On Complexity Classes of Spiking Neural P Systems

Alfonso Rodríguez-Patón[1], Petr Sosík[1,2], Luděk Cienciala[2]

[1] Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Monteganceo s/n, Boadilla del Monte, 28660 Madrid, Spain

[2] Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava, 74601 Opava, Czech Republic

Summary. A sequence of papers have been recently published, pointing out various intractable problems which may be solved in certain fashions within the framework of spiking neural (SN) P systems. On the other hand, there are also results demonstrating limitations of SN P systems. In this paper we define recognizer SN P systems providing a general platform for this type of results. We intend to give a more systematic characterization of computational power of variants of SN P systems, and establish their relation to standard complexity classes.

1 Introduction

The spiking neural P systems, incorporating in membrane computing ideas from spiking neurons, see, e.g., [18], were introduced in [12]. In short, an SN P system consists of a set of neurons placed in the nodes of a graph, representing synapses. The neurons send signals (spikes) along synapses (edges of the graph). This is done by means of firing rules, which are of the form $E/a^c \rightarrow a; d$, where E is a regular expression, c is the number of spikes consumed by the rule, and d is the delay from firing the rule and emitting the spike. The rule can be used only if the number of spikes collected by the neuron is "covered" by expression E, in the sense that the current number of spikes in the neuron, n, is such that $a^n \in L(E)$, where $L(E)$ is the language described by expression E. In the interval between firing a rule and emitting the spike, the neuron is closed/blocked, it does not receive other spikes and cannot fire again. There also are rules for forgetting spikes, of the form $a^s \rightarrow \lambda$ (s spikes are just removed from the neuron). Starting from an initial distribution of spikes in the neurons and using the rules in a synchronized manner (a global clock is assumed), the system evolves. A sequence of transitions among configurations of an SN P system, starting in the initial configuration, is called a computation. One of the neurons is designated as the output neuron and its spikes can also exit the system. The sequence of steps when the output neuron sends spikes to the environment is called the spike train of the computation.

In section 3.2 we propose a definition of (uniform) families of recognizer SN P systems which should be still more elaborated in the future work. Then we use this apparatus to study the computational power of deterministic (or, more generally, *confluent*) SN P systems under the following conditions:

general regular expressions / single-star normal form,
polynomial / exponential number of activated neurons,
polynomial uniformity / non-uniformity by Turing machines,
cyclic / acyclic architecture.

Main theoretical tools we use to study these topics are logic circuits and RAM computers which can simulate SN P systems in a convenient way.

2 Prerequisites

We assume the reader to be familiar with basic language and automata theory, as well as with basic membrane computing, e.g., from [22] and [26], respectively (we also refer to [21] for an up-to-date information about membrane computing), so that we introduce here only some notation used later in proofs.

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 $|x|$.

For definitions of boolean circuits we refer the reader to [2]. Let $\mathbf{UCKT}(C(n), D(n))$, where $C(n) \geq n$, denote the class of problems solvable by logspace-uniform circuit families with unbounded fan-in where element corresponding to n has size $\mathcal{O}(C(n))$ and depth $\mathcal{O}(D(n))$.

2.1 Regular expressions and normal forms

We recall the definition of regular expression mostly in order 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 $(E_1) \cup (E_2), (E_1)(E_2)$, 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.

2.2 Random Access Machines

RAM is a computing device with an infinite random access array of data registers $M_1, M_2, \ldots$, each of which can store an arbitrary integer, and with a set of labelled instructions $P = \{P_1, P_2, \ldots\}$ each from the instruction set described below.

$constant \longrightarrow M_{res}$	
$M_{op1} \longrightarrow M_{res}$	
$*M_{op1} \longrightarrow M_{res}$	The contents of a register whose address is stored in register M_{op} - indirect memory READ
$M_{op1} \longrightarrow *M_{res}$	The contents of a register M_{op} is written into the register whose address is in register M_{res} - indirect memory WRITE
$M_{op1} + M_{op2} \longrightarrow M_{res}$	
$M_{op1} - M_{op2} \longrightarrow M_{res}$	
GOTO label	
GOTO label if $M_{op1} @ M_{op2}$	$@ \in \{=, <\}$
HALT	

Non-deterministic RAM is defined in the same manner as its deterministic version except that more than one instruction can have the same label and their choice is non-deterministic.

A parallel RAM (PRAM) has a sequence of RAM's $R_1, R_2, \ldots$ operating synchronously in parallel. They all have their own local registers and they can read and write to common registers too. This can be done by indirect memory READ or WRITE. In CRCW PRAM (concurrent-read, concurrent-write) if more than one processor attempts to write into the same location in common memory at the same time, the lowest numbered processor succeeds. All processors run the same program.

In addition to the programs, another part of specification of a particular CRCW PRAM is a function $P(n)$ from positive integers to positive integers called the *processor bound*. An input of size n consists of n binary words, each of length at most n. A CRCW PRAM is given an input of size n by placing n words in the first n locations of common memory, and the first $P(n)$ processors $R_1, \ldots, R_{P(n)}$

are started. Each instruction takes one time unit. The computation halts when $R_1, \ldots, R_{P(n)}$ have all halted. The machine operates in time $T(n)$ if it halts within $T(n)$ steps on every input of size n. When the computation halts, the output can be found in initial contiguous block of common memory of length at most n. The model is essentially identical to the SIMDAG of Goldschlager [8] and similar to the P-RAM of Fortune and Wyllie [7]. Note that [8] and [7] characterized the power of these models when the number of processors grows exponentially in n.

Denote by $\mathbf{CRCW}(P(n), T(n))$ the class of problems solvable on CRCW PRAM in $\mathcal{O}(T(n))$ time with $\mathcal{O}(P(n))$ processors. It is known that $\mathbf{CRCW}(poly(n), poly(n)) = \mathbf{P}$ while $\mathbf{CRCW}(exp(n), poly(n)) = \mathbf{PSPACE}$.

Lemma 2 ([28]). $\mathbf{CRCW}(P(n), T(n)) \subseteq \mathbf{UCKT}(poly(P(n), T(n), n), T(n))$

The above result was presented first in [28] where also its detailed proof together with constructions of logical circuits implementing PRAM can be found. The formulation in Lemma 2 is inspired by [14].

3 Spiking Neural P Systems

A *spiking neural membrane system* (abbreviated as SN P system), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

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. $in, out \in \{1, 2, \ldots, m\}$ indicates 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 system is called *deterministic* if for every neuron that occurs in the system, any two rules $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$ in the neuron have $L(E_1) \cap L(E_2) = \emptyset$.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron. During a computation, the "state" of the system is described by both by the number of spikes present in each neuron, and by 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.

3.1 Descriptional complexity of SN P systems

The size of description of a SN P system is an important measure when one wants to study families of SN P systems. The first detailed specification of descriptional size of a SN P system Π is given in [16]. It is based on the number of bits necessary to fully describe the system Π. 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 that occur in Π (this will be discussed later). Then the total size of description of Π is polynomial with respect to m, R, S and $\log N$.

Some authors introducing space complexity measures for P systems as [25] suggested that an n-tuple of identical objects should occupy a space n. In SN P systems, however, the spikes are not physical objects but an electric potential. Furthermore, if we decided for unary representation of spikes, then it would be also fair to assume unary representation of regular expressions in neurons which, however, could restrict their power significantly.

Finally, observe that classical models of P systems and their uniform families (see the next section for definition of families) actually use binary representation of data. A member of the family processing inputs of size n can use $poly(n)$ different objects and code its input by their (non)presence in the input membrane.

Therefore, to represent n spikes, just the number $\log n$ of bits is taken as the size of its description, both in neurons and regular expressions. This succinct representation is used also in [16] and other papers. Note, however, that many of the results presented here remain valid even if the unary representation of spikes were adopted.

3.2 Families of SN P systems solving decision problems

Standard SN P systems were shown to be universal already in the introductory paper [12]. However, as demonstrated in [20], no standard spiking neural P system with a constant number of neurons can simulate Turing machines with less than exponential time and space overheads. Therefore, for application of SN P systems to solve intractable problems, many authors have (implicitly) 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. A first step towards such a specification was taken already in [15] for the case of *non-deterministic* (and non-confluent) SN P systems. Most of definitions in this section is motivated by [23].

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.

It was suggested by some authors that a SN P systems solving an instance $w \in I_X$ would halt if and only if $\theta_X(w) = 1$. However, this is incompatible with definitions of basic complexity classes (which do not allow non-halting computations) and also with standard construction of families of recognizer P systems [23] which is extensively used. Hence we suggest the convention used implicitly by many authors ([9, 16] and others) when describing "neural" solutions to Subset Sum, 3-SAT and similar problems: the system always halts and the result is determined by the fact whether the output neuron emits a spike during the computation. Other convention are possible, such as the existence of two output neurons signalling *true* or *false*.

Definition 3. *A* recognizer SN P system *is a SN P system which has all computations halting, and whose output neuron spikes no more than once during each computation. Its computation is called* accepting *if the output neuron spikes exactly once, otherwise it is* rejecting*.*

This definition is also compatible with the variant when the system is asked to spike *at least once* in the case of accepting computation. Any such SN P system can be added 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 it receives.

We distinguish recognizer SN P systems with input or without input. In the first case the input (i.e., an instance $w \in I_X$) is sent in the form of binary spike train $bin(w)$ to the input neuron, where $bin(w)$ is an arbitrary *binary* encoding of w. There are reasons for choosing binary encoding: it is known that standard SN P systems can simulate logic gates with unbounded fan-in in a unit time [9]. Hence, their computational potential is at least as high as that of logic circuits. The unary input/output convention, however, would decrease their computational power *exponentially* in many cases, on one hand. On the other hand, to transform the input value into the number of spikes in a neuron, the extended rules or maximal parallelism would be necessary [17].

In the case of SN P systems without input, instances of a problem are encoded within the structure of SN P system.

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.* Necessary conditions must be met by families of recognizer SN P systems to solve algorithmically a given decision problem. Conditions of *soundness* and *completeness* of $\mathbf{\Pi}$ with respect to X are defined in [23]. Conjunction of these two conditions for SN P systems without input (resp., with input) ensures that for every $w \in I_X$, if $\theta_X(w) = 1$, then every computation of the SN P systems solving w is accepting, and if $\theta_X(w) = 0$, then every such computation is rejecting. Note that this SN P system 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.*

Definition 5. *Let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function. A decision problem X is solvable in time bounded by f by a family $\mathbf{\Pi} = \{\Pi(w) : w \in I_X\}$ of recognizer SN P systems of type $\mathcal{R}$ without input, 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 time-bounded; 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 and complete with respect to X.*

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 6. *Let $f : \mathbb{N} \to \mathbb{N}$ be a constructible function. A decision problem X is solvable in time bounded by f by a family $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ of recognizer SN P systems of type $\mathcal{R}$ with input, 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 time-bounded; that is, for each instance $w \in I_X$, every computation of the member of $\mathbf{\Pi}$ solving w performs at most $f(|w|)$ steps.*
- *The family $\mathbf{\Pi}$ is sound and complete with respect to X.*

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

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.

4 Simulation of SN P systems with RAM

Theorem 1. *For each confluent (respectively, non-confluent) SN P system Π with all regular expressions in single-star normal form (SSNF) and with description of size s, there is a deterministic (resp., non-deterministic) RAM constructed in polynomial time with unit costs of operations which simulates t steps of Π in time $\mathcal{O}(t(s + t))$.*

Proof. All information about the topology of the SN P system and rules in neurons can be contained within the simulating RAM program. The configuration of each neuron of a SN P systems will be represented in RAM by three registers counting

(i) the number of spikes in the neuron, (ii) current delay period of the neuron (i.e., the remaining number of steps during which the neuron will be closed), and (iii) the number of spikes prepared to be emitted (0 or 1, but in case of *extended* SN P systems it can be more). Simulation of single step of a SN P system with RAM consists of the following phases:

1. For each neuron:
   - If the delay period is 0, check whether the number of spikes in neuron is covered by a regular expression in some of its rules. If more than one rule matches, choose one randomly. If a rule $E/a^c \to a; d$ or $a^c \to \lambda$ can be applied, then
     a) decrease the number of spikes in neuron by c,
     b) set the delay period to d in the case of firing rule,
     c) set the number of spikes to be emitted to 1 for firing rule or to 0 for erasing rule.
   - Else decrement the delay period by 1.
2. For each neuron with delay period 0 and the number of spikes to be emitted ¿0: reset the number of spikes to be emitted and increase by 1 the number of spikes in all neurons connected to the output of the processed neuron.
3. If all neurons have delay period 0, none rule was applied and none spike was emitted, halt the computation.

It is easy to verify that almost each elementary operation described above for a single neuron or a synapse can be implemented on RAM in $\mathcal{O}(1)$ time, hence the total number of necessary RAM operations for all neurons is $\mathcal{O}(s)$.

The only exception is the evaluation whether the number of spikes in a neuron is covered by regular expressions. Consider a regular expression $E = E_1 \cup \ldots \cup E_n$ in SSNF. Each E_i, $1 \leq i \leq n$, can be easily rewritten to the form $a^q(a^r)^*$, for $q, r \geq 0$. The evaluation whether E_i covers a number k of spikes present in the neuron consist of checking the divisibility of $k - q$ by r. This operation is performed in time proportional to the number of bits of k. Total number of bits to describe all spikes in all neurons after t steps of computation is $\mathcal{O}(s + \log t)$ for standard SN P systems, or $\mathcal{O}(s + t)$ in the case of maximal parallelism or exhaustive rules. Hence, the total number of RAM operations necessary to simulate t steps is $\mathcal{O}(t(s + t))$.

The situation is more complicated when we deal with general regular expressions in neurons. Let us prove the following lemma first.

Lemma 3. *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 or a Turing machine in non-deterministic polynomial time with respect to $s \log k$.*

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 subexpressions 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 $\langle 0, k \rangle$, 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.

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 price, except the multiplication which requires $O(\log k)$ time. Total number of tree-traversal steps is $\mathcal{O}(s)$. If we implement the algorithm on Turing machine, the resulting time increases only polynomially.

Theorem 2. *Any confluent or non-confluent SN P system Π of size s with general regular expressions which performs t steps can be simulated on a RAM in polynomial space with respect to t and s.*

Proof. Consider the confluent case first. The construction of RAM used in the proof of Theorem 1 can remain in this proof unchanged except the problem of matching a number of spikes in a neuron with a general regular expression E over one letter alphabet. Lemma 3 states that this problem can be solved in a non-deterministic polynomial time on a RAM. Therefore, it might seem that the whole simulation of a SN P systems might belong to the class **NP**. However, it is not the case (unless **NP=co-NP**). The problem is that we cannot iterate the non-deterministic solutions guessing whether certain neuron spikes, since the overall result of computation might depend of the fact whether some neurons *do* spike and others *do not* spike.

Therefore, since **NP** $\subseteq$ **PSPACE**, we can only guarantee that the whole simulation can be done in deterministic polynomial space. Indeed, if one replaces the random selection in the proof of Lemma 3 by dept-first-search of all possible variants, one gets a deterministic algorithm performing matching in polynomial time and exponential space.

Finally, the same argument can be applied for the case of non-deterministic and non-confluent SN P systems. In this case the simulating machine (RAM) must further keep trace of non-deterministic choices of rules in each step and try systematically all the possibilities. This requires obviously a polynomial space with respect to s and t.

5 Relation to standard complexity classes

For the first result we recall the NP-complete problem SUBSET SUM:

Problem 1. NAME: SUBSET SUM

INSTANCE: a (multi)set $V = \{v_1, v_2, \dots, v_n\}$ of positive integers, and a positive integer S.
QUESTION: is there a sub(multi)set $B \subseteq V$ such that $\sum_{b \in B} = S$?

The usual agreed instance size of SUBSET SUM is $\Theta(n \log K)$, where $K = \max\{v_1, \dots, v_n, S\}$.

Theorem 3. *The problem* SUBSET SUM *is solvable in semi-uniform way by a uniform family of deterministic recognizer SN P systems in constant time, where each member of the family solving an instance of size* $n \log K$ *has a single-neuron and a description of size* $\mathcal{O}(n \log K)$.

Proof. Consider the SN P system described in [16], Proposition 1 at p. 242–243. Given an instance of the SUBSET SUM problem defined ibidem, the construction of the SN P system is clearly polynomially uniform by Turing machine, the size of the description of the system is $\mathcal{O}(n \log K)$ (i.e., equivalent to the size of the instance), the system is deterministic and it solves the given instance in a single step.

Note: The above theorem remains valid only in the case of succinct representation of spikes and regular expressions. Otherwise the size of the SN P system would be nK, i.e., exponentially greater than the size of the instance.

The following lemma is due to [9] where detailed constructions of acyclic "neural" modules simulating gates with unbounded fan-in can be found.

Lemma 4. *SN P systems with regular expressions of the form* a^n, $n \geq 1$ *and without delays can simulate logic gates AND, OR, XOR with unbounded fan-in and fan-out in constant time and space.*

Theorem 4. *For an arbitrary polynomial* T, $\mathbf{CRCW}(poly(n), T(n)) \subseteq \mathbf{SN}_{-reg,-del}(T(n))$

Proof. By Lemma 4, $\mathbf{UCKT}(poly(n), T(n)) \subseteq \mathbf{SN}_{-reg,-del}(T(n))$ since a polynomially-sized circuit can be simulated in linear time a SN P system which can clearly be constructed in polynomial time by a Turing machine, and hence is a member of a uniform family. Then by Lemma 2, $\mathbf{CRCW}(poly(n), T(n)) \subseteq \mathbf{UCKT}(poly(n), T(n))$ for a polynomial T and, furthermore, the family of circuit is logspace-uniform, hence also polynomial time uniform by Turing machines. The whole construction of the simulating SN P system is therefore polynomially uniform and the statement of the theorem follows.

Note that the above result would hold for acyclic SN P systems as well since the neural modules referred to in Lemma 4 are acyclic. Since SN P systems allow for cyclic architectures, one might ask whether this fact could not improve the above result. Unlike acyclic circuits, SN P systems could simulate many steps of a CRCW PRAM program by the same computing modules. However, the size of the resulting SN P system would still increase with $P(n)$ and $T(n)$ as the number of bits processed by each RAM unit is bounded by $O(T(n))$ and the size of the SN P system increases accordingly. Hence in is interesting to observe that the computational power of uniform families of cyclic and acyclic SN P systems *differs only polynomially.* Solving the same problem, cyclic systems would have their size bounded by a polynomial of a lower degree.

The statement of the above theorem was partially foretold already in [10] where "neural" circuits of SN P systems performing arithmetics are presented. In that paper, however, these operations are performed sequentially, bit-by-bit, unlike in standard binary computers where all bits are processed in parallel.

Corollary 1. $\mathbf{P} = \mathbf{PSN}_{ssnf} = \mathbf{PSN}^*_{ssnf}$

Proof. Observe that $\mathbf{CRCW}(poly(n), poly(n)) = \mathbf{P}$, then the statement follows by Theorem 4 and 1.

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

Proof. Follows by Theorems 2 and 3. Note that, thanks to the power of general regular expressions, there is no difference between constant and polynomial time.

We do not know whether for families of recognizer SN P systems also uniform solutions to NP-complete problems would be possible, allowed by some kind of "universal regular expression" for a given size of instances.

6 Beyond P and NP

Our first focus in this section is devoted to *non-uniform* families of SN P systems. This means that an unlimited number of resources and even a super-Turing computing devices can be used to construct members of the family. Clearly, without any further limitations, such families could solve any decision problem including undecidable ones. Therefore certain limitations are imposed especially on the size of members of the family. It is known that, to characterize the computing power of such families, the Turing machine with advice function is a useful tool. Informally, an advice is a binary string which is given to the Turing machine solving a decision problem in addition to its input. The advice is the same for all inputs of identical size, hence it depends only on the size of the input. It contains (possibly non-computable) correct information which can help to adopt a decision about the input.

Definition 7. *Let $f : N \to N$. An $f(n)$-advice sequence is a sequence of binary strings $A = (a_1, a_2, \ldots)$, where $|a_n| \leq f(n)$. For a language $B \subseteq \{0, 1, \#\}^*$ let $B@A = \{x|\ x\#a_{|x|} \in B\}$. Let $\mathbf{P}/f(n) = \{B@A : B \in P$ and A is an $f(n)$-advice sequence$\}$. Let $\mathbf{P}/\text{poly} = \bigcup_k \mathbf{P}/n^k$.*

Theorem 6. *Non-uniform families of deterministic recognizer SN P systems of polynomial size with regular expressions in single-star normal form can compute in polynomial time exactly the class of problems* $\mathbf{P}/\text{poly}$.

Proof. By Theorem 1, any polynomial-time restricted computations of a SN P system of a polynomial size with regular expressions in single-star normal form can be simulated by RAM (and hence also by Turing machine) in polynomial time. Furthermore, by Theorem 4 in [24], each computation of such a Turing machine can be simulated by a polynomial size circuit. Finally, by Theorem 2.2 in [2], each problem with polynomial circuit complexity is in $\mathbf{P}/\text{poly}$.

The converse inclusion follows by Lemma 4 which shows that each circuit of polynomial size can be simulated by a SN P system of an equivalent size with simple regular expressions.

Note that uniform solutions are necessary because they form a part of the standard definition of sets recognized by circuits [2]. If we thought about semi-uniform solutions, then for each instance we could have a special SN P system, and since their family is non-uniform, it could solve in constant time *any* problem, even non-decidable, by pre-computing the result.

Note also that, even if the sets recognized by the mentioned families of SN P systems are also recognized by a Turing machine in polynomial time, this does *not* imply that these sets are in **P** since these machines were derived from SN P systems constructed in a non-uniform way and hence they can contain an advice of polynomial size.

Finally we briefly mention SN P systems with pre-computed resources. The basic idea is that we start with an exponentially large number of inactive neurons, which will be subsequently activated and used during the computation. These neurons are arranged into a simple regular structure in the sense that a prototype neuron and the interconnection pattern is computed in polynomial time and then simply replicated [9]. It has been shown [13] that such SN P systems can solve in linear time **PSPACE**-complete problems QSAT and Q3SAT.

Conjecture 1. If a *deterministic* SN P system can activate an *exponential number of neurons* in a polynomial time, the resulting machine is equivalent to the parallel RAM, a standard Second Machine Class model, whose polynomial time-bounded computations characterize **PSPACE**.

Proof. (Sketch) The statement follows by Theorem 1, Theorem 4 and by the construction of CRCW PRAM as described, e.g., in [28].

Similar arguments indicate that, if a *non-deterministic* SN P system can activate an *exponential number of neurons* in a polynomial time, the resulting machine may prove equivalent to the P-RAM [7], a computing model whose non-deterministic polynomial time-bounded computations characterize the class **NEXPTIME**. These open problems are the next to be addressed in the future research.

7 Conclusion

In this contribution we tried to generalize results concerning complexity issues of SN P systems published previously in a sequence of papers. We have defined (uniform) families of recognizer SN P systems which should still be more elaborated in future work. Main results can be summarized as follows:

1. Polynomially uniform families of recognizer SN P systems with regular expressions in *single-star normal form* characterize by their polynomial time-bounded computations the class **P**.
2. Polynomially uniform families of recognizer SN P systems with *general regular expressions* can in polynomial time solve a class of problems bounded between **NP** and **PSPACE**.
3. If a confluent or deterministic SN P system can activate an *exponential number of neurons* in a polynomial time, it is probably computationally equivalent to parallel RAM, a standard Second Machine Class model whose computations in polynomial time characterize **PSPACE**.
4. *Non-uniform* families of SN P systems of *polynomial size* with regular expressions in *single-star normal form* characterize by their *uniform solutions* in polynomial time the class **P**/poly.

Rather surprisingly, it turned out that there is no substantial difference in power of uniform families cyclic and acyclic SN P systems. The use of single-star normal form for regular expressions to demonstrate the borderline between SN P systems able to solve tractable and intractable problems is also interesting.

Acknowledgements

Authors are grateful to Mario Pérez-Jiménez for suggestions about definitions of complexity classes for SN P systems. The research was partially supported by the Ministerio de Ciencia e Innovación (MICINN), Spain, under project TIN2009–14421, by the program I3, by the Comunidad de Madrid (grant No. CCG06-UPM/TIC-0386 to the LIA research group), and by the Silesian University in Opava, grant No. SGS/4/2010.

References

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

2. R.B. Boppana and M. Sipser. The complexity of finite functions. In van Leeuwen [29], pages 757–804.
3. Cristian S. Calude, José Félix Costa, Rudolf Freund, Marion Oswald, and Grzegorz Rozenberg, editors. *Unconventional Computing, 7th International Conference, UC 2008, Vienna, Austria, August 25-28, 2008. Proceedings*, volume 5204 of *Lecture Notes in Computer Science*. Springer, 2008.
4. D. Díaz-Pernil et al., editor. *Proceedings of Sixth Brainstorming Week on Membrane Computing*, Sevilla, 2008. Fénix Editora.
5. George Eleftherakis, Petros Kefalas, and Gheorghe Paun, editors. *Proceedings of the 8th International Workshop on Membrane Computing*. SEERC, 2007.
6. George Eleftherakis, Petros Kefalas, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors. *Membrane Computing, 8th International Workshop, WMC 2007*, volume 4860 of *Lecture Notes in Computer Science*. Springer, 2007.
7. S. Fortune and J. Wyllie. Parallelism in random access machines. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118, New York, NY, USA, 1978. ACM.
8. L.M. Goldschlager. A universal interconnection pattern for parallel computers. *J. ACM*, 29(4):1073–1086, 1982.
9. M.A. Gutiérrez-Naranjo and A. Leporati. Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. In D. Díaz-Pernil et al. [4], pages 193–210.
10. M.A. Gutiérrez-Naranjo and A. Leporati. Performing arithmetic operations with spiking neural P systems. In Martínez-del-Amor et al. [19], pages 181–198. Volume 1.
11. M.A. Gutiérrez-Naranjo, G. Păun, A. Romero-Jiménez, and A. Riscos-Núñez, editors. *Proceedings of Fifth Brainstorming Week on Membrane Computing*, Sevilla, 2007. Fénix Editora.
12. M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2–3):279–308, 2006.
13. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, and X. Zhang. Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science*, In Press, 2010.
14. R.M. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In van Leeuwen [29], pages 869–942.
15. A. Leporati, G. Mauri, C. Zandron, G. Păun, and M.J. Pérez-Jiménez. Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, 8(4):681–702, 2009.
16. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. On the computational power of spiking neural P systems. In Gutiérrez-Naranjo et al. [11], pages 227–245.
17. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. Solving numerical NP-complete problems with spiking neural P systems. In Eleftherakis et al. [6], pages 336–352.
18. W. Maass and C. Bishop, editors. *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.
19. M.A. Martínez-del-Amor, E.F. Orejuela-Pinedo, G. Păun, I. Pérez-Hurtado, and A. Riscos-Núñez, editors. *Seventh Brainstorming Week on Membrane Computing*, Sevilla, 2009. Fénix Editora.
20. T. Neary. On the computational complexity of spiking neural P systems. In Calude et al. [3], pages 189–205.
21. The P Systems Web Page. `http://ppage.psystems.eu/`. [cit. 2009-12-29].

22. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, 2009.
23. M.J. Pérez-Jiménez. A computational complexity theory in membrane computing. In Păun et al. [27], pages 125–148.
24. N. Pippenger and M.J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.
25. A.E. Porreca, A. Leporati, G. Mauri, and C. Zandron. Introducing a space complexity measure for P systems. *Int. J. of Computers, Communications & Control*, IV(4):301–310, 2009.
26. G. Păun. *Membrane Computing – An Introduction*. Springer, Berlin, 2002.
27. G. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors. *Membrane Computing, 10th International Workshop, WMC 2009*, volume 5957 of *Lecture Notes in Computer Science*, Berlin, 2010. Springer.
28. L. Stockmeyer and U. Vishkinj. Simulation of parallel random access machines by circuits. *SIAM J. Comput.*, 13 (2):409–422, 1984.
29. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity. Elsevier, Amsterdam, 1990.