POLYNOMIAL TIME-BOUNDED COMPUTATIONS IN SPIKING NEURAL P SYSTEMS

Petr Sosík[*†], *Alfonso Rodríguez-Patón*[*], *Lucie Ciencialová*[†]

Abstract: The paper introduces a formal framework for the study of computational power of spiking neural (SN) P systems. We define complexity classes of uniform families of recognizer SN P systems with and without input, in a way which is standard in P systems theory. Then we study properties of the resulting complexity classes, extending previous results on SN P systems. We demonstrate that the computational power of several variants of confluent SN P systems, under polynomial time restriction, is characterized by classes ranging from **P** to **PSPACE**.

Key words: *Membrane computing, spiking neural P system, complexity class*

Received: July 15, 2012
Revised and accepted: January 29, 2013

1. Introduction

Spiking neural P system (abbreviated as SN P system) introduced in [7] is an abstract computing model inspired by the theory of membrane computing, on one hand, and spiking neural networks, on the other hand. The structure of an SN P system is formed by membrane cells (called neurons) arranged in nodes of a directed graph whose arcs represent synapses. Neurons contain spikes which are represented as copies of a specified symbol, most often a. Each neuron has its own rules for either sending spikes to all neurons linked by synapses to the emitting neuron (firing rules optionally with delays) or for internally consuming spikes (forgetting rules). One neuron is designed as input neuron and another one as output neuron. Computation of SN P system starts from an initial configuration, in which an

[*]Petr Sosík, Alfonso Rodríguez-Patón
Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain, E-mail: {psosik,arpaton}@fi.upm.es

[†]Petr Sosík, Lucie Ciencialová
Research Institute of the IT4Innovations Centre of Excellence, Faculty of Philosophy and Science, Silesian University in Opava, 74601 Opava, Czech Republic, E-mail: lucie.ciencialova@fpf.slu.cz

initial number of spikes is placed in each neuron. At each moment (the system is synchronized by means of a global clock), a computational step is executed by applying a rule in each neuron in which this is possible. The applicability of a rule is determined by a regular expression associated with it, against which the number of spikes occurring in the neuron is checked. If the computation halts (no neuron has an applicable rule), then the result of the computation is obtained from the output neuron.

The computational power of SN P system has been extensively studied and several intractable problems such as Subset Sum, SAT, QSAT and others have been shown effectively solvable by SN P systems under certain conditions ([8, 9, 10, 12]).

Standard SN P systems were shown to be computationally universal already in the introductory paper [7]. However, as demonstrated in [13], no standard spiking neural P system with a constant number of neurons can simulate Turing machines with less than exponential time and space overheads. This is due to the unary character of its unlimited memory - spikes accumulated in neurons. Therefore, to achieve computational effectiveness in solving problems, many authors have used families of SN P systems such that each member of a family solves only a finite set of instances of a given size. Another way how to effectively solve NP-complete problems is to use SN P systems with neuron division ([16]).

In this paper we establish a formal framework based on computational complexity theory, allowing to characterize the computational power of SN P systems more precisely. This framework allows to standardize, compare and extend the results already mentioned in the literature. A sequence of formal prerequisites is established in Section 3 which then allow to define polynomially uniform families of *confluent* SN P systems in Section 4 and to study their properties. Confluent SN P systems are generally non-deterministic but each system with a given input has either only rejecting computations or only accepting computations.

Then we focus on uniform families of SN P systems. We show the closure of their polynomial time-restricted complexity classes under complement and polynomial time reduction. Finally we provide a characterization or limitation of several variants of uniform families of recognizer SN P system in Section 5. We also mention the differences between unary and binary encoding and study their influence on the presented results. We show that some restricted variants of regular expressions (including the single star normal form) allow to characterize the class **P**, while in general their computational power lies between the classes **NP**, **co-NP** and **PSPACE**.

2. Prerequisites

In this section we recall some useful notations and constructions used throughout the paper. We assume the reader to be familiar with basic language and automata theory, as well as with elements of the computational complexity theory (see for example [5]). We also refer to [14] for up-to-date information about membrane computing.

We denote by $\mathbb{N}$ the set of nonnegative integers. For a finite alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted

by λ, and the set of all nonempty finite strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The length of a string $x \in V^*$ is denoted by $|x|$. Next we recall the definition of regular expression.

Definition 1 *For a finite alphabet V : (i) λ and each $a \in V$ are regular expressions, (ii) if E_1, E_2 are regular expressions over V, then also $(E_1) \cup (E_2), (E_1) \cdot (E_2)$, and $(E_1)^*$ are regular expressions over V, and (iii) nothing else is a regular expression over V.*

The catenation operator $\cdot$ and non-necessary parentheses may be omitted when writing a regular expression. With each expression E we associate its language $L(E)$ defined in a usual way: (i) $L(\lambda) = \{\lambda\}$, $L(a) = \{a\}$ for each $a \in V$, (ii) if E_1, E_2 are regular expressions over V, then $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1) \cdot (E_2)) = L(E_1) \cdot L(E_2)$, and $L((E_1)^*) = L(E_1)^*$.

We call two expressions E_1 and E_2 *equivalent* if $L(E_1) = L(E_2)$.

Definition 2 ([1]) *We say that a regular expression $E = E_1 \cup \ldots \cup E_n$ (where each E_i contains only $\cdot$ and $*$ operators) is in* single-star normal form *(SSNF) if $\forall i \in \{1, \ldots, n\}$, E_i has at most one occurrence of $*$.*

Lemma 1 ([1]) *Every regular expression over one-letter alphabet can be transformed into an equivalent single-star normal form.*

This transformation, however, might require an exponential time and the size of the resulting expression can be exponential with respect to the size of the original expression.

3. Spiking Neural P Systems

A *spiking neural P system* of degree $m \geq 1$ is a construct of the form $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out)$, 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) $n_i \in \mathbb{N}$ 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) $E/a^c \to a; d$, where E is a regular expression over a specific for each rule, $c \geq 1$ and $d \geq 0$ are integers;

(2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);

4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input neuron* (resp., *output neuron*).

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains 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 consumes (removes) c spikes (thus only $k - c$ remain in σ_i), the neuron is fired and it produces a spike after d time units (as usual in membrane computing, a global clock is assumed, 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 $t + d + 1$).

The rules of type (2) are *forgetting* rules; 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.

Remark. If a rule $E/a^c \rightarrow a; d$ of type (1) has $E = a^c$, then we will write it in the following 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. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

Remark. A generalization of SN P systems, introduced in [3], is to use *extended rules*, of the form $E/a^i \rightarrow a^p; d$, with $p \geq 0$. Such a rule operates in the same manner as before except that firing sends p spikes along each outgoing synapse.

During a computation, a *configuration* of the system is described by both the number of spikes present in each neuron and the number of steps remaining until the neuron becomes open (if the neuron is already open, the number is 0). The configuration $\langle n_1/t_1, \ldots, n_m/t_m \rangle$ is the configuration where neuron σ_i contains $n_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, for $i = 1, 2, \ldots, m$. An *initial configuration* adopts always the form $\langle n_1/0, \ldots, n_m/0 \rangle$, with all neurons being open.

Using the rules described above, one can define a transition between two configurations C_1, C_2 and denote it 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. With any computation (halting or not) we associate a *spike train*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes in a particular time step, then we write 1, otherwise we write 0.

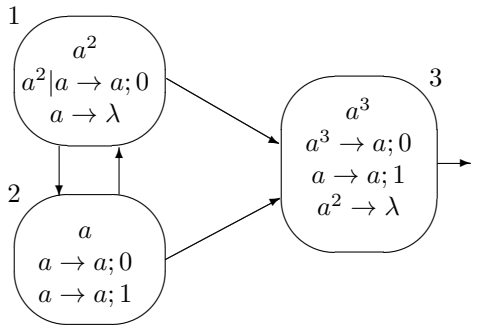

Fig. 1 *An SN P system generating all natural numbers greater than 1.*

An *output* of a given computation of an SN P system is defined as the spike train (i.e., a binary string) leaving the output neuron during the computation. Alternatively, many authors consider as output the time interval (i.e., an integer) between two consecutive spikes. See Section 3.1 or [6] for more details.

Similarly, an *input* of a given computation of an SN P system is defined as an external spike train entering the input neuron, starting from the beginning of the computation.

SN P systems can be used in the *generating* mode when no input neuron and no input is considered. The system non-deterministically generates during various computations a (possibly infinite) set of output values. In the *accepting* mode, an input is provided and the system indicates during its computation (possibly by the behavior of the output neuron) whether the input was accepted or not, similarly as a formal automaton. More details are explained in Section 3.2.

Example 1 ([6]) Consider the following generating SN P system shown at Fig.1:

$$\begin{aligned}
\Pi &= (O, \sigma_1, \sigma_2, \sigma_3, syn, out)\,, \\
O &= \{a\}\,, \\
\sigma_1 &= \left(2, \left\{a^2/a \to a;0,\ a \to \lambda\right\}\right), \\
\sigma_2 &= (1, \{a \to a;0,\ a \to a;1\})\,, \\
\sigma_3 &= \left(3, \left\{a^3 \to a;0,\ a \to a;1,\ a^2 \to \lambda\right\}\right), \\
syn &= \{(1,2)\,,(2,1)\,,(1,3)\,,(2,3)\}\,, \\
out &= 3
\end{aligned}$$

In the initial configuration $\langle 2/0, 1/0, 3/0\rangle$ all the neurons can fire. The neuron σ_1 can fire only if it has two spikes inside and after firing one spike remains available for the next step. There is one spike and two applicable firing rules in the neuron σ_2. The rules differ only in delay. The neuron σ_3 contains three spikes and in the first step they are consumed by an application of rule.

In the next steps both neurons σ_1 and σ_2 send a spike to each other and to the output neuron σ_3. Two spikes are forgotten in the next step as long as the neuron σ_2 uses the rule with delay 0. The configuration after each of these steps is $\langle 2/0, 1/0, 2/0\rangle$.

If the neuron σ_2 uses the rule with delay 1, then σ_2 is closed in this step and cannot obtain a spike from the neuron σ_1. The neuron σ_2 does not send the spike either to σ_1 or σ_3. The reached configuration is $\langle 1/0, 0/1, 1/0\rangle$.

In the next step the neuron σ_1 uses a forgetting rule and the neuron σ_3 fires. Simultaneously, the neuron σ_2 sends a spike to σ_1 and σ_3 but σ_3 is closed in this step because of the firing rule $a \to a; 1$. The system enters the configuration $\langle 1/0, 0/0, 0/1\rangle$. In the next step the neuron σ_3 spikes, the spike in σ_1 is forgotten and the system halts. The system reaches the final configuration $\langle 0/0, 0/0, 0/0\rangle$.

Because of delay in the firing rule $a \to a; 1$ of neuron σ_3, the two spikes emitted from this neuron cannot come immediately consecutive. When the neuron σ_2 uses the rule $a \to a; 1$ in the first step, we get the shortest possible output spike train 101, representing the output value 2. Thus, the SN P system Π generates all natural numbers greater than 1.

3.1 Input/output convention: unary versus binary

Original works on SN P systems as, e.g., [7, 17] focused on SN P systems working in the generating mode where the output value was represented as a time interval between two spikes of the output neuron. This means that the output values were represented in *unary*, and similarly were later treated also input values [10].

Unary input/output encoding
A natural number n is represented as a spike train $10^{n-1}1$.

In the case of SN P systems computing functions or solving decision problems, the binary encoding may be a better choice:

Binary input/output encoding
A natural number n is represented as a spike train $b_k b_{k-1} \dots b_0$, where $b_k b_{k-1} \dots b_0$ is the binary representation of n.

We use the binary encoding in the rest of the paper. One should be aware that, to switch from binary to unary encoding, a time exponential to the size of the original binary string is needed, unless an extended SN P system with maximal parallelism is used ([12]).

Remark. The following facts justify our choice of binary encoding:

- Standard SN P systems can simulate logic gates AND, OR, XOR with an unbounded fan-in in a unit time and, hence, they can simulate also arbitrary logic circuits in linear time [4]. An acyclic circuit of a depth n with a single input can process an m-bit information in time $\mathcal{O}(m+n)$ [21]. Therefore, so can an equivalent SN P system with binary input.

  On the contrary, a P system with unary input/output convention would need the time $\Omega(2^m)$ just to read the input spike train of the length 2^m encoding m bits of information.

- When dealing with SN P systems *without input*, one can encode an n-bit value into a neuronal structure of size $\mathcal{O}(n)$ and the information can be then processed in time $\mathcal{O}(n)$ (see, e.g., [9]). In an SN P system with the

input neuron, the unary input convention implies the processing time $\Omega(2^n)$. Hence, the time complexity of a uniform solution to a given problem could be exponentially higher than that of the non-uniform solution.

Therefore, some recent papers dealing with SN P systems solving NP-complete problems as, e.g., [10, 16, 22] adopt the binary input convention. Actually, the input convention in [16, 22] is even ternary as the input neuron can, in each step, receive 0, 1 or 2 spikes.

3.2 Recognizer SN P systems

In this subsection we define SN P systems solving decision problems. Let us call *decision problem* a pair $X = (I_X, \theta_X)$ where I_X is a language over a finite alphabet (elements of I_X are called instances) and θ_X is a total Boolean function over I_X. The following convention was suggested by some authors: an SN P systems solving an instance $w \in I_X$ would halt if and only if $\theta_X(w) = 1$. Such an SN P system is called *accepting* in [10]. However, this convention was rarely implemented. Instead, many authors demonstrated SN P systems which always halt and the output neuron spikes if and only if $\theta_X(w) = 1$, see [4, 9, 10, 11, 12] and others. This convention is also more compatible with definitions of standard complexity classes of recognizer P systems [18]. We suggest the following definition:

Definition 3 *A* recognizer SN P system *satisfies the following conditions: all computations are halting, and the output neuron spikes no more than once during each computation. The computation is called* accepting *if the output neuron spikes exactly once, otherwise it is* rejecting.

Observe that the definition is compatible with the variant when the system is asked to spike *at least once* in the case of accepting computation. To any such SN P system we can add another neuron connected to the original output, with two initial spikes and the rules $a \to \lambda$ and $a^3 \to a; 0$ which emits only the first spike of those received.

Actually, the difference between the halting and spiking convention is not so great. The following result is demonstrated in [10].

Lemma 2 *Given an SN P system* Π *with standard or extended rules, with or without delays, we can construct an SN P system* Π' *with rules of the same kinds as those of* Π, *such that the output neuron of* Π' *spikes if and only if* Π *halts.*

Note that Π' does not have to halt if Π does not halt. In the other direction, we can extend Lemma 2 as follows:

Lemma 3 *Given a system* Π *with standard or extended rules, with or without delays, we can construct a system* Π' *with rules of the same kinds as those of* Π *which halts if and only if the output neuron of* Π *spikes.*

Proof. The proof technique is inspired by [10]: consider an SN P system Π (possibly with extended rules), and let σ_{out} be its output neuron. Assume, without loss of generality, that σ_{out} has no outgoing synapses. We construct an SN P system Π' as follows:

1. We "triple" the system Π by:

   - tripling the number of spikes present in the initial configuration in each neuron,
   - replacing each rule $E/a^c \to a^p; d$ with $3E/a^{3c} \to a^p; d$, where $3E$ is a regular expression for the set $\{www \mid w \in L(E)\}$,
   - tripling each neuron σ_c except for σ_{out} : adding two identical neurons $\sigma_{c'}$, $\sigma_{c''}$ and adding new synapses: if σ_c had originally a synapse to a neuron γ, now each of σ_c, $\sigma_{c'}$ and $\sigma_{c''}$ will have synapses to each of γ, γ' and γ''.

   Let us denote the "tripled" system by 3Π.

2. We add to each neuron of 3Π the rule $(aaa)^*aa/a \to a$.

3. Let d_{max} denote the maximum delay used in any spiking rule appearing in Π. We add to 3Π a module described in Fig. 2, where the neuron σ_0 has an incoming synapse from σ_{out} and σ_{halt} has an outgoing synapse to each neuron of 3Π.

The resulting system is denoted by Π′. Observe that each neuron in 3Π spikes if and only if its original version spikes in Π. If a neuron σ_c in Π contains n spikes in a certain moment, then each of σ_c, $\sigma_{c'}$ and $\sigma_{c''}$ in 3Π contains $3n$ spikes.

⇐ Suppose that the output neuron σ_{out} of Π spikes. Then the circuit in Fig. 2 produces $d_{max}+1$ consecutive spikes. Each neuron of 3Π receives a spike, its accumulated number spikes increases to $3n+1$, $n \geq 0$, and none of its original rules can be applied. When receiving more spikes from σ_{halt}, neurons apply the rule$(aaa)^*aa/a \to a$ to contain again $3n+1$ spikes. The spikes produced by the rule $(aaa)^*aa/a \to a$ circulate in the system 3Π but, as all the neurons are triplicated, each neuron can receive $3k$ of these spikes, for some integer $k \geq 0$, and hence it will contain $3(n+k)+1$ spikes. Observe also that the neuron σ_0 in Fig. 2 spikes only once when σ_{out} spikes for the first time. Eventual subsequent spikes received from the neuron σ_{out} are consumed by the rule $a \to \lambda$.

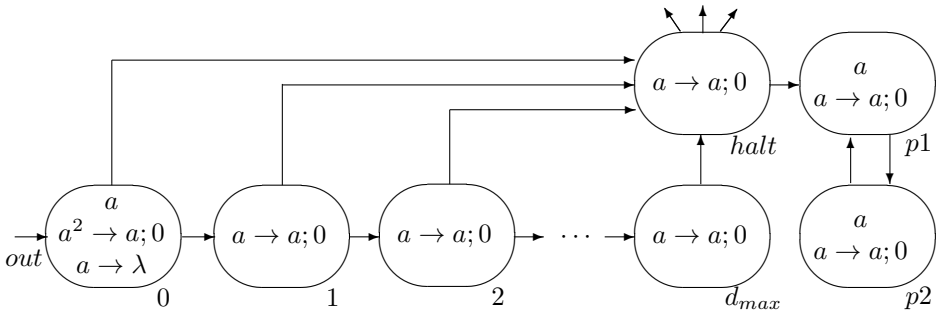

Fig. 2 *A module emitting $d_{max}+1$ consecutive spikes after the output neuron spikes.*

When the neuron σ_{out} spikes, some of the neurons of 3Π may be closed due to their refractory periods. This period would end in d_{max} steps during which the neuron σ_{halt} keeps emitting spikes. Hence, after this period, each neuron of 3Π will keep $3n+1$ spikes, for some $n \geq 0$, and cannot spike any more.

Finally, consider neurons σ_{p1} and σ_{p2} in Fig. 2 which spike at each step since the beginning of computation, until the neuron σ_{halt} spikes. When receiving a spike from σ_{halt} and σ_{p2} simultaneously, the neuron σ_{p1} contains more than one spike and cannot spike any more. In the next step the neuron σ_{p2} becomes empty and spikes no more. Hence the whole system Π' halts.

$\Rightarrow$ Suppose that Π' halts, which implies that also the neurons σ_{p1} and σ_{p2} stop spiking. As explained in the previous paragraph, this happens only if the neuron σ_{halt} spikes. The neuron σ_{halt} can spike only if it receives a spike from some of neurons $\sigma_0, \ldots, \sigma_{d_{max}}$ in Fig. 2, and this happens only if the neuron σ_0 receives a spike from σ_{out}. Therefore, the output neuron σ_{out} of Π spikes.

□

3.3 Descriptional complexity versus size of SN P systems

In the next sections we deal with uniform families of SN P systems, so we need to specify the size of each member of a family. According to [11], the size of an SN P system Π is based on the number of bits necessary for its full description. Let m be the number of neurons, N be the maximum natural number that appears in the definition of Π – the maximum of the number of the neurons, the number and the length of the rules, the number of synapses. Further, let R be the maximum number of rules which occur in its neurons, and S be the maximum size required by the regular expressions in succinct form that occur in Π. (The succinct form means that an expression a^n is represented just by $\mathcal{O}(\log n)$ bits.) Then the total size of description of Π is polynomial with respect to m, R, S and $\log N$.

Some authors [6, 10, 12] distinguish between the size of an SN P system and the size of its description. They point out that the initial number of spikes in neurons or the length of (unary) strings in regular expressions can be exponential with respect to the size of description of the system. They conclude that an exponential time might be needed to construct such an SN P system. However, spikes in neurons correspond to an electric potential and not physical objects. Also in recent implementations *in silico* the number of spikes is represented as a binary value. Then one does not need an exponential time to construct such an SN P system. Hence, we assume the size of an SN P system and the size of its description the same.

4. Families of Recognizer SN P Systems

In this section we propose a formal specification for families of SN P systems. All definitions in this section are inspired by [18] which studies families of P systems working with objects.

Definition 4 *A family* $\mathbf{\Pi} = \{\Pi(w)|\ w \in I_X\}$ *(respectively,* $\mathbf{\Pi} = \{\Pi(n)|\ n \in \mathbb{N}\}$*) of recognizer SN P systems without input (resp., with input) is* polynomially uniform by Turing machines *for any instance* $w \in I_X$ *(resp.,* $n \in \mathbb{N}$*) if there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *(resp.,* $\Pi(n)$*).*

In the sequel we will denote such a family simply as *uniform.* The selection of a proper member of the family and its input in the case of families of SN P systems with input is done as follows.

Definition 5 *Let* $X = (I_X, \theta_X)$ *be a decision problem, and* $\mathbf{\Pi} = \{\Pi(n)|\ n \in \mathbb{N}\}$ *a family of recognizer SN P systems with input membrane. A polynomial encoding of* X *in* $\mathbf{\Pi}$ *is a pair* $(cod; s)$ *of polynomial-time computable functions over* I_X *such that, for each instance* $w \in I_X$, $cod(w)$ *is a binary string – an input of the system* $\Pi(s(w))$ *and* $s(w)$ *is a natural number (obtained by means of a reasonable encoding scheme).*

The common case is that $s(w)$ is the length of the instance $w \in I_X$ (recall that each instance is a string over a finite alphabet). As Definitions 4 and 5 conform those in [18, 19], we can adopt the following result whose proof in [19] is not affected by a different type of P system.

Lemma 4 *Let* X_1, X_2 *be decision problems,* r *a polynomial-time reduction from* X_1 *to* X_2*, and* $(cod; s)$ *a polynomial encoding of* X_2 *in* $\mathbf{\Pi}$*. Then,* $(cod \circ r; s \circ r)$ *is a polynomial encoding of* X_1 *in* $\mathbf{\Pi}$*.*

Let $\mathcal{R}$ denote an arbitrary type of recognizer SN P systems. To describe a specific type $\mathcal{R}$ of SN P systems, we denote:

$-reg$ for systems with regular expressions restricted to the form a^n, $n \geq 1$,

$-del$ for systems without delays,

$ssnf$ for systems with regular expressions in the single-star normal form.

The following definitions are inspired by [18] and [10].

Definition 6 *Let* $f : \mathbb{N} \to \mathbb{N}$ *be a Turing-constructible function. We denote by* $\mathbf{SN}^*_{\mathcal{R}}(f)$ *the class of decision problems solvable by a family of recognizer SN P systems of type* $\mathcal{R}$ *without input in time bounded by* f*. A problem* X *is in* $\mathbf{SN}^*_{\mathcal{R}}(f)$ *if a family* $\mathbf{\Pi} = \{\Pi(w)|\ w \in I_X\}$ *exists such that:*

- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines.*
- *The family* $\mathbf{\Pi}$ *is* f*-bounded with respect to* X*; that is, for each instance* $w \in I_X$*, every computation of* $\Pi(w)$ *performs at most* $f(|w|)$ *steps.*
- *The family* $\mathbf{\Pi}$ *is sound with respect to* X*; that is, for each* $w \in I_X$*, if there exists an accepting computation of* $\Pi(w)$*, then* $\theta_X(w) = 1$*.*
- *The family* $\mathbf{\Pi}$ *is complete with respect to* X*; that is, for each* $w \in I_X$*, if* $\theta_X(w) = 1$*, then every computation of* $\Pi(w)$ *is an accepting computation.*

Note that a recognizer SN P system solving an instance w due to the above definition can have different possible computations but with the same result. Such an SN P system is also called *confluent.* Obviously, not all recognizer SN P systems are confluent. The family $\mathbf{\Pi}$ is said to provide a *semi-uniform solution* to the problem X. In this case, for each instance of X we have a special SN P system. Specifically, we denote by

$$\mathbf{PSN}^*_{\mathcal{R}} = \bigcup_{f \text{ polynomial}} \mathbf{SN}^*_{\mathcal{R}}(f)$$

the class of problems to which uniform families of SN P systems of type $\mathcal{R}$ without input provide semi-uniform solution in polynomial time. Analogously, we define families which provide uniform solutions to decision problems.

Definition 7 *Let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function. We denote by $\mathbf{SN}_{\mathcal{R}}(f)$ the class of decision problems solvable by a family of recognizer SN P systems of type $\mathcal{R}$ with input in time bounded by f. A problem X is in $\mathbf{SN}_{\mathcal{R}}(f)$ if a family $\mathbf{\Pi} = \{\Pi(n) |\ n \in \mathbb{N}\}$ exists such that:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *There exists a polynomial encoding (cod, s) of X in $\mathbf{\Pi}$ such that:*
  - *The family $\mathbf{\Pi}$ is f-bounded with respect to X; that is, for each instance $w \in I_X$, every computation of $\Pi(s(w))$ with input $cod(w)$ performs at most $f(|w|)$ steps.*
  - *The family $\mathbf{\Pi}$ is sound with respect to (X, cod, s); that is, for each $w \in I_X$, if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$, then $\theta_X(w) = 1$.*
  - *The family $\mathbf{\Pi}$ is complete with respect to (X, cod, s); that is, for each $w \in I_X$, if $\theta_X(w) = 1$, then every computation of $\Pi(s(w))$ with input $cod(w)$ is an accepting computation.*

The family $\mathbf{\Pi}$ is said to provide a *uniform solution* to the problem X. Again, we denote by

$$\mathbf{PSN}_{\mathcal{R}} = \bigcup_{f \text{ polynomial}} \mathbf{SN}_{\mathcal{R}}(f)$$

the class of problems to which uniform families of SN P systems of type $\mathcal{R}$ with input provide uniform solution in polynomial time. When $\mathcal{R}$ is omitted, the standard definition of SN P systems is assumed.

Observe that, when $X \in \mathbf{SN}^*_{\mathcal{R}}(f)$, each instance $w \in I_X$ is solved by a specific SN P system $\Pi(w)$ and the description of w is encoded in the structure of $\Pi(w)$, on one hand. On the other hand, when $X \in \mathbf{SN}_{\mathcal{R}}(f)$, then all the instances $w \in I_X$ of the same length $n = s(w)$ are solved by the same SN P system $\Pi(n)$ and their encodings $cod(w)$ are sent as spike trains to the input neuron of $\Pi(n)$. Given a particular instance w, it is very easy to add to $\Pi(n)$ a set of neurons which would produce the spike train $cod(w)$ so that the input is not needed. Therefore, for any constructible function f and a class of SN P systems $\mathcal{R}$ we have

$$\mathbf{SN}_{\mathcal{R}}(f) \subseteq \mathbf{SN}^*_{\mathcal{R}}(f) \quad \text{and} \quad \mathbf{PSN}_{\mathcal{R}} \subseteq \mathbf{PSN}^*_{\mathcal{R}}.$$

Theorem 1 *The classes* $\mathbf{SN}_{\mathcal{R}}(f)$ *and* $\mathbf{SN}^{*}_{\mathcal{R}}(f)$ *are closed under the operation of complement, for* $\mathcal{R}$ *omitted or* $\mathcal{R} \in \{-reg, -del, ssnf\}$.

Proof. It is necessary to show that for each confluent SN P system Π there exists a system Π' whose computation is accepting if and only if the computation of Π is rejecting. Assume the construction described in Section 4, Fig. 6 in [10]. It presents a module which, when added to any SN P system Π, emits a spike only after the system Π halts. The provided construction works for many variants of SN P systems (i.e., with or without delay, with simplified regular expressions restricted to the form λ or a^*, and also for SN P systems with extended rules).

Note that this module contains a set of rules $a^k \to a; 0$ for all $k \in K$, where K is the set constructed as follows. For each neuron σ_i of Π, $1 \leq i \leq n$ (where n is the degree of Π) denote

$$P_i = \bigcup_{1 \leq i \leq n} \{p \,|\, E/a^c \to a^p; d \text{ is a rule of } \sigma_i\},$$

and

$$K = \left\{ \sum_{i=1}^{n} p_i \,|\, p_i \in P_i \right\} - \{0\}. \tag{1}$$

Hence K contains sums of all possible n-tuples containing one element of each P_i, hence the number of these n-tuples may be exponential with respect to n. However, in such a case many of these sums will be equal. Let

$$p_{\max} = \max\{p \,|\, E/a^c \to a^p; d \text{ is a rule of } \sigma_i\},$$

then each sum on the right-hand side of (1) will be bounded by $np_{\max}$. Therefore,

$$K \subseteq \{1, 2, \ldots, np_{\max}\}$$

and hence the size of K is linear with respect to n. (However, in the case of extended SN P systems with $p_{\max} \gg n$, the size of K could be exponentially greater than the size of Π which is polynomial with respect to $n \log p_{\max}$, see Section 3.3).

Let us extend the module in Fig. 6 in [10] as follows. Denote σ_{halt} the output neuron of this module. Let a spike emitted from σ_{halt} after halting of the system Π feed two new neurons, each with a rule $a \to a; 0$. Finally, add a new neuron $\sigma_{out'}$ with incoming synapses from these two neurons, another synapse from σ_{out}, i.e., the original output neuron of Π, and with a rule $a^2 \to a; 0$. The resulting module is displayed in Fig. 3, where D is a maximum delay in any of the rules of Π. Let $\sigma_{out'}$ be the output neuron of Π'. Note that $\sigma_{out'}$ spikes if and only if Π halts and its output neuron σ_{out} does not spike which concludes the proof. □

Note that the above proof also holds for a certain subclass of extended SN P systems with $p_{\max}$ bounded from above by $poly(n)$. This condition guarantees that the size of the complementary system is polynomial with respect to the size of the original system, hence the family remains polynomially uniform by Turing machines. It is an open problem whether an analogous result holds for unrestricted extended SN P systems.

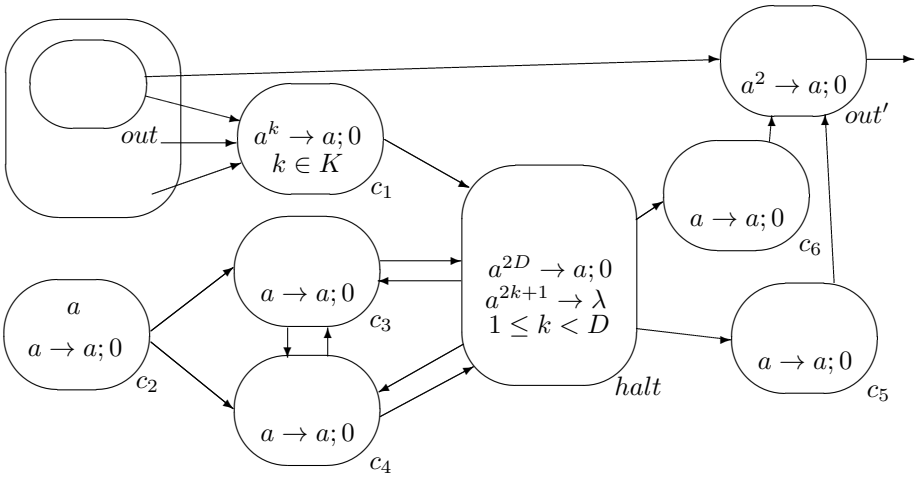

Fig. 3 *A module emitting a spike if and only if the system* Π *halts and its output neuron* σ_{out} *does* not *spike.*

Corollary 1 *The classes* $\mathbf{PSN}_{\mathcal{R}}$ *and* $\mathbf{PSN}^*_{\mathcal{R}}$ *are closed under the operation of complement, for* $\mathcal{R}$ *omitted or* $\mathcal{R} \in \{-reg, -del, ssnf\}$.

Theorem 2 *Let* $\mathcal{R}$ *be an arbitrary class of SN P systems, i.e.,* $\mathcal{R} \in \{-reg, -del, ssnf\}$ *or* $\mathcal{R}$ *omitted. Let* X *and* Y *be decision problems such that* X *is reducible to* Y *in polynomial time. If* $Y \in \mathbf{PSN}_{\mathcal{R}}$ *(respectively,* $Y \in \mathbf{PSN}^*_{\mathcal{R}}$*), then* $X \in \mathbf{PSN}_{\mathcal{R}}$ *(resp.,* $X \in \mathbf{PSN}^*_{\mathcal{R}}$*).*

Proof. We prove the case of SN P systems with input, adopting the technique used in [19], the case without input is analogous. Let $\mathbf{\Pi}$ be a family providing uniform solution to the problem Y. By its definition, let p be a polynomial and (cod, s) a polynomial encoding of Y in $\mathbf{\Pi}$ such that $\mathbf{\Pi}$ is p-bounded with respect to Y and sound and complete with respect to (Y, cod, s).

Let $r : I_X \to I_Y$ be a polynomial time reduction from X to Y, hence there is a polynomial q such that for each $w \in I_X$, $|r(w)| \le q(|w|)$. Observe that:

- By Lemma 4, $(cod \circ r; s \circ r)$ is a polynomial encoding of X in $\mathbf{\Pi}$.
- $\mathbf{\Pi}$ is $(p \circ q)$-bounded with respect to X since for each $w \in I_X$, every computation of $\Pi(s(r(w)))$ with input $cod(r(w))$ performs at most $p(|r(w)|) \le p(q(|w|))$ steps.
- $\mathbf{\Pi}$ is sound and complete with respect to $(X, cod \circ r, s \circ r)$ since for each $w \in I_X$,
  - if there exists an accepting computation of $\Pi(s(r(w)))$ with input $cod(r(w))$, then $\theta_Y(r(w)) = 1$ and, by reduction, also $\theta_X(w) = 1$,
  - if $\theta_X(w) = 1$, then also $\theta_Y(r(w)) = 1$ and hence every computation of $\Pi(s(r(w)))$ with input $cod(r(w))$ is an accepting computation.

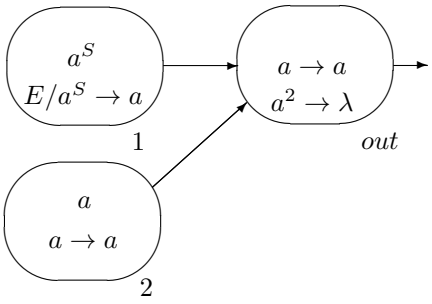

Fig. 4 *A standard deterministic SN P system solving the problem co-*Subset Sum *in two steps. The regular expression E is derived from a particular case of* co-Subset Sum.

Consequently, $X \in \mathbf{SN}_{\mathcal{R}}(p \circ q)$ and hence also in $\mathbf{PSN}_{\mathcal{R}}$. □

5. Efficiency of Basic Classes of SN P Systems

As we have already mentioned, no standard SN P system can simulate Turing machine with less than exponential time and space overheads ([13]). Therefore, we focus on families of recognizer SN P systems in this section. We start with a simple variant of SN P systems with restrictions imposed on their regular expressions. Results in [11, 20] imply that their computational power is equivalent to **P**.

Theorem 3 $\mathbf{PSN}_{-reg,-del} = \mathbf{PSN}^*_{-reg,-del} = \mathbf{PSN}_{ssnf} = \mathbf{PSN}^*_{ssnf} = \mathbf{P}$.

Proof. By Theorem 9 in [20], for each confluent (resp. non-confluent) SN P system Π with all regular expressions in single-star normal form and with description of size s, there is a deterministic (resp. non-deterministic) Random Access Machine (RAM) constructed in polynomial time with unit cost of operations which simulates t steps of Π in time $\mathcal{O}(t(t+s))$. Due to a polynomial number of steps of computation for $\mathbf{PSN}_{ssnf}$ and $\mathbf{PSN}^*_{ssnf}$ we can write $\mathbf{PSN}_{ssnf} \subseteq \mathbf{PSN}^*_{ssnf} \subseteq \mathbf{P}$.

Furthermore, let $\mathbf{CRCW}(poly(n), T(n))$ be the class of problems solved by a CRCW PRAM (Concurrent Read Concurrent Write Parallel RAM, for definitions see, e.g., [2]) with a polynomial number of processors in time $T(n)$. By Theorem 14 in [20], $\mathbf{CRCW}(poly(n), T(n)) \subseteq \mathbf{SN}_{-reg,-del}(T(n))$ for an arbitrary polynomial T. From the observation that $\mathbf{CRCW}(poly(n), poly(n)) = \mathbf{P}$ we obtain that $\mathbf{P} \subseteq \mathbf{PSN}_{-reg,-del}$. Finally, by definition, $\mathbf{PSN}_{-reg,-del} \subseteq \mathbf{PSN}^*_{-reg,-del}$, $\mathbf{PSN}_{-reg,-del} \subseteq \mathbf{PSN}_{ssnf}$ and $\mathbf{PSN}^*_{-reg,-del} \subseteq \mathbf{PSN}^*_{ssnf}$, and we get the statement of the theorem. □

These results show that families of standard confluent SN P systems can reach the computational power beyond **P** only with the aid of complex regular expressions. Whenever we release the condition of single star normal forms in regular expressions, the computational power of SN P systems reaches the class **NP**.

In the proof of the next Theorem 4 we refer to Proposition 1 in [11] which is recalled here:

Proposition 1 ([11]) *Let* Π *be an SN P system having a single neuron that contains a spiking rule* $E/a^c \to a; d$. *If* E *is described in a succinct form, then deciding whether this rule can be applied is at least* **NP***-hard.*

Recall that the succinct form means that an expression a^n is represented by $\mathcal{O}(\log n)$ bits. We refer the reader to [15], Chapter 20 for more details.

Theorem 4 *Let regular expressions and initial numbers of spikes in neurons in SN P systems be described in succinct form. Then* $(\mathbf{NP} \cup \mathbf{co\text{-}NP}) \subseteq \mathbf{SN}^*_{-del}(2)$.

Proof. A part of the statement concerning **NP** follows by the proof of Proposition 1 in [11] which presents a construction of standard deterministic SN P system without delays solving the NP-complete problem Subset Sum in one step. Each instance of Subset Sum is solved by a specific SN P system, forming a family of recognizer SN P systems without input. By the polynomial time reducibility, also any other NP-complete problem can be solved by such a family.

By Theorem 1, also the complementary problem co-Subset Sum (which is co-NP-complete) can be solved in the same way. Actually, in this case it is enough to add two more neurons which add one more step of computation. Let $V = (\{v_1, v_2, \ldots, v_n\}, S)$ be an instance of co-Subset Sum. The corresponding SN P system is illustrated in Fig. 4, where the regular expression E adopts the form $E = (\lambda \cup a^{v_1}) \cdot (\lambda \cup a^{v_2}) \cdots (\lambda \cup a^{v_n})$. □

Corollary 2 *Let regular expressions and initial number of spikes in neurons in SN P systems be described in succinct form. Then* $(\mathbf{NP} \cup \mathbf{co\text{-}NP}) \subseteq \mathbf{PSN}^*_{-del}$.

Note that if one assumes unary representation of regular expressions and of number of spikes in neurons, then uniform families of standard confluent SN P system cannot solve NP-complete problems unless $\mathbf{P} = \mathbf{NP}$. Recall that any confluent SN P system with simple regular expressions can be simulated by a deterministic Turing machine in polynomial time ([11]). With the unary representation, one can extend the result also to general regular expressions: an expression E can be transformed into the equivalent non-deterministic finite automaton (NFA) in polynomial time. Then it is decidable in polynomial time with respect to the size of E and k, whether the NFA accepts the string a^k representing k spikes in a neuron.

It remains an open problem whether a result analogous to Corollary 2 holds for standard confluent SN P systems with input. We conjecture that this can be achieved only with the aid of maximal parallelism or extended rules which would allow to transform rapidly a binary input to an exponential number of spikes present in some neuron as in [12]. Other known solutions to NP-hard problems with families of SN P systems use various extension of the standard definition, as non-confluent and non-deterministic SN P systems ([10, 11]) or exponential number of neurons ([4, 9]).

To establish an upper bound on the power of standard confluent families of SN P systems with unlimited regular expressions, we need the following lemma first:

Lemma 5 *Matching of a regular expression* E *of size* s *in succinct form over a singleton alphabet with a string* a^k *can be done on a RAM in non-deterministic time* $\mathcal{O}(s \log k)$.

Proof. Assume that we have the syntactic tree of the expression E at our disposal (its parsing can be done in deterministic polynomial time). We treat the sub-expressions of the form a^n as constants and assign them a leaf node of the tree with the value n. The matching algorithm works as follows:

- Produce non-deterministically a random element of $L(E)$ in succinct form by the depth-first search traversal of its syntactic tree: start in the root and evaluate recursively each node depending on its type as follows:
  - leaf node containing a constant: return the value of the node;
  - catenation: evaluate both subtrees of this node and add the results;
  - union: choose non-deterministically one of the subtrees of this node and evaluate it;
  - star: draw a random number of iterations x within the range $\langle 0, k\rangle$, evaluate the subtree starting in this node and multiply the result by x.
- Compare the drawn element of $L(E)$ with a^k whether they are equal or not.

Whenever, during the evaluation, the computed value exceeds k, the algorithm halts immediately and reports that a^k does not match $L(E)$. This guarantees that the number of bits processed in each operation is always $O(\log k)$.

Each of the elementary operations described above can be performed in constant time on RAM with unit instruction cost, except the multiplication which requires $O(\log k)$ time. Total number of tree-traversal steps is $\mathcal{O}(s)$. □

Theorem 5 $\mathbf{PSN}^* \subseteq \mathbf{PSPACE}$

Proof. It has been shown in [11] that any confluent SN P system with simple regular expressions can be simulated by a deterministic Turing machine in polynomial time. Assuming general regular expressions, by Lemma 5, their matching can be done in non-deterministic polynomial time, and, since $\mathbf{NP} \subseteq \mathbf{PSPACE}$, also in deterministic polynomial space. Indeed, if one replaces the random selection in the proof of Lemma 5 by the depth-first search of all configurations reachable by making non-deterministic choices, one gets a deterministic algorithm running in polynomial space and exponential time.

Denote by s the size of description of an SN P system Π. Observe that the total number of bits to describe spikes in all neurons after t steps of computation is $\mathcal{O}(s + t)$ even in the case of maximal parallelism or exhaustive rules. The total size of all regular expressions in Π is $\mathcal{O}(s)$. Hence, by Lemma 5, the simulation of Π performs in polynomial space with respect to $s + t$. □

Finally, let us note that deterministic solutions to PSPACE-complete problems QSAT and Q3SAT with families of SN P systems with pre-computed resources (i.e., with exponential amount of neurons) have been shown in [8].

6. Conclusion

We have introduced uniform families of standard confluent SN P systems and studied their closure properties and computational power under polynomial time restriction. Several factors influencing the results were focused on the input encoding, the form of output (halting versus spiking), the descriptional complexity, the form of regular expressions. It is an open problem whether the closure properties of these families can be extended to the case of unrestricted extended SN P systems.

It was shown that, with the restriction of regular expressions to the single star normal form, these families of SN P systems characterize the class **P**. It remains an open problem whether this condition can be further relaxed.

When complex regular expressions are allowed (but note that the operation * is not necessary), these families of SN P systems without input are capable to solve **NP**-complete problems in constant time, but this is only due to the capability of quick parsing of regular expressions. The succinct representation of regular expressions and of spikes in neurons is necessary to achieve this computational potential (unless **P**=**NP**). Finally, the power of these families under polynomial time restriction is bounded from above by **PSPACE**.

While presenting the results related to classes **NP** and **PSPACE**, families of SN P systems without input were considered. It is an open problem whether these results can be extended also to families of standard confluent SN P systems with input, but it is likely that extended rules and/or maximal parallelism must be allowed to achieve that.

Acknowledgements

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by the Silesian University in Opava under the Student Funding Scheme, project SGS/7/2011, and by the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2012-36992.

References

[1] Andrei S., Cavadini S. V., Chin W.-N.: A new algorithm for regularizing one-letter context-free grammars. Theoretical Computer Science, 306, 2003, pp. 113–122.

[2] Beame P. W.: Limits on the power of concurrent-write parallel machines. Information and Computation, **76**, 1, 1988, pp. 13–28.

[3] Chen M., Ionescu M., Ishdorj T.-O., Păun A., Păun Gh., Pérez-Jimenéz M. J.: Spiking neural P systems with extended rules: universality and languages. Natural Computing (special issue devoted to DNA 12 Conference), Natural Computing, 7, 2008, pp. 147–166.

[4] Gutiérrez-Naranjo M. A., Leporati A.: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. In: Díaz-Pernil D. et al.(eds.) Proceedings of Sixth Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, 2008, pp. 193–210.

[5] Hopcroft J. E., Motwani R., Ullman J. D.: Introduction to Automata Theory, Languages, and Computation (3rd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[6] Ibarra O. H., Leporati A., Păun A., Woodworth S.: Spiking neural P systems. In: Păun Gh., Rozenberg G., Salomaa A. (eds.) The Oxford Handbook of Membrane Computing, University Press, Oxford, 2009, pp. 337–362.

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

[8] Ishdorj T.-O., Leporati A., Pan L., Zeng X., Zhang X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. In: Martínez-del-Amor M. A. et al. (eds.) Seventh Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, **2**, 2009, pp. 1–27.

[9] Ishdorj T.-O., Leporati A., Pan L., Zeng X., Zhang X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. Theoretical Computer Science, In press (2010).

[10] Leporati A., Mauri G., Zandron C., Păun Gh., Pérez-Jiménez M. J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. Natural Computing, **8**, 4, 2009, pp. 681–702.

[11] Leporati A., Zandron C., Ferretti C., Mauri G.: On the computational power of spiking neural P systems. In: Gutiérrez-Naranjo M. A. et al. (eds.) Proceedings of Fifth Brainstorming Week on Membrane Computing, Sevilla, Fenix Editora, 2007, pp. 227–245.

[12] Leporati A., Zandron C., Ferretti C., Mauri G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis G. (eds.) Membrane Computing, 8th International Workshop, WMC 2007, LNCS, Springer, Heidelberg, **4860**, 2007, pp. 336–352.

[13] Neary T.: On the computational complexity of spiking neural P systems. In: Calude C. S. et al. (eds.) Unconventional Computing, 7th International Conference, UC 2008, Vienna, Austria, August 25–28, 2008. Proceedings, LNCS, Springer, Heidelberg, 5204, 2008, pp. 189–205.

[14] The P Systems Web Page. `http://ppage.psystems.eu/`. [cit. 2013-1-11].

[15] Papadimitriou C. H.: Computational Complexity, Addison-Wesley, 1994.

[16] Pan L., Păun Gh., Pérez-Jiménez M. J.: Spiking neural P systems with neuron division and budding. Science in China, Series F: Information Sciences, **54**, 8, 2011, pp. 1596–1607.

[17] 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.

[18] Pérez-Jiménez M. J.: A computational complexity theory in membrane computing. In: Păun Gh. et al. (eds.) Membrane Computing, 10th International Workshop, WMC 2009, LNCS, Springer, Heidelberg 5957, 2010, pp. 125–148.

[19] Pérez-Jiménez M. J., Romero-Jiménez A., Sancho-Caparrini F.: A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics, **11**, 4, 2006, pp. 423–434.

[20] Sosík P., Rodríguez-Patón A., Cienciala L.: On the power of families of recognizer spiking neural P systems. Intern. J. Found. Computer Sci., **22**, 1, 2011, pp. 75–88.

[21] van Leeuwen J. (ed.): Handbook of Theoretical Computer Science (vol. A), Elsevier Science Publisher, Amsterdam, 1990.

[22] Wang J., Hoogeboom H. J., Pan L.: Spiking neural P systems with neuron division. In: Gheorghe M. et al. (eds.), Membrane Computing, CMC 2010, LNCS, **6501**, Springer, Heidelberg, 2011, pp. 361–376.