of languages, such a system Π is easy to be constructed (as already done in [3]): we start with the systems Π_1, Π_2 without any spike inside and we consider a module which non-deterministically activates one of these systems, by introducing in their neurons as many spikes as we have in the initial configurations of Π_1 and Π_2.

Not so simple is the case of concatenation, which, however, can be handled as in Fig. 1. We start with system Π_1 as it is (with the neurons loaded with the necessary spikes), and with system Π_2 without any spike inside.

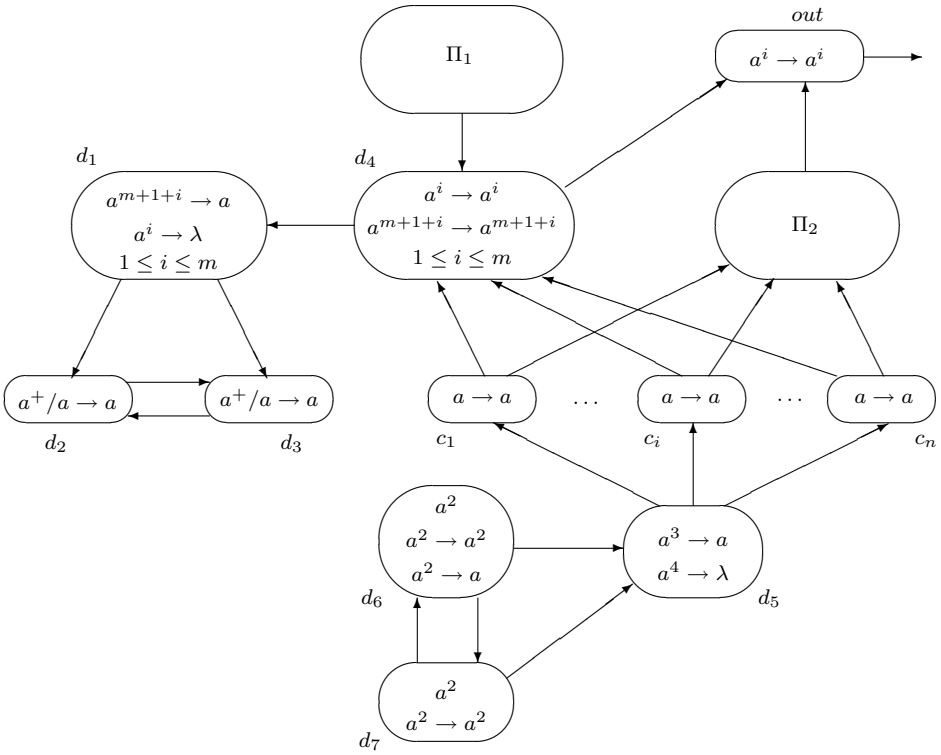

Fig. 1. Computing the concatenation of two languages.

We have in Fig. 1 three sub-systems/modules with specific tasks to solve. For instance, neurons $\sigma_{d_5}, \sigma_{d_6}, \sigma_{d_7}$ non-deterministically choose a moment when the string generated by system Π_1 is assumed completed. After using rule $a^2 \to a$ in σ_{d_6}, neuron σ_{d_5} fires, this activates neurons $\sigma_{c_1}, \ldots, \sigma_{c_n}$, and these neurons both "flood" neuron σ_{d_4} with $m+1$ spikes and activate the neurons of system Π_2, introducing as many spikes as Π_2 has in its initial configuration. Specifically, we have $n = \max\{m+1, spin(\Pi_2)\}$, where m is the cardinality of the alphabet we work with, and $spin(\Pi_2)$

is the maximum of the number of spikes present in any neuron of Π_2 in the initial configuration. Then we have synapses (c_i, d_4) for $1 \leq i \leq m+1$, and (c_i, k), for σ_k a neuron in Π_2, for $1 \leq i \leq n_k$, where n_k is the number of spikes present in σ_k in the initial configuration of Π_2.

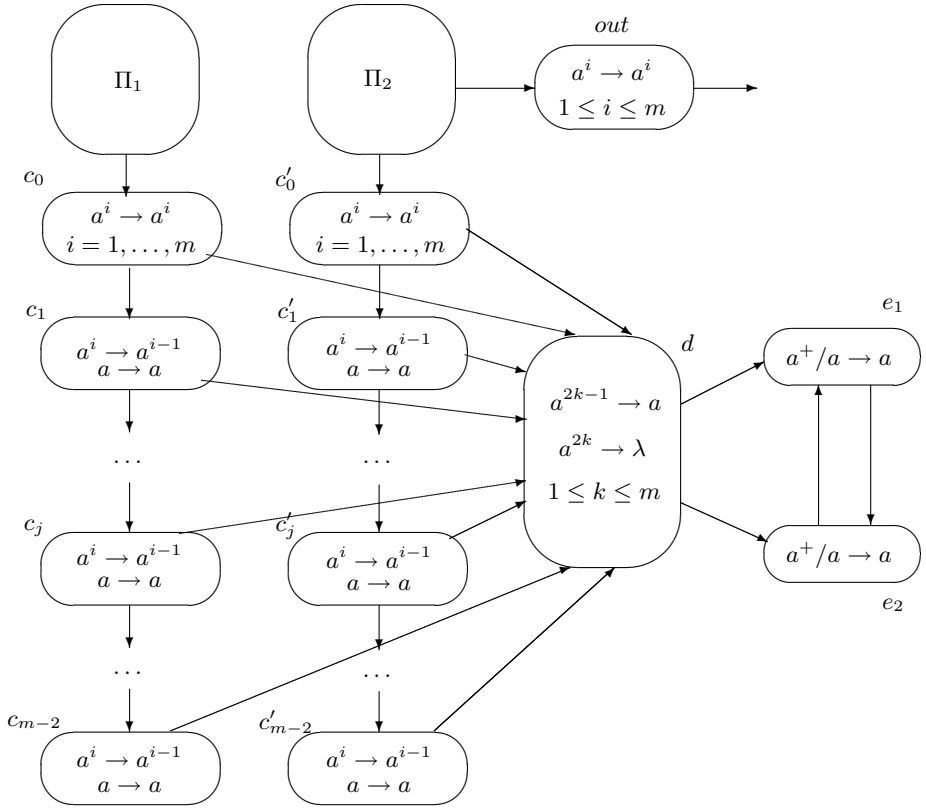

Fig. 2. Computing the intersection with a regular language.

The pair of neurons $\sigma_{d_4}, \sigma_{out}$ takes care of the output of the whole system, first passing the output of Π_1 to σ_{out} and then taking the output of Π_2 and sending it out. If σ_{d_4} receives any further spike from Π_1 after neurons $\sigma_{d_5}, \sigma_{d_6}, \sigma_{d_7}$ have "decided" that the work of Π_1 is finished, then σ_{d_4} fires (note that it cannot fire for exactly $m+1$ spikes), this makes σ_{d_1} fire, and then the computation will never finish, because of the pair of neurons $\sigma_{d_2}, \sigma_{d_3}$.

Thus, the computation ends if and only if after sending out a complete string generated by Π_1 we also send out a string generated by Π_2, hence we generate the concatenation of strings produced by the two systems.

Consider now an arbitrary SN P system Π_1 and an SN P system Π_2 simulating a regular grammar G, with the following changes: chain rules $A_i \to A_i$ are added to grammar G for all nonterminals A_i; then, we assume that the number of rules (n in

the construction) is strictly bigger than the number of symbols (m) – if this is not the case, then we simply duplicate some rules. The system looks now as in Fig. 3 (k can be 0 only for chain rules $A_i \to A_i$, where $b_0 = \lambda$). Thus, after simulating a rule $A_i \to b_k$, neurons σ_1, σ_2 are "flooded" and have to stop. The grammar G – and hence also Π_2 – outputs a terminal symbol after an arbitrary number of steps of using chain rules $A_i \to A_i$, hence steps when nothing exits the system. This makes possible the synchronization of Π_1 and Π_2 in the sense that they output spikes in the same steps. What remains to do is to compare the number of spikes emitted by the two systems, so that we can select the strings from the intersection $L_\lambda(\Pi_1) \cap L_\lambda(\Pi_2)$.

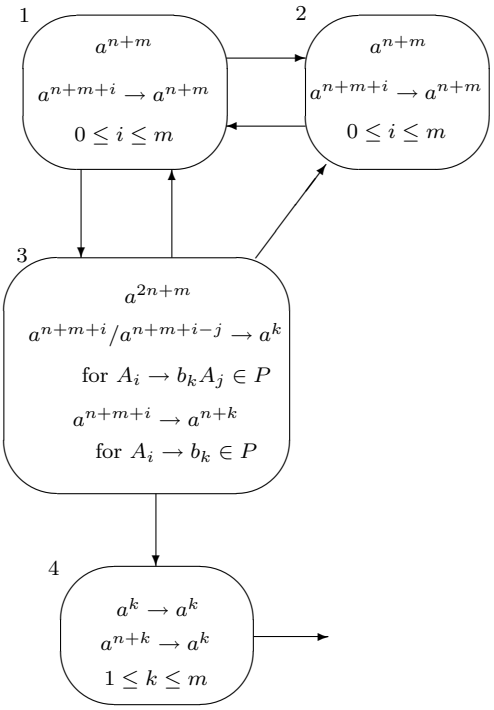

Fig. 3. Simulating a regular grammars having chain rules.

This is ensured as suggested in Fig. 2 (in order to keep the figure smaller, we have not indicated the range of parameter i, but it is as follows: in all neurons σ_{c_j} and $\sigma_{c'_j}$, $1 \le j \le m-2$, we have $2 \le i \le m$). If the two systems Π_1 and Π_2 do not spike at the same time or one sends out $r \ge 1$ spikes and the other one $s \ge 1$ spikes for $r \ne s$, then the neurons $\sigma_{e_1}, \sigma_{e_2}$ will get activated and the computation never stops: the spikes emitted by the two systems circulate from top down along the chains of neurons $\sigma_{c_0}, \sigma_{c_1}, \ldots, \sigma_{c_{m-2}}$ and $\sigma_{c'_0}, \sigma_{c'_1}, \ldots, \sigma_{c'_{m-2}}$, and if we do not obtain exactly one spike at the same time in the two columns, then the neuron σ_d fires and activates the neurons $\sigma_{e_1}, \sigma_{e_2}$.

We do not know how to compute – in an elegant way – morphisms, but the particular case of weak codings can be handled as in Fig. 4. The difficulty is to have

$h(b_i) = b_j$ with $i < j$, and to this aim the "spike supplier" pair of neurons $\sigma_{c_1}, \sigma_{c_2}$ is considered. In each step, they send $m+1$ spikes to neuron σ_{c_3}. If this neuron receives nothing at the same time from the system Π, then the $m+1$ spikes are forgotten. If i spikes come from system Π, $1 \leq i \leq m$, then, using the $m+1+i$ spikes, neuron σ_{c_3} can send $j+1$ spikes to the output neuron, which emits the right number of spikes to the environment.

At any time, the neurons $\sigma_{c_1}, \sigma_{c_2}$ can stop their work; if this happens prematurely (before having the system Π halted), then neuron σ_{c_3} will emit only one spike, and this triggers the "never halting module", composed of the neurons $\sigma_{c_4}, \sigma_{c_5}, \sigma_{c_6}$, which will continue to work forever.

The reader can check that the system produces indeed the language $h(L_\lambda(\Pi))$, for a weak coding h which moves some b_i into b_k, and erases other symbols b_j.

We do not know how to compute arbitrary morphisms or the other AFL operations, Kleene + and inverse morphisms.

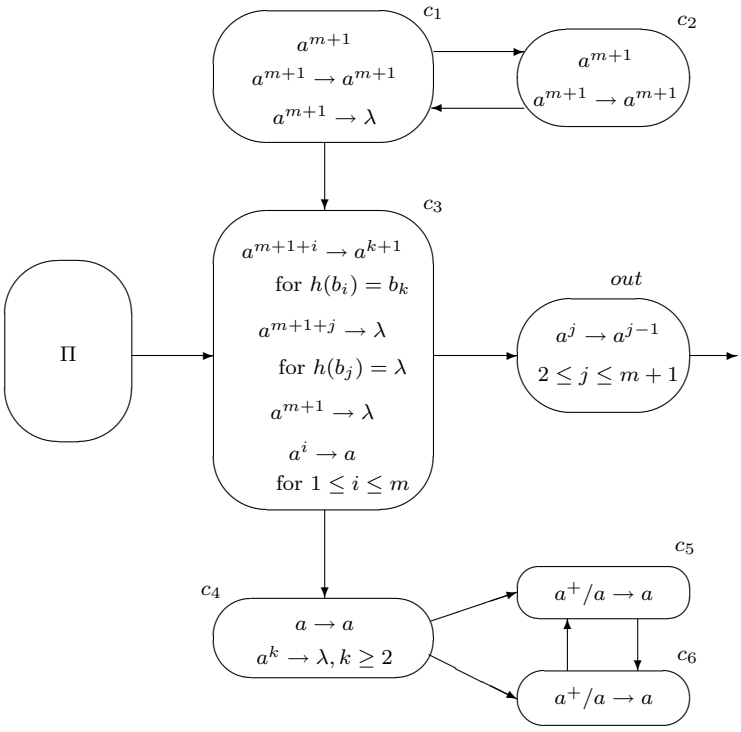

Fig. 4. Computing a weak coding.

A possible way to address these problems is to reduce them to another problem, that of introducing delays of arbitrarily many steps in between any two steps of computations in an arbitrary SN P system Π (in the same way as the chain rules

introduce such "dummy steps" in the work of a regular grammar). If such a slowing-down of a system would be possible, then we can both compute arbitrary morphisms and the intersection of languages generated by two arbitrary SN P systems (not only with one of them generating a regular language, as above).

Another open problem of interest (but difficult, we believe) is to find an SN P system, as small as possible in the number of neurons, generating a Dyck language (over at least two pairs of parentheses). If such a systems would be found, then a representation of context-free languages would be obtained, using the Chomsky-Schützenberger characterization of these languages as the weak coding of the intersection of a Dyck language with a regular language.

4. Computing Power of Extended SN P Systems

For the reader convenience, we recall now (without a proof) several results concerning the computing power of SN P systems with extended rules. Details can be found in [2].

4.1. Extended SN P Systems as Number Generators

Because non-extended SN P systems are already computationally universal, this result is directly valid also for extended systems. However, the construction on which the proof is based is much simpler in the extended case (in particular, it does not use the delay feature), that is why we mention it.

Theorem 4.1. $NRE = Spik_2SN^eP_*(rule_5, cons_5, prod_2)$.

4.2. Languages in the Restricted Case

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

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

4.2.1. A Characterization of FIN

SN P systems with standard rules cannot generate all finite languages (see [1]), but extended rules help in this respect.

Lemma 4.1. $L_\alpha SN^eP_1(rule_*, cons_*, prod_*) \subseteq FIN$, $\alpha \in \{res, \lambda\}$.

Lemma 4.2. $FIN \subseteq L_\alpha SN^eP_1(rule_*, cons_*, prod_*)$, $\alpha \in \{res, \lambda\}$.

Theorem 4.2. $FIN = L_{res}SN^eP_1(rule_*, cons_*, prod_*) = L_\lambda SN^eP_1(rule_*, cons_*, prod_*)$.

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

Proposition 4.1. $L_\alpha SN^e P_2(rule_2, cons_3, prod_3) - FIN \neq \emptyset$, $\alpha \in \{res, \lambda\}$.

4.2.2. Representations of Regular Languages

Such representations are obtained in [1] starting from languages of the form $L_{bin}(\Pi)$, but in the extended SN P systems, regular languages can be represented in an easier and more direct way like in Figure 3.

Theorem 4.3. *If $L \subseteq V^*$, $L \in REG$, then $\{b_0\}L \in L_{res}SN^eP_4(rule_*, cons_*, prod_*)$.*

Corollary 4.1. *Every language $L \in REG, L \subseteq V^*$, can be written in the form $L = \partial^l_{b_0}(L')$ for some $L' \in L_{res}SN^eP_4(rule_*, cons_*, prod_*)$.*

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

Theorem 4.4. *If $L \subseteq V^*$, $L \in REG$, then $L\{b_0\} \in L_{res}SN^eP_3(rule_*, cons_*, prod_*)$.*

Corollary 4.2. *Every language $L \in REG, L \subseteq V^*$, can be written in the form $L = \partial^r_{b_0}(L')$ for some $L' \in L_{res}SN^eP_3(rule_*, cons_*, prod_*)$.*

4.2.3. Going Beyond REG

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

Lemma 4.3. *The number of configurations reachable after n steps by an extended SN P system of degree m is bounded by a polynomial $g(n)$ of degree m.*

Theorem 4.5. *If $f : V^+ \longrightarrow V^+$ is an injective function, $card(V) \geq 2$, then there is no extended SN P system Π such that $L_f(V) = \{x\, f(x) \mid x \in V^+\} = L_{res}(\Pi)$.*

Corollary 4.3. *The following languages are not in $L_{res}SN^eP_*(rule_*, cons_*, prod_*)$ (in all cases, $card(V) = k \geq 2$):*

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

Note that language L_1 above is a non-regular minimal linear one, L_2 is context-sensitive non-context-free, and L_3 is non-semilinear. In all cases, we can also add a fixed tail of any length (e.g., considering $L'_1 = \{x\, mi(x)z \mid x \in V^+\}$, where $z \in V^+$ is a

given string), and the conclusion is the same – hence a result like that in Theorem 4.4 cannot be extended to minimal linear languages.

4.3. Languages in the Non-Restricted Case

As expected, the possibility of having intermediate steps when no output is produced is helpful, because this provides intervals for internal computations. In this way, we can get rid of the operations used in [1] and in the previous section when dealing with regular and with recursively enumerable languages.

4.3.1. Relationships with REG

Lemma 4.4. $L_\lambda SN^eP_2(rule_*, cons_*, prod_*) \subseteq REG$.

Lemma 4.5. $REG \subseteq L_\lambda SN^eP_3(rule_*, cons_*, prod_*)$.

This last inclusion is proper:

Proposition 4.2. $L_\lambda SN^eP_3(rule_3, cons_4, prod_2) - REG \neq \emptyset$.

Corollary 4.4. $L_\lambda SN^eP_1(rule_*, cons_*, prod_*) \subset L_\lambda SN^eP_2(rule_*, cons_*, prod_*)$ $\subset L_\lambda SN^eP_3(rule_*, cons_*, prod_*)$, *strict inclusions.*

4.3.2. Going Beyond CF

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

Theorem 4.6. *The family* $L_\lambda SN^eP_3(rule_3, cons_6, prod_4)$ *contains non-semilinear languages.*

4.3.3. A Characterization of RE

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

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

Lemma 4.6. $mRE \subseteq mL_\lambda SN^eP_*(rule_{m'}, cons_m, prod_m)$, *where* $m' = \max(m, 2)$ *and* $m \geq 1$.

Theorem 4.7. $RE = L_\lambda SN^eP_*(rule_*, cons_*, prod_*)$.

Corollary 4.5. *Every language* $L \in RE, L \subseteq V^*$, *can be written in the form* $L = h(L')$ *for some* $L' \in L_{res}SN^eP_*(rule_*, cons_*, prod_*)$, *where* h *is a projection on* $V \cup \{b_0\}$ *which removes the symbol* b_0.

5. Final Remarks

We have presented here some constructions for performing operations with languages generated by SN P systems with extended rules (rules allowing to introduce several spikes at the same time). We have also recalled some results of the power of such SN P systems both as number generators and as language generators. Characterizations of finite and recursively enumerable languages, and representations of regular languages have been found.

Finding characterizations (or at least representations) of other families of languages from Chomsky hierarchy and Lindenmayer area remains as a research topic. It is also of interest to investigate the possible hierarchy on the number of neurons, extending the result from Corollary 4.4.

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

References

[1] CHEN, H., FREUND, R., IONESCU, M., PĂUN, GH., PÉREZ-JIMÉNEZ, M.J., *On string languages generated by spiking neural P systems*, *Proc. Fourth Brainstorming Week on Membrane Computing*, vol. **I**, Sevilla, 2006, pp. 169–193. Available at [9].

[2] CHEN, H., ISHDORJ, T.-O., PĂUN, GH., PÉREZ-JIMÉNEZ, M.J., *Spiking neural P systems with extended rules*, *Proc. Fourth Brainstorming Week on Membrane Computing*, vol. **I**, Sevilla, 2006, pp. 241–265. Available at [9].

[3] IONESCU, M., PĂUN, GH., YOKOMORI, T., *Spiking neural P systems*, Fundamenta Informaticae, **71**, 2–3 (2006), pp. 279–308.

[4] MINSKY, M., *Computation – Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.

[5] PĂUN, A., PĂUN, GH., *Small universal spiking neural P systems*, BioSystems, to appear, 2006.

[6] PĂUN, GH., PÉREZ-JIMÉNEZ, M.J., ROZENBERG, G., *Spike trains in spiking neural P systems*, Intern. J. Found. Computer Sci., **17**, 4 (2006), pp. 975–1002.

[7] ROZENBERG, G., SALOMAA, A. (eds.), *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.

[8] SALOMAA, A., *Formal Languages*, Academic Press, New York, 1973.

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