Families of languages encoded by SN P systems

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València,
jsempere@dsic.upv.es

Abstract. In this work, we propose the study of SN P systems as classical information encoders. By taking the spike train of an SN P system as a (binary) source of information, we can obtain different languages according to a previously defined encoding alphabet. We provide a characterization of the language families generated by the SN P systems in this way. This characterization depends on the way we define the encoding scheme: bounded or not bounded and, in the first case, with one-to-one or non injective encodings. Finally, we propose a network topology in order to define a cascading encoder.

Keywords: SN P systems, formal languages, codes, word enumerations.

1 Introduction

Spiking Neural P systems (SN P systems) were proposed as a model that combines some aspects of neural networks and some others from P systems. Basically, they have been proposed as acceptor systems, language generators or (encoded) word transducers. We focus our attention on the generative capacity of this model. Typically, the language generated by the system is taken as the set of binary words defined by the spike train that the system outputs. This approach was first formulated in [5], and later developed in [1].

In this work, we consider a SN P system as a classical information source that can generate encoded strings as outputs. The binary codes can be established in an exogenous predefined way and, for a fixed encoding alphabet, the system generates a (possibly) infinite language. So, any SN P system can generate different languages depending on the encoding that has been defined. We will overview different situations within this approach: First, for a fixed integer value we will distinguish between one-to-one and non-injective cases. Then, different encoding schemes where the integer value tends to infinity will be overviewed and, finally, a network topology that connect different SN P systems to produce a cascading encoder will be proposed.

2 Basic concepts

We consider that the reader knows basic concepts and results from formal language theory, otherwise we refer to [10]. In the same way, we consider that the

reader is familiar with the basic concepts and results about P systems and membrane computing, otherwise we refer to [8] and [4].

In what follows, we provide some basic definitions related to Spiking Neural P systems from [4].

Definition 1. *A spiking neural P system (*SN P system*, for short) of degree* $m \geq 1$ *is defined by the tuple* $\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma_m, syn, in, out)$ *where*

1. $O = \{a\}$ *is the singleton alphabet of* spikes
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* σ_i
   (b) R_i *is a finite set of rules of the following two forms*
      i. *firing or spiking rules* $E/a^c \rightarrow a; d$ *where* E *is a regular expression over* a, *and* $c \geq 1$, $d \geq 0$ *are integer numbers. We will omit* E *whenever it be equal to* a^c, *and we will omit* d *if it is equal to 0.*
      ii. *forgetting rules* $a^s \rightarrow \lambda$, *for* $s \geq 1$, *with the restriction that for each spiking rule* $E/a^c \rightarrow a; d$ *then* $a^s \notin L(E)$ *(*$L(E)$ *is the regular language defined by* E*)*
3. $syn \subseteq \{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\}$ *with* $(i, i) \notin syn$, *for* $1 \leq i \leq m$, *is the directed graph of synapses between neurons;*
4. $in, out \in \{1 \cdots m\}$ *indicate the* input *and the* output *neurons of* Π.

At neuron σ_i, the firing rules $E/a^c \rightarrow a; d$ are applied as follows: if the neuron contains $k \geq c$ spikes and $a^k \in E$ then c spikes are removed from σ_i and one spike is delivered to all the neurons σ_j connected to σ_i with $(i, j) \in syn$. If $d = 0$ the spike is immediately emitted, otherwise it is emitted after d computation steps (during these computation steps, the neuron is closed, so it cannot receive spikes, it cannot apply the rules and, subsequently it cannot send new spikes). At neuron σ_i, the forgetting rule $a^s \rightarrow \lambda$ is applied as follows: if the neuron σ_i contains exactly s spikes and no firing rule can be applied then all the spikes of the neuron are removed.

A configuration of the system at an instant t during a computation is defined by the tuple $(i_1/t_1, \cdots, i_m/t_m)$ that denotes the number of spikes that are at every neuron together with the computation time needed to open the neuron. The initial configuration of the SN P system is $(n_1/0, \cdots, n_m/0)$. A computation of Π is a (finite or infinite) sequence of configurations such that: (a) the first term of the sequence is the initial configuration of the system and each of the remaining configurations are obtained from the previous one by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and (b) if the sequence is finite (called halting computation) then the last term of the sequence is a halting configuration, that is a configuration where all neurons are open and no rule can be applied to it.

During a computation, the moments of time when a spike is emitted by the output neuron will be marked by '1' while the other moments are marked by '0'. The binary sequence that is obtained in such a way during the computation is called the *spike train* of the system. In the sequel, we will omit the input neuron, and we will work with SN P systems as *language generators*.

The language generated by any SN P system depends on the interpretation given to the spike train that it outputs. For any halting computation, we can take the finite spike train as a string over the binary alphabet $B = \{0, 1\}$, or we can take the intervals between output spikes with different approaches such as those described in [9]. In what follows, we will consider the spike train as a generator of binary strings.

For any SN P system Π, the language generated by Π as described before will be denoted by $L_1(\Pi)$.

3 Languages encoded by SN P systems

Our approach to the languages generated by SN P systems is different from the previously referred ones. Actually, the present research idea occurred in a framework related to classical communication channels with encoded information, where, for every SN P system, different languages can be associated to the system depending on a parameter that fixes a time window to analyze the spike train.

For any SN P system Π, we take the binary language $L_1(\Pi)$ and we encode blocks of k digits, for all the positive integer values k, in such a way that languages $L_k(\Pi)$ are obtained. Of course, we have to take care of the case when the spike train is not of a length which is a multiple of the considered k. In this case, we add symbols 0 so that the obtained binary string is of a length divisible by k.

More formally, let $B = \{0, 1\}$ be the binary alphabet, let $k \geq 1$ be a natural number, let B^k be the set of all strings from B whose length is k, and V_k be an alphabet. In general, any alphabet can be considered but we will associate a different symbol for every word in B^k. Consider a mapping $\varphi_k : B^k \longrightarrow V_k$. For each string $w \in B^*$ we consider the string ${}_kw = w0^t$, where $t = \min\{n \geq 0 \mid |w0^n| \text{ is a multiple of } k\}$.

The string ${}_kw$ can be written in the form ${}_kw = x_1x_2 \dots x_s$, such that $|x_j| = k$ for all $j = 1, 2, \dots, s$. Then, φ_k can be extended to $(B^k)^*$ in the natural way: $\varphi_k(y_1y_2 \dots y_t) = \varphi_k(y_1)\varphi_k(y_2) \dots \varphi_k(y_t)$ for all $y_i \in B^k, 1 \leq i \leq t, t \geq 0$. We can see the encoding approach that we have just described in Fig.1.

Thus, for an SN P system Π and an encoding φ_k as above, we can define the language

$$L_{\varphi_k}(\Pi) = \{\varphi_k({}_kw) \mid w \in L_1(\Pi)\}.$$

In what follows, we write $L_k(\Pi)$ instead of $L_{\varphi_k}(\Pi)$. The language $L_k(\Pi)$ depends on the encoding φ_k, hence a family of languages can be associated with Π by varying k and the mapping φ_k. Observe, that the language $L_1(\Pi)$, as defined at the end of section 2, is a particular case of $L_k(\Pi)$ when $k = 1$, given that φ_1 can be trivially defined as the identity mapping. We define the family of languages $F(\Pi) = \{L_k(\Pi) | k \geq 1\}$.

Already at this very general level there appear several research issues. In what follows, we consider two classes of mappings φ_k and investigate the closure properties of the corresponding families of languages generated by SN P systems.

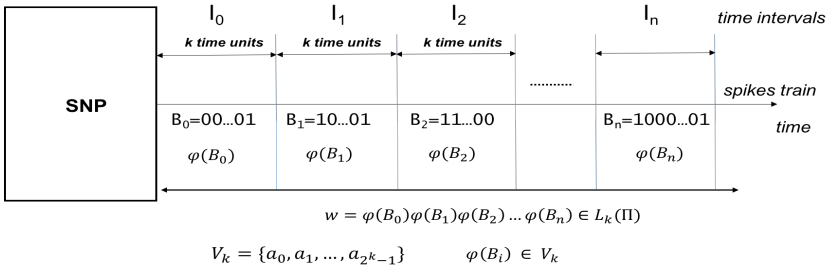

Fig. 1. SN P systems as language encoders: The case of intervals of lenght k.

3.1 The one-to-one case

A natural possibility is to order in a precise way, e.g., lexicographically, the strings in B^k, and to associate with each of them a distinct symbol from an alphabet V_k with 2^k elements, that is, assuming that φ_k is injective.

We can establish the following properties depending on whether $L_1(\Pi)$ is finite or not.

Property 1. Let Π be an SN P system. Then, if $L_1(\Pi)$ is finite the so are each $L_k(\Pi)$ for $k > 1$.

From the Property 1, we can deduce that if $L_1(\Pi)$ is finite, then the family $F(\Pi)$ is finite, up to a renaming of symbols of alphabets V_k.

Property 2. Let Π be an SN P system. Then, if $L_1(\Pi)$ is infinite, then so are each $L_k(\Pi)$ for $k > 1$.

If $L_1(\Pi)$ is infinite, then $F(\Pi)$ can be an infinite family, because the alphabet of $L_{k+1}(\Pi)$ might be larger than the alphabet of $L_k(\Pi)$. This is the case, for instance, for the SN P system Π generating $L_1(\Pi) = \{1^n01^m \mid n, m \geq 1\}$ (which is an infinite regular language).

The fact that the encoding is one-to-one is rather restrictive: the passing from the binary language $L_1(\Pi)$ to a given $L_k(\Pi)$ can be done by means of a sequential transducer (a gsm, in the usual terminology, [10]). Conversely, the passage from $L_k(\Pi)$ to $L_1(\Pi)$ is done by an one-to-one (non-erasing) morphism, which implies that the converse passage is done by an inverse morphism. This observation can be formally formulated as follows.

Proposition 1. *If $L_1(\Pi) \in FL$, where FL is a family of languages closed under gsm mappings or under inverse morphisms, then $L_k(\Pi) \in FL$, for all $k \geq 1$. If FL is closed under non-erasing morphisms and $L_k(\Pi) \in FL$, then also $L_1(\Pi) \in FL$.*

Families as FL above are REG, LIN, CF in the Chomsky hierarchy, hence if $L_1(\Pi)$ is regular, linear or context-free, then so are all languages $L_k(\Pi)$, and conversely.

This means that each family $F(\Pi)$ contains only languages of the same type in the Chomsky hierarchy (for instance, it is not possible to have a context-free non-regular language $L_k(\Pi)$ together with a regular language $L_j(\Pi)$, for some $k \neq j$.

3.2 The non-injective case

The previous type-preserving Proposition 1 does not hold in the case of using encodings which are not one-to-one.

Here is an example: Consider Π such that $L_1(\Pi) = \{1^n 01^n \mid n \geq 1\}$ (SN P systems are universal, [5], hence any language can be taken as the starting language). Of course, $L_1(\Pi)$ is context-free non-regular.

Consider the encoding $\varphi_k : B^k \longrightarrow \{a, b\}$ defined by $\varphi_k(w) = a$ if $|w|_0 \leq 1$, and $\varphi_k(w) = b$ if $|w|_0 \geq 2$. We get

$$L_k(\Pi) = a^+ \cup a^* b, \text{ for } k \geq 4,$$

$L_3(\Pi) = a^+ \cup (aa)^+ b$, and $L_2(\Pi) = aa^+$.

Clearly, the languages $L_k(\Pi), k \geq 2$, are regular, in spite of the fact that $L_1(\Pi)$ is (context-free) non-regular.

The properties of the encoding is crucial for the properties of the obtained language families (this is true also in other frameworks, see, e.g., [3] and its references), hence this issue deserves further research efforts.

4 The unbounded case

In the previous section, an encoding of the languages based on blocks of length k has been considered. Now, we consider the limit case, when every string from $L_1(\Pi)$ encodes a different string while k tends to ∞.

Formally, we consider an alphabet $\Sigma = \{a_0, a_1, \cdots, a_p\}$, and the ordered set of strings $\Sigma^* = \{w_0, w_1, \cdots w_i, \cdots\}$. We define the encoding $\varphi_{int} : B^* \longrightarrow \Sigma^*$ such that for every binary string x, $\varphi_{int}(x) = w_{int(x)}$ where $int(x)$ is the integer value of x by taking x as a binary number. The encoding scheme over the SN P system is shown at Fig.2.

For a given alphabet Σ and an application $\varphi_{int} : B^* \longrightarrow \Sigma^*$, we can define the encoded language of any SN P system, as we have described before, as follows

$$L_\infty(\Pi) = \{w \in \Sigma^* \mid \exists x \in L_1(\Pi) \text{ such that } w = z_{int(x)}\}$$

Observe, that Σ^* must be ordered within a precise enumeration of all its words. In this case, the enumeration of the strings in Σ^* (actually, its order) is decisive to preserve the language class from $L_1(\Pi)$ to $L_\infty(\Pi)$.

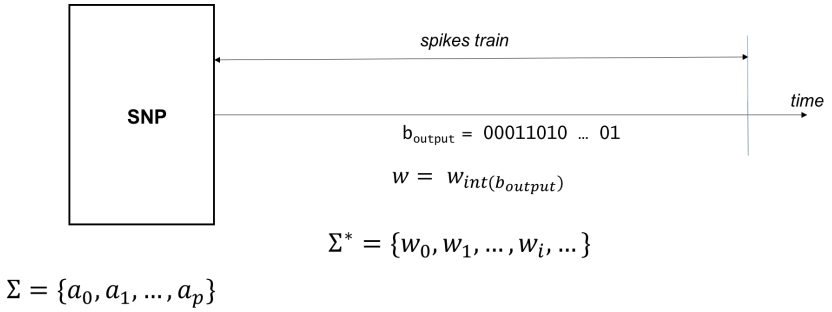

Fig. 2. SN P systems as language encoders: The unbounded case.

For example, let us take $L_1(\Pi) = \{(01)^n \mid n \geq 0\}$ that is a regular language that can be generated by an SN P system given that they have been proved to be universal.

Let $\Sigma = \{a, b\}$ and the languages $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \Sigma^* - L_1$. We consider that $L_1 = \{x_1, x_2, x_3, \cdots\}$ and $L_2 = \{y_1, y_2, y_3, \cdots\}$ are lexicographically ordered.

We can define the following enumeration over $\Sigma^* = \{z_1, z_2, \cdots, z_i, \cdots\}$, where

1. If $i \mod 2 = 0$ then $z_i = y_{\frac{i}{2}} \in L_2$ (*even indexes*)
2. If $i \mod 2 = 1$ then $z_i = x_{\lceil \frac{i}{2} \rceil} \in L_1$ (*odd indexes*)

Observe that every string $x \in L_1(\Pi) = \{(01)^n : n \geq 0\}$ encodes an odd integer number given that the binary string ends with '1'. Hence, $L_\infty(\Pi)$ is an infinite subset of L_1 given that, for every string x in $L_1(\Pi)$, the string $z_{int(x)}$ occupies an odd position and, subsequently, it belongs to L_1. Hence. $L_1(\Pi)$ is regular while L_∞ is not.

5 Networks of SN P systems as cascading encoders

Finally, we propose a new way of encoding languages by composing a finite number of SN P systems. In this case we propose a topology based on SN P systems with a tissue-like configuration within a bus connection. Our proposal is shown in Fig.3.

We have a finite set of n SN P systems defined in the usual way. We connect them in the following way: every time that the SN P system i halts, its spike train encodes an integer value k_i that is the parameter to encode the language in the SN P system $i + 1$. Hence, a network of SN P systems can be viewed as a cascading encoder for languages. If we connect the SN P systems in a bus topology then, for the iterated case, the last system is connected to the first one.

This opens a new framework which is related to previous works on DNA computing and formal languages [6, 7], where iterated transductions were proved to characterize the entire class of recursively enumerable languages.

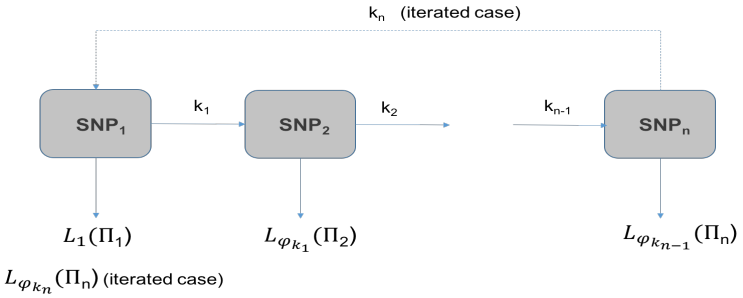

Fig. 3. A network of SN P systems generates a family of languages.

6 Final comments and future research

The idea of associating a family of languages with a given P system is rather natural. We have illustrated it here with the case of SN P systems, but the same strategy can be applied for any type of P systems producing a language (such that cell-like P systems with external output, SN P systems generating trace languages [2], etc.).

A more systematic study of this idea is of interest, starting with relevant examples, continuing with "standard" formal language theory questions, and ending with possible applications of this approach (as languages generated by the same P system are "genetically" related, maybe in this way one can capture biological connections/dependencies or other types of relationships).

More precisely, we enumerate the following questions related to our proposal:

1. We have described a way to encode languages within SN P systems. Now, the reverse problem arises i.e., to decode languages from the spike train. Here, from a spike train we should obtain the set of binary spike trains that encode it. This issue should be studied in order to complete a classical communication framework.
2. With respect to the encoding properties, we have overviewed only the aspects related to the (non)injective property. Different properties from code theory should produce new results that connect formal language theory, SN P systems and communications systems.
3. The last issue that we have proposed opens different problems related to it. If a network of SN P systems is proposed then we should study the effects of the network topology and the number of SN P systems over the families of languages. In this sense, the number of SN P system could be considered a descriptional complexity measure.

These aspects and new ones will be reported in future works.

Acknowledgements

Part of this work appeared as *Families of Languages Associated with SN P Systems: Preliminary Ideas, Open Problems*. Gh. Păun, J.M. Sempere. *Bulletin of the Membrane Computing Society*, Issue 2, December 2016, pp. 161-164. http://membranecomputing.net/IMCSBulletin/. The author is indebted to Gh. Păun for his original contribution to this work.

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez. On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), pp. 141-162.
2. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems, *Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, pp. 94–105.
3. E. Csuhaj-Varjú, G. Vaszil: On counter machines versus dP automata. *Membrane Computing. 14th Intern. Conf., CMC 2013, Chişinău, August 2013* (A. Alhazov et al., eds.), LNCS 8340, Springer, (2014), pp. 138–150.
4. O.H. Ibarra, A. Leporati, A. Păun, S. Woodworth: Spiking neural P Systems, in *The Oxford Handbook of Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2010.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), pp. 279–308.
6. V. Manca: On the generative power of iterated transduction. In *Words, Semigroups, & Transductions* (M. Ito, Gh. Păun, S. Yu, eds.), pp. 315-327. World Scientific, 2001.
7. V. Manca, C. Martín-Vide, Gh. Păun: New computing paradigms suggested by DNA computing: computing by carving. *BioSystems*, 52 (1999), pp. 47-54.
8. Gh. Păun. Membrane Computing. An Introduction, Springer, 2002.
9. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg. Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science*, 17, 4 (2006), pp. 975-1002.
10. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 vols., Springer-Verlag, 1997.