Gheorghe Păun | Mario J. Pérez-Jiménez | Arto Salomaa

Bounding the Indegree of Spiking Neural P Systems

Turku Centre *for* Computer Science

TUCS Technical Report
No 773, June 2006

Bounding the Indegree of Spiking Neural P Systems

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

Mario J. Pérez-Jiménez
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Arto Salomaa
Turku Centre for Computer Science
Lemminkäisenkatu 14, 20520 Turku, Finland
asalomaa@utu.fi

TUCS Technical Report
No 773, June 2006

Abstract

We continue the search of normal forms for spiking neural P systems, and we prove that the indegree of such systems (the maximal number of incoming synapses of neurons) can be bounded by 2 without losing the computational completeness.

Keywords: spiking neuron, spike train, membrane computing, synapses, indegree of graph, recursively enumerable

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Introduction

This paper is a direct continuation of [1], where several normal forms for spiking neural P systems [2] were found. In particular, it was shown that the outdegree of these systems can be diminished to 2, still preserving the equivalence with Turing machines. The dual problem, of the indegree bounding, was formulated as an open problem in [1]. We solve here this problem, proving, like in the case of the outdegree, that again a normal form holds true: systems with indegree two are computationally complete.

In the next section we will introduce the spiking neural P systems (in short, SN P systems), but for the reader's convenience, we informally mention here that an SN P system consists of a set of *neurons* placed in the nodes of a graph whose edges are called *synapses*; the neurons contain *spikes*, objects of a unique type, which can be processed by means of *firing/spiking rules* and by means of *forgetting rules*; the rules of the first type consume some spikes and produce a new spike, which is sent to all neurons linked by a synapse to the neuron where the rule was used, while the forgetting rules just remove spikes from neurons. One of the neurons is designated as the *output* one, and its spikes can also exit into the environment, thus providing a trace of the system evolution. Like in [2], as the result of a computation we consider the number of steps elapsed between the first two spikes which exit (the system through) the output neuron, without caring about subsequent spikes; in particular, we do not care whether or not the computation halts. (Our result can be easily extended to other ways of defining the output of a computation, such as those considered in [4], but we do not explicitly consider other cases here.)

In this way, we can compute all Turing computable sets of natural numbers. In [1], it is shown that the computing power is not diminished by requiring that each neuron has only two outgoing synapses. The same is proved here for the ingoing synapses. Actually, the two restrictions can be combined: SN P systems with indegree at most two and outdegree at most two are Turing complete.

What remains to investigate are the families of sets of numbers generated by systems with (indegree, outdegree) $\in \{(1,1),(1,2),(2,1)\}$. We show here that they contain all finite sets of numbers and are included in the family of semilinear sets of numbers, but a more precise study of these families remains to be carried out.

2 Prerequisites

We assume the reader to have some familiarity with (basic elements of) language and automata theory, e.g., from [5], and introduce directly the computing devices we investigate.

A *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, is a construct of the form

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

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 by the neuron;

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 with a the only symbol used, $c \geq 1$, and $d \geq 0$;

(2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \in L(E)$ for no rule $E/a^c \to a; d$ of type (1) from R_i;

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

4. $out \in \{1, 2, \ldots, m\}$ indicates the *output neuron*.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows: if the neuron contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a; d$ can be applied, and this means that c spikes are consumed, only $k - c$ remain in the neuron, the neuron is fired, and it produces a spike after d time units (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, and so on. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then the spike is lost). In step $t+d$, the neuron spikes and becomes again open, hence can receive spikes (which can be used in step $t+d+1$). A spike emitted by a neuron σ_i branches and goes to all neurons σ_j such that $(i, j) \in syn$.

The rules of type (2) are *forgetting* rules, and they are applied as follows: if the neuron contains exactly s spikes, then the rule $a^s \to \lambda$ can be used, and this means that all s spikes are removed from the neuron.

In each time unit, in each neuron which can use a rule we have to use a rule, either a firing or a forgetting one. Because 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 then one of them is chosen non-deterministically. Note however that we cannot interchange a firing rule with a forgetting rule, as all pairs of rules $E/a^c \to a; d$, $a^s \to \lambda$ have disjoint domains, in the sense that $a^s \notin L(E)$.

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 system is described both by the numbers of spikes present in each neuron and by the state of each neuron, in the open-closed sense; specifically, if a neuron is closed, we have to specify the moment when it will become again open.

Using the rules as suggested above, we 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 $\langle t_1, t_2, \ldots \rangle$ of natural numbers $1 \leq t_1 < t_2 < \ldots$, indicating time instances, when the output neuron sends a spike out of the system (we also say that the system itself spikes at that time). With any spike train containing at least two spikes we associate a result, in the form of the number $t_2 - t_1$; we say that this number is computed by Π. The set of all numbers computed in this way by Π is denoted by $N_2(\Pi)$ (the subscript indicates that we only consider the distance between the first two spikes of any computation).

There are several parameters describing the complexity of an SN P system: number of neurons, number of rules, number of spikes consumed or forgotten by a rule, etc. Here we consider only the following two: the outdegree and the indegree of the synapse graph. By $NSNP(ind_n, oud_m)$ we denote the family of sets $N_2(\Pi)$ computed by SN P systems Π with the indegree at most n and the outdegree at most m. As usual, if one of these parameters is not bounded, then we replace the respective subscript with $*$. By $NFIN, NSLIN, NRE$ we denote the families of finite, semilinear, and Turing computable sets of (positive) natural numbers (number 0 is ignored).

In [2] it is proved that $NSNP(ind_*, oud_*) = NRE$, then in [1] this result was improved to $NSNP(ind_*, oud_2) = NRE$, leaving as an open problem whether or not a similar result is true also for indegree.

In the next section we will prove that this is the case, and in the proof we use the characterization of NRE by means of register machines.

Such a device – in the non-deterministic version – is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction `HALT`), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n present in register 1 at that time is said to be generated by M. (Without loss of generality we may assume that in the halting configuration all other registers are empty; also, we may assume that register 1 is never subject of SUB instructions, but only of ADD instructions.) It is known (see, e.g., [3]) that register machines generate all sets of numbers which are Turing computable.

3 Bounding the Indegree

The main result of our note is the following one:

Theorem 3.1 $NSNP(ind_2, oud_*) = NRE$.

Proof. We only have to prove the inclusion $NRE \subseteq NSNP(ind_2, oud_*)$, and we do this in two steps: first we modify the constructions from [2], [4] by which the similar inclusion is proved without a bound on the indegree (an SN P system is constructed, simulating a given register machine), then we also bound the indegree of the constructed SN P system.

For the first step, let us take an arbitrary register machine $M = (m, H, l_0, l_h, I)$, as specified in the end of the previous section. We construct

an SN P system Π such that $N(M) = N_2(\Pi)$, following the same idea as in [2]: modules are built for simulating the ADD and SUB instructions of M, as well as for providing the output (i.e., for sending out two spikes at the right moments of time). A neuron is associated with each register and with each label of M; if a register r contains the number n, then the corresponding neuron σ_r contains $2n$ spikes. At the beginning of the computation, there is only one spike in the system, in neuron σ_{l_0}. This means that in the first step, this neuron fires. In general, a neuron associated with a label of M is empty during the computation, except when it is activated by receiving a spike. We will describe furthermore the functioning of the system Π after presenting its modules.

An ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is simulated by a module as indicated in Figure 1, and an SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated by a module as in Figure 2.

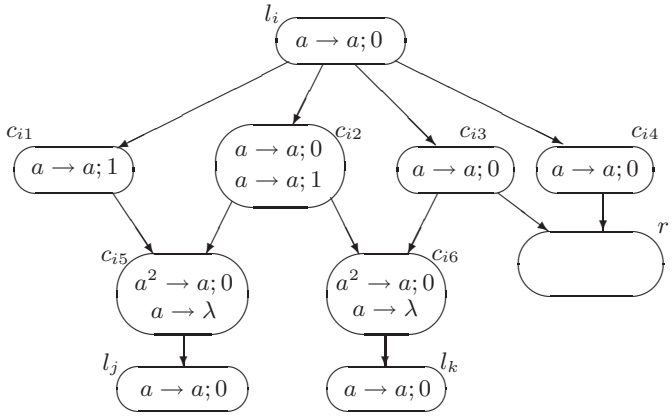

Figure 1: Module ADD, simulating the instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$

These modules are similar to those in [2], with an additional care paid to the indegree of certain neurons. Specifically, one introduces the new neurons with labels c_{i5}, c_{i6} in module ADD and c_{i3}, c_{i4} in module SUB (note that all neurons $\sigma_{c_{ij}}$ are uniquely associated with the respective modules, because the label l_i is associated with only one instruction of M). Thus, we do not give full details concerning the functioning of these modules, and refer the reader to [2]; we only mention that the execution of a module starts by introducing a spike in neuron σ_{l_i}, and ends by introducing a spike in one of the neurons with labels l_j and l_k, thus activating the respective module. The correct choice of the "exit" neuron is ensured by the interplay between neurons with spiking rules $a \to a; 0$ and $a \to a; 1$. In the meantime, the neuron σ_r receives two spikes in the case of the ADD module, or is checked for zero and two spikes removed when this is possible, in the case of the SUB module.

If the computation of M never halts, then the work of ADD and SUB modules of Π never halts. If the instruction $l_h : \texttt{HALT}$ is reached, then neuron σ_{l_h} receives a spike, and then the OUTPUT module from Figure 3 is activated. Note that the ADD modules do not use rules of neurons σ_r and that neuron σ_1 is only subject of modules ADD (register 1 is never decremented). This ensures the correct functioning of module OUTPUT, which will spike exactly twice, after a number of steps equal to the contents of register 1 of M.

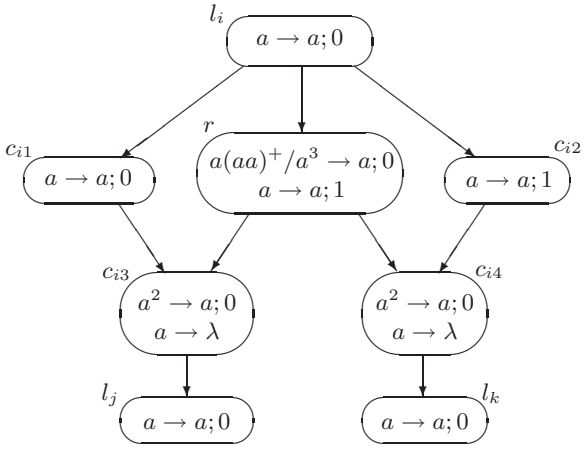

Figure 2: Module SUB, simulating the instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$

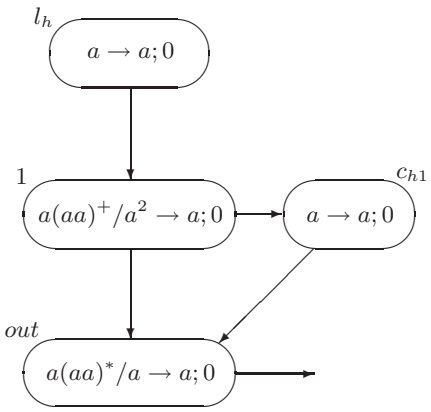

Figure 3: Module OUTPUT

The equality $N(M) = N_2(\Pi)$ is obtained. Let us now examine the indegree of the system Π. Neurons with labels c_{ij} have the indegree one or two, but neurons associated with labels of M and with registers of M can have an arbitrarily large indegree.

The case of neurons σ_l, where $l \in lab(M)$, is simpler: in each step of a computation, each such neuron can receive at most one spike along one of its incoming synapses. By introducing intermediate neurons as suggested in Figure 4, we can replace the synapses to neuron σ_l in such a way that its indegree becomes 2. Note that instead of one computation step, we perform now a number of steps of the order of $\log_2 k$, where k was the previous indegree of the neuron.

Slightly more complex is the situation of neurons σ_r: in a step of a computation, such a neuron receives *no spike* if it is not involved in the current operation, *one spike* if it is involved in a SUB instruction, or *two spikes* if it is involved in an ADD instruction. Assume that we have the synapses $(e_j, r), 1 \leq j \leq s$, along which one spike can come, and the pairs of synapses $(c_j, r), (d_j, r)$, for

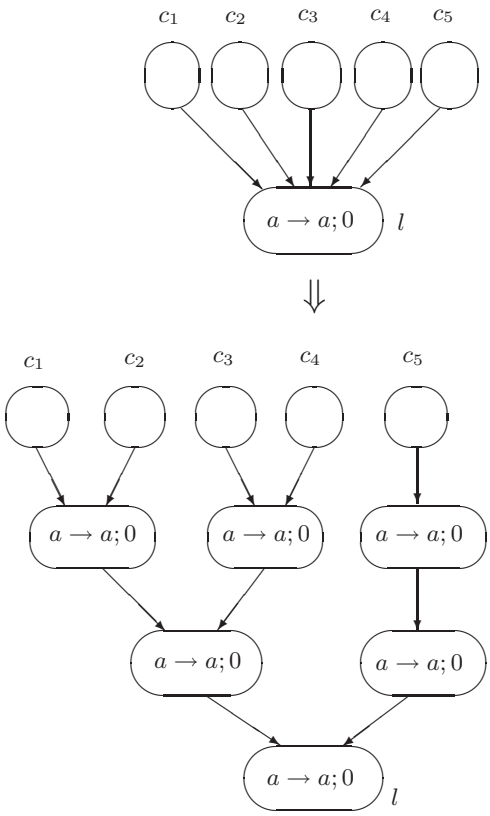

Figure 4: Decreasing the indegree of neurons $\sigma_l, l \in lab(M)$

some $1 \leq j \leq k$, such that one of the pairs of neurons $(\sigma_{c_j}, \sigma_{d_j})$ sends two spikes to σ_r. Then we can proceed as follows: we apply the procedure described in Figure 4 separately for neurons σ_{c_j}, for neurons σ_{d_j}, and for neurons σ_{e_j}, concentrating step by step the synapses until reaching an intermediate unique neuron $\sigma_C, \sigma_D, \sigma_E$, respectively. From neurons σ_D, σ_E we build synapses to a further neuron, σ_{DE}. Now, from σ_C and σ_{DE} we construct synapses to the neuron σ_r. The indegree of all neurons is now at most two (of course, "delaying" neurons, using a rule $a \to a; 0$ only for passing the spike further, are necessary if $s \neq k$).

Still, a problem remains to be solved, that of the synchronization of the system. The procedures described above take more steps than initially necessary for the spikes to reach their targets. Let us denote by α the maximal number of steps necessary in any of the previously described procedures for sending the spikes from the input neurons to the output neurons.

First, we add "delaying" neurons to constructions as the one in Figure 4, such that all blocks of this type take exactly α steps for sending the spikes from the input to the output neurons (this is an easy task: just add neurons with the rule $a \to a; 0$ as many times as necessary).

Let us now examine again the modules ADD and SUB as changed after decreasing the indegree.

In module ADD, the way from $\sigma_{c_{i3}}$ and $\sigma_{c_{i4}}$ to σ_r takes now α steps instead of one; in order to re-synchronize the process, we have to add $\alpha - 1$ delaying

neurons also along the synapses $(l_i, c_{i1}), (l_i, c_{i3}), (c_{i2}, c_{i5}), (c_{i3}, c_{i6})$. In this way, the paths from σ_{l_i} to $\sigma_{c_{i5}}$ and $\sigma_{c_{i6}}$ will last $\alpha + 1$ steps, as that from σ_{l_i} to σ_r.

Similarly for the SUB modules: we add $\alpha - 1$ delaying neurons along synapses $(l_i, c_{i1}), (l_i, c_{i2})$, and thus the spikes of σ_{l_i} reach all neurons $\sigma_r, \sigma_{ci1}, \sigma_{c_{i2}}$ at the same time, after α steps.

In this way, the system obtained in the end of all these operations is equivalent with Π and has the indegree 2. □

The previous construction can be combined with the construction used in [1] for decreasing the outdegree, hence we get the following combined normal form result:

Corollary 3.1 $NSNP(ind_2, oud_2) = NRE$.

4 The Size of the Small Families

What remains to investigate is the size and the properties of families $NSNP(ind_i, oud_j)$ for $(i, j) \in \{(1, 1), (1, 2), (2, 1)\}$.

First, let us remark that $NFIN \subseteq NSNP(ind_1, oud_1)$: given a finite set $F = \{n_i \mid 1 \leq i \leq k\}$ of natural numbers, for the system

$$\Pi = (\{a\}, (2, \{a^2/a \to a; 0\} \cup \{a \to a; n_i - 1 \mid 1 \leq i \leq k\}), \emptyset, 1)$$

we have $N_2(\Pi) = F$ (we can consider that this system has the indegree and the outdegree zero, as no synapse appears in it).

Then, because the systems with outdegree one cannot increase the number of spikes from their neurons, it follows (as already observed in [2]) that $NSNP(ind_2, oud_1) \subseteq NSLIN$ (the system can be easily simulated by a finite automaton).

A similar result is valid also for the family $NSNP(ind_1, oud_2)$. A system with such indegree and outdegree has the synapse graph of a very particular form: a possible cycle, from which starts binary trees. If there is no cycle, then only the tree containing the output neuron is relevant (no other tree contributes to the computations which determine the output), and from it only the synapses going to the output neuron – hence we can reduce the tree to a line. If there is a cycle, and the output neuron is on it, then no tree is relevant. If there is a cycle and the output neuron is on a tree emerging from the cycle, then we can trim all trees different from the one containing the output neuron, as well as all branches of this tree which are not on the way from the cycle to the output neuron, or after the output neuron.

In conclusion, the graph is either a linear tree ended with the output neuron, or a cycle from which emerges a linear tree ended with the output neuron. In the first case, there are only a finite number of possible computations, hence the generated set of numbers is finite. In the second case, the number of spikes cannot increase in the neurons of the cycle, but it can increase in the neurons of the tree, because the cycle can repeatedly introduce spikes in the tree. However, if a neuron of the tree can ever use a rule, then its contents cannot increase unboundedly: after using a spiking rule or a forgetting rule, the number of spikes in the neuron decreases. From the cycle, we get at most one spike in a step, hence the increase of the number of spikes is one by one; this means that when we reach again the number of spikes which enable the used rule, the rule is used again, decreasing the number of spikes. If a neuron never uses a

rule, then it is useless, and can be removed from the system, and then we can remove also all neurons not linked to the output neuron. Consequently, again the contents of neurons is bounded, hence can be controlled by the states of a finite automaton.

We synthesize all these observations in the following theorem:

Theorem 4.1 $NFIN \subseteq NSNP(ind_1, oud_1) \subseteq NSNP(ind_i, oud_j) \subseteq NSLIN$, *for each* $(i, j) \in \{(1, 2), (2, 1)\}$.

We *conjecture* that all families $NSNP(ind_i, oud_j)$ from the previous result are equal to $NFIN$.

5 Closing Remarks

We have proved here that the indegree of spiking neural P systems can be bounded by 2 without losing the computational completeness, also, in parallel with bounding the outdegree by 2. It remains as a research topic to see whether other graph-theoretic restrictions might be of interest for SN P systems. Cycle structure of the graphs could be studied. As regards planarity, it is interesting to note that most of the examples from [2] and [4] deal, indeed, with planar SN P systems, but this is not the case with the systems used in the proofs. In turn, the ADD, SUB, and OUTPUT modules from the proof of Theorem 3.1 are also planar, but their combination in the system is not necessarily so, because this depends on the relations between the instructions of the starting register machine.

References

[1] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In *Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.

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

[3] M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

[4] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, to appear (also available at [6]).

[5] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.

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

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1738-3
ISSN 1239-1891