Polynomial Complexity Classes in Spiking Neural P Systems

Petr Sosík[1,2], Alfonso Rodríguez-Patón[1], and Lucie Ciencialová[2]

[1] Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain
{psosik,arpaton}@fi.upm.es

[2] Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava, 74601 Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz

Abstract. We study the computational potential of spiking neural (SN) P systems. Under various conditions, several intractable problems have been proven to be solvable by these systems in polynomial or even constant time. We study first their formal aspects such as the input encoding, halting versus spiking, and descriptional complexity. Then we establish a formal platform for complexity classes of uniform families of confluent recognizer SN P systems. Finally, we present results characterizing the computational power of several variants of confluent SN P systems, characterized by classes from **P** to **PSPACE**.

1 Introduction

Spiking neural P system (abbreviated as SN P system) introduced in [4] is an abstract computing model inspired by the theory of membrane computing, on one hand, and spiking neural networks, on the other hand. 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 various conditions.

In this paper we intend to establish a platform allowing to characterize the computational power of SN P systems more precisely. Computational complexity theory provides basic tools for this task. As the first step, a sequence of formal prerequisites will be established in Section 3, including the input encoding of SN P systems, the requirements for halting and providing an output, and finally the size of the description of an SN P system. These postulates 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 (and analogously in the case of SN P system without input).

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 some 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 classes **NP**, **co-NP** and **PSPACE**.

2 Prerequisites

In this section we recall some useful notation 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. We also refer to [11] for an up-to-date information about membrane computing.

For an 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 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 to fix the notation.

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. 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 membrane 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) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \to a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
      (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 $E/a^c \to a; d$ can be applied. The application of this rule means consuming (removing) c spikes (thus only $k-c$ remain in σ_i), the neuron is fired, and it produces a spike after d time units (as usual in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost). In the step $t+d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t+d+1$).

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

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

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, $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$, 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.

The initial configuration of the system is described by the numbers of spikes $n_1, n_2, \ldots, n_m$ present in each neuron. During a computation, the "state" of the system is described by both the number of spikes present in each neuron, and the open/closed condition of each neuron: if a neuron is closed, then we have to specify when it will become open again.

Using the rules as described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. 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, then we write 1, otherwise we write 0. We refer the reader to [13] for more details.

3.1 Unary versus binary input/output

Original works on SN P systems, e.g., [4, 12] focused on SN P systems working the generating mode. This induced also the following output convention: the output value was represented as a time interval between two spikes of a designated neuron. This means that the output values were represented in *unary.* A similar convention was later adopted also for input [7].

Unary input/output encoding
An input/output sequence of natural numbers $n_1, \ldots, n_k$ is represented as a spike train $10^{n_1-1}10^{n_2-1}1 \ldots 10^{n_k-1}1$.

This unary encoding can be easily transformed to another type of unary encoding when the input/output values are represented as a number of spikes accumulated in neurons. Unary encoding is suitable for SN P systems generating sets. In the case of SN P systems computing functions or solving decision problems, binary encoding is probably a better choice. It is known that standard SN P systems can simulate logic gates with unbounded fan-in in a unit time [3] and, hence, also arbitrary logic circuits in linear time. The unary input/output convention, however, would decrease their computational power *exponentially* in many cases. Assume, e.g., a SN P system performing addition of two arguments m, n such that $m \leq n$. Logic circuits (and also other models as Turing machines) performs this operation in time $\mathcal{O}(\log n)$. However, an SN P system with unary convention needs time $\mathcal{O}(n)$ to read/write its input/output.

Binary input/output encoding
An input/output to a SN P system is a binary spike train received/emitted by a designated input/output neuron.

When dealing with families of SN P systems *without input* (see Section 4), one can encode a k-bit value into the structure of neurons of size $\mathcal{O}(k)$. If we adopted unary input conventions, then uniform and non-uniform solutions to various problems could take exponentially different time, on one hand. On the other hand, it is known that regular expressions provide SN P systems with a large part of their power, and to exploit it, one probably needs to accumulate an exponential number (in terms of size of input) of spikes in a neuron This is impossible in polynomial time when standard rules and a binary input encoding is used. It seems that both binary and unary choice lead to possible discrepancies between uniform and semi-uniform solutions.

We assume the use of binary encoding from now on. Obviously, to switch from binary to unary encoding, one needs a time exponential with respect to the size of the original binary string, unless an extended SN P system with maximal parallelism is used [9].

3.2 Spiking versus halting

In this subsection we focus on 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 (whose elements are called instances) and θ_X is a total boolean function over I_X. The following convention was suggested by some authors: a SN P systems solving an instance $w \in I_X$ would halt if and only if $\theta_X(w) = 1$. Such a SN P system is called *accepting* in [7]. However, this convention was rarely implemented. Instead, many authors demonstrated SN P systems which always halt and an output neuron spikes if and only if $\theta_X(w) = 1$, see [3, 6–9] and others. We add that this convention is more compatible with definitions of standard complexity classes and also with standard construction of families of recognizer P systems [14]. Hence 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 [7].

Lemma 2. *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 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 results in [7] 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 Π spikes.*

Proof. We start with the following construction inspired by [7]: let us consider an SN P system Π (possibly with extended rules), and let σ_{out} be its output neuron. We "triple" this system 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 $\{3n \mid n \in L(E)\}$,
– tripling each neuron σ_c : 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 by 3Π the obtained system. In this way the behavior of the system is not changed, in the sense that each neuron in 3Π spikes if and only if it spikes in Π.

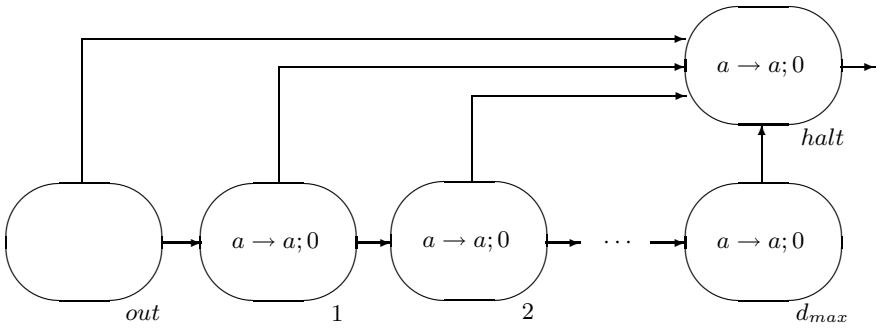

Fig. 1. A module emitting $d_{max} + 1$ consecutive spikes after the output neuron spikes.

Let d_{max} denote the maximal delay in any neuron of Π. Let us construct a module with an incoming synapse from σ_{out} which produces $d_{max}+1$ consecutive spikes when obtaining a spike from σ_{out}, see Fig. 1. The neuron σ_{halt} will have an outgoing synapse to each neuron of 3Π. Even if some of these neurons may be closed due to their refractory period, this period would end after $\leq d_{max}$ steps. Hence, during $d_{max} + 1$ steps when the neuron σ_{halt} spikes, each neuron of 3Π receives a spike, its accumulated number spikes increases to $3n+1$, $n \geq 0$, and none of its rules can be applied. However, more spikes from σ_{halt} may come. Hence, we add to each neuron of 3Π the rule $(aaa)^*aa/a \to \lambda$. In this way, each neuron will be kept with $3n + 1$ spikes and the system eventually halts after $d_{max} + 2$ steps.

Finally, let us add to 3Π a "perpetuum mobile" circuit of two mutually interconnected neurons with a single initial spike and the rule $a \to a; 0$ in each, and with an incoming synapse from σ_{halt}. In this way we ensure that 3Π halts *only* if Π spikes. □

3.3 Descriptional complexity and size of SN P systems

As in the next sections we deal with uniform families of SN P systems, it is necessary to specify the size of each member of a family. Such a study has been already presented in [8]. The size of a system Π is based on the number of bits necessary to its full description. Let m be the number of neurons, N be the

maximum natural number that appears in the definition of Π, R the maximum number of rules which occur in its neurons, and S 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 [7, 9, 13] distinguish between the size of a 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 a SN P system. Here we prefer a different approach, based on the fact that if there were a physical implementation of SN P systems, spikes in neurons would be most likely represented not as physical objects but as an electric potential as in biological neural networks. Also in recent implementations *in silico* is the number of spikes represented as a binary value. Therefore, we see no necessity to assume an exponential time effort to construct such a SN P system, and we do not distinguish between the size of a SN P system and the size of its description.

4 Families of recognizer SN P systems

Standard SN P systems were shown to be universal already in the introductory paper [4]. However, as demonstrated in [10], 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, 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. In this section we propose a formal specification for families of SN P systems. All definitions in this section follow closely [14] 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 without input (resp., with input) is* polynomially uniform by Turing machines *if there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *(resp.,* $\Pi(n)$*) from the instance* $w \in I_X$ *(resp., from* $n \in \mathbb{N}$*).*

In the sequel we will for short denote such a family just as *uniform.* For families of SN P systems with input, the selection of a proper member of the family and its 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* $w \in I_X$, $s(w)$ *is a natural number (obtained by means of a reasonable encoding scheme) and* $cod(w)$ *is a binary string – an input of the system* $\Pi(s(w))$.

The common case is that $s(w)$ is the size of w. Since Definition 4 and 5 conform those in [14, 15] we can adopt the following result whose proof in [15] 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. The following definitions are inspired by [14] and [7].

Definition 6. *Let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function. A decision problem X is solvable by a family $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ of recognizer SN P systems of type $\mathcal{R}$ without input in time bounded by f, denoted by $X \in \mathbf{SN}^*_{\mathcal{R}}(f)$, if the following holds:*

- *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 the SN P system solving an instance w can be generally nondeterministic, i.e, it may have different possible computations, but with the same result. Such a P system is also called *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 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 solutions to decision problems.

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

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *There exists a polynomial encoding (cod, s) of X in Π 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. Obviously, for any constructible function f and a class of SN P systems $\mathcal{R}$ we have

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

We use the following notation to describe a specific type $\mathcal{R}$ of SN P systems: $-reg$ for systems with regular expressions of the form a^n, $n \geq 1$, $-del$ for systems without delays, and $ssnf$ for systems with regular expressions in the single-star normal form. When $\mathcal{R}$ is omitted, the standard definition of SN P systems is used.

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 [7]. 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, without regular expressions and also for extended SN P systems).

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 described at Fig. 6 in [7] as follows. Let σ_{out} be the output neuron of this module. Let a spike emitted from σ_{out} 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 the original output neuron of Π, and with a rule $a^2 \to a; 0$. 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 holds also 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.

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. 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 [15], the case without input is analogous. Let $\mathbf{\Pi}$ by 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 $w \in I_X$, $|r(w)| \leq q(|w|)$. 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 $p(|r(w)|) \leq p(q(|w|))$ 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.

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, any standard SN P system cannot simulate Turing machine with less than exponential time and space overheads [10]. Therefore, we focus on families of SN P systems providing stronger computational power in the rest of this section.

We start with a simple variant of SN P systems with restrictions imposed on their regular expressions. It was shown already in [2] that SN P systems without delays and with all regular expressions of the form a^n, $n \geq 1$, are computationally universal. Results in [8] together with Theorems 1 and 4 in [16] imply the following statement.

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

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**.

Theorem 4. $(\mathbf{NP} \cup \mathbf{co\text{-}NP}) \subseteq \mathbf{SN}^*_{-del}(2)$

Proof. A part of the statement concerning **NP** follows by Proposition 1 in [8] which presents a construction of a standard deterministic SN P system solving the problem Subset Sum in one step. By Theorem 1, also the complement of this problem (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. □

Corollary 2. $(\mathbf{NP} \cup \mathbf{co\text{-}NP}) \subseteq \mathbf{PSN}^*_{-del}$

Note that Theorem 4 and Corollary 2 hold only for succinct (binary) representation of unary strings in regular expressions and also initial number of spikes in neurons.

Indeed, 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 [8]. With the unary representation, one can extend the result also to general regular expressions: an expression E can be transformed into the equivalent 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.

Also, it is 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 [9]. 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 [7, 8] or exponential number of neurons [3, 6].

On the other hand, we know no better upper bound on the power of standard confluent families of SN P systems with unlimited regular expressions yet than **PSPACE**. 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 a depth-first search traversal of its syntactic tree. Start with the value 0 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 $\langle 0, k \rangle$, and if $x > 0$, evaluate the subtree starting in this node and multiply the result by x, otherwise return 0.
- Compare the drawn element of $L(E)$ with a^k whether they are equal.

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. PSN* $\subseteq$ PSPACE

Proof. It has been shown in [8] that any confluent SN P system with simple regular expressions can be simulated by a deterministic Turing machine in polynomial time. Concerning general regular expressions, by Lemma 5 their matching can be done in non-deterministic polynomial time, and since **NP** $\subseteq$ **PSPACE**, also in deterministic polynomial space. Indeed, if one replaces the random selection in the proof of Lemma 5 by depth-first-search of all configurations reachable by making nondeterministic choices, one gets a deterministic algorithm running in polynomial space and exponential time.

Denote by s the size of description of a 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 a deterministic solution 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 [5].

6 Conclusion

We have introduced uniform families of standard confluent SN P systems and studied their computational power under polynomial time restriction. Several factors were focused on, influencing the obtained results: the input encoding, the form of output (halting versus spiking), the descriptional complexity, the form of regular expressions.

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 are capable to solve **NP**-complete problems in constant time. The succinct representation of regular expressions and spikes is a necessary condition to achieve this computational potential (unless **P**=**NP**). Finally, the power of these families under polynomial time restriction is bounded from above by **PSPACE**.

The results concerning intractable problems were shown for the case of families without input. It is very likely that the same result for the case of families with input is possible only when extended rules and/or maximal parallelism are allowed. However, there is no formal proof known yet.

Acknowledgements

The research was supported by the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2009–14421, by the Comunidad de Madrid (grant No. CCG06-UPM/TIC-0386 to the LIA research group), by the Czech Science Foundation, grant No. 201/09/P075, and by the Silesian University in Opava, grant No. SGS/4/2010.

References

1. Andrei, S., Cavadini, S.V. and Chin, W.-N.: A new algorithm for regularizing one-letter context-free grammars. *Theoretical Computer Science* 306, 113-122 (2003).
2. García-Arnau, M., Pérez, D., Rodríguez-Patón, A. and Sosík, P.: Spiking neural P systems: Stronger normal forms. *Intern. J. of Unconventional Computing*, 5(5), 411–425 (2009).

3. Gutiérrez-Naranjo, M.A. and 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*, pp. 193–210, Sevilla, Fenix Editora (2008).
4. Ionescu, M., Păun, Gh. and Yokomori, T: Spiking neural P systems. *Fundamenta Informaticae*, 71(2–3), 279-308 (2006).
5. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X. and Zhang, X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. In: Martínez-del-Amor, M.A., Orejuela-Pinedo, E.F., Păun, Gh., Pérez-Hurtado, I., Riscos-Núnez, A. (eds.) *Seventh Brainstorming Week on Membrane Computing*, Volume 2, pp. 1–27, Sevilla, Fenix Editora (2009).
6. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X. and Zhang, X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science*, In Press (2010).
7. Leporati, A., Mauri, G., Zandron, C., Păun, Gh. and Pérez-Jiménez, M.J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, 8(4), 681–702 (2009).
8. Leporati, A., Zandron, C., Ferretti, C. and Mauri, G.: On the computational power of spiking neural P systems. In: Gutiérrez-Naranjo, M.A., Păun, Gh., Romero-Jiménez, A., Riscos-Núnez, A. (eds.) *Proceedings of Fifth Brainstorming Week on Membrane Computing*, pp. 227–245, Sevilla, Fenix Editora (2007).
9. Leporati, A., Zandron, C., Ferretti, C. and Mauri, G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, 8th International Workshop, WMC 2007*, LNCS, vol. 4860, pp. 336-352, Springer, Heidelberg (2007).
10. Neary, T.: On the computational complexity of spiking neural P systems. In: Calude, C.S., Costa, J.F., Freund, R., Oswald, M., Rozenberg, G. (eds.) *Unconventional Computing, 7th International Conference, UC 2008, Vienna, Austria, August 25–28, 2008. Proceedings*, LNCS, vol. 5204, pp. 189-205, Springer, Heidelberg (2008).
11. The P Systems Web Page. `http://ppage.psystems.eu/`. [cit. 2009-12-29].
12. Păun, Gh., Pérez-Jiménez, M.J. and Rozenberg, G.: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17(4), 975-1002 (2006).
13. Ibarra, O.H., Leporati, A., Păun, A. and Woodworth, S.: Spiking neural P systems. In: Păun, Gh., Rozenberg, G. and Salomaa, A. (eds.) *The Oxford Handbook of Membrane Computing*, pp. 337–362, Oxford University Press, Oxford (2009).
14. Pérez-Jiménez, M.J.: A computational complexity theory in membrane computing. In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núnez, A., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, 10th International Workshop, WMC 2009*, LNCS, vol. 5957, pp. 125–148, Berlin, Springer (2010).
15. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, 11(4), 423–434 (2006).
16. Rodríguez-Patón, A., Sosík, P. and Cienciala, L.: On complexity clases of spiking neural P systems. In *Proceedings of Eighth Brainstorming Week on Membrane Computing*, Sevilla, Fenix Editora (2010). To appear.