Spiking Neural P Systems: Stronger Normal Forms

MARC GARCÍA-ARNAU¹, DAVID PÉREZ¹, ALFONSO RODRÍGUEZ-PATÓN¹ AND PETR SOSÍK¹,²

¹Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Facultad de Informática, Madrid, Spain
²Institute of Computer Science, Silesian University, Opava, Czech Republic
Email: petr.sosik@ff.slu.cz

Received: May 2007. Accepted: September 2007.

Spiking neural P system is a computing device recently introduced as a bridge between spiking neural nets and membrane computing. In this paper we focus on normal forms of these systems while preserving their computational power. We show that certain combinations of existing normal forms are possible without loss of computational power, thus answering partially open problems stated in [3, 8]. We also extend some of the already known normal forms for spiking neural P systems by considering determinism and strong acceptance condition.

Keywords: P system, spiking neuron, membrane computing, normal form.

1 INTRODUCTION

Spiking neural P system (SN P system) is a new bio-inspired computational model that incorporates to membrane computing [9] some ideas from spiking neurons [5, 6].

Since they were first presented in [4], the number of publications dealing with this model is constantly growing. An interesting review on the current research topics in SN P systems can be found in [8].

Informally, an SN P system consists of a set of neurons placed in the nodes of a graph that are linked by synapses. These neurons send signals (spikes) along the arcs of the graph according to firing (or spiking) rules. Rules are of the form $E/a^r \rightarrow a; t$ with $E$ being a regular expression, $r$ being the number of spikes consumed by the rule and $t$ being the delay from firing the rule and emitting the spike. A firing rule can be used only if the number $n$ of spikes

411
collected by the neuron is such that \(a^n \in L(E)\), that is, \(a^n\) is covered by the regular expression \(E\), and \(n \geq r\). The neurons also imitate the refractory period of real neural cells. A neuron is closed/blocked for exactly \(t\) time steps after firing. During this period it cannot fire again. The second type of rules have the form \(a^i \rightarrow \lambda\) and are called forgetting rules. They are used to simply forget (remove) \(s\) spikes from a cell. SN P systems start from an initial configuration of spikes and evolve in a synchronized manner (a global clock is assumed for the whole system). One of the neurons is designated as the output cell and the spikes it sends to the environment constitute the output of the system.

Primarily, [4] presented SN P systems as devices generating or accepting sets of natural numbers. Its universality was proven when no bound is imposed on the number of spikes. Later in [3], universality was proven even without the use of delays. Furthermore, the outdegree of neurons was reduced to the minimal bound of two. Next, SN P systems were found to be also universal when forgetting rules were removed and, finally, computational completeness was still achieved when using the simplest possible regular expressions \(\lambda\) and \(a^*\) over the alphabet \(\{a\}\) in firing rules. Another work [11] has also shown that not only the outdegree but also the indegree of neurons can be bound to two without losing universality.

In this paper we deal with some of the open problems stated in [3]. We show that SN P systems are still universal when eliminating delays while simplifying regular expressions. We have also proven the universality of the model in two more cases: (1) using simple regular expressions with strong halting condition and (2) using simple regular expressions with SN P systems working in the accepting mode.

## 2 DEFINITIONS

We consider the reader to be familiar with elements of membrane computing and language and automata theory. One can find in [12] the most updated information on membrane computing area.

Let \(V\) denote an alphabet, while \(V^*\) denotes the set of all finite strings of symbols from \(V\). The empty string is denoted by \(\lambda\) and we define \(V^+ = V^* \setminus \{\lambda\}\). The length of a string \(x \in V^*\) is denoted by \(|x|\). In the domain of SN P systems \(V = \{a\}\), i.e., the alphabet \(V\) is a singleton. Then \(a^*\) and \(a^+\) are normally used instead of \(\{a\}^*\) and \(\{a\}^+\).

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

\[
\Pi = (O, \sigma_1, \ldots, \sigma_m, \text{syn}, i_0),
\]

where:

1. \(O = \{a\}\) is the singleton alphabet (\(a\) is called spike);
2. σ₁, ..., σₘ are neurons, of the form

$$\sigma_i = (n_i, R_i), \quad 1 \leq i \leq m,$$

where:

(a) \( n_i \geq 0 \) is the initial number of spikes contained in \( \sigma_i \);

(b) \( R_i \) is a finite set of rules of the following two forms:

1. \( E/a^r \rightarrow a^t \), where \( E \) is a regular expression over \( a \), \( r \geq 1 \), and \( t \geq 0 \);

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

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

4. \( i_0 \in \{1, 2, \ldots, m\} \) indicates the output neuron (i.e., \( \sigma_{i_0} \) is the output neuron).

The rules of type (1) are firing rules (also called spiking rules). When the number of spikes present in a neuron is covered by the regular expression \( E \), the neuron gets fired, \( r \) spikes are consumed and, after \( t \) time steps, one spike is emitted by the neuron and replicated to all its neighbors. These spikes are available in the receiving neurons in the next step. In the interval \( t \) between getting fired and releasing the spike, the neuron is assumed to be closed – it omits any other spike received during this interval and, of course, it cannot fire. When \( E = a^r \), the rule is written as \( a^r \rightarrow a^t \); to simplify.

SN P systems use parallelism at the level of the system: in one step, all neurons that can use a rule have to use it. However, at most one (non-deterministically chosen) rule is used in each step by a neuron.

The rules of type (2) are called forgetting rules. They can be applied only when the neuron contains exactly \( s \) spikes which are then simply removed from the cell. The above definition does not allow nondeterminism between firing and forgetting rules, that is, in a single step a neuron has either to fire or forget, without being possible to freely choose between these two actions.

A computation of an SN P system is a sequence of steps during which rules are applied in the above described parallel manner. A computation starts in the initial configuration when each neuron \( \sigma_i \) contains \( n_i \) spikes, \( 1 \leq i \leq m \). A halting computation is that which reaches a state with no applicable rules. A computation is called strong halting if, in addition, no spike is present in the system when it halts.

The usual way of interpreting outputs of SN P systems is considering intervals between spikes of the output neuron \( i_0 \). Taking into consideration only the computations having exactly 2 output spikes (strong case), the set of
integers generated in this way is denoted by $N_2^\pi$. If only halting (strong halting) computations are taken into the account, we denote the resulting sets by $N_2^h(\pi)$ ($N_2^h(\pi)$, respectively). The reader will find in [10] and [3] several other variants of output. For further extensions and variants of spiking neural P systems we refer the reader e.g. to [1, 2, 8, 12].

We denote by $\text{Spik}_2^\beta P_m(\text{rule}_k, \text{cons}_p, \text{forg}_q, \text{dley}_r, \text{outd}_s)$ the family of all sets $N_2^\beta(\pi)$ (with $\beta \in \{h, h\}$), for all systems $\pi$ with at most $m$ neurons, each neuron having at most $k$ rules, at most $s$ outgoing synapses, each of the spiking rules consuming at most $p$ spikes, having a delay at most $r$, each of the forgetting rules removing no more than $q$ spikes. We also may write $\text{rule}_k^\lambda$ if the firing rules are of the form $E/a^r \rightarrow a; t$ with the regular expression of one of the forms $E = \lambda$ or $E = a^*$. The following result of [3] (Theorem 7.1) and its proof is frequently referred to in the next sections:

**Theorem 2.1.** $\text{Spik}_2^h P_4(\text{rule}_1^h, \text{cons}_2, \text{forg}_1, \text{dley}_2, \text{outd}_2) = \text{NRE}$, where either $\beta = h$ or $\beta$ is omitted.

The proof of this theorem is based on the construction of a P system simulating a register machine and using only neurons with regular expressions of the form $\lambda$ and $a^*$. The core idea of the proof is the construction of a module with dynamical circulation of spikes which simulates a register of a register machine. In this module, adding or removing of spikes circulating in a closed circuit is achieved with a proper timing of incoming spikes. These incoming spikes are emitted by modules associated with the ADD or SUB instructions of a register machine (see below).

A *nondeterministic register machine* [7] 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, $l_h$ is the halt label (assigned to the instruction HALT), and $I$ is the set of instructions. Each label from $H$ labels one instruction from $I$ (but the same instruction may be assigned to more labels). The instructions are of the following forms:

- $l_1 : (\text{ADD}(r), l_2, l_3)$ (add 1 to register $r$ and then go to one of the instructions labeled $l_2, l_3$),
- $l_1 : (\text{SUB}(r), l_2, l_3)$ (if register $r$ is non-empty, then subtract 1 from it and go to the instruction labeled $l_2$, otherwise go to the instruction labeled $l_3$),
- $l_h : \text{HALT}$ (the halt instruction).

In the generative version, a register machine starts with all registers empty (storing a zero) and applies the instructions in the order indicated by the labels. When the halt instruction is reached, the number $n$ stored in the first register is
considered the result of computation. The set of all numbers computed by the machine \( M \) is denoted by \( N(M) \). Register machines can generate in this way all recursively enumerable sets of nonnegative integers. Without loss of generality, we may assume that in the halting configuration, all registers except the first one are empty.

A register machine can also work in the accepting mode: a number \( n \) is introduced in the first register (all other registers being empty) at the start of computation. If the computation eventually halts, then the number \( n \) is accepted. Register machines are universal also in the accepting mode; moreover, this is true even for deterministic machines, having ADD rules of the form \( l_1 : (\text{ADD}(r), l_2, l_3) \) with \( l_2 = l_3 \) (in such a case, the instruction is written in the form \( l_1 : (\text{ADD}(r), l_2) \)). Again, without loss of generality, we may assume that in the halting configuration all registers are empty.

3 REMOVING DELAYS AND SIMPLIFYING REGULAR EXPRESSIONS

In this section we consider the same restrictions of regular expressions of a SN P system as in Theorem 2.1, but removing also delays simultaneously. Surprisingly, SN P systems are still universal even in that case. Moreover, except the non-deterministic neuron \( c_3 \) in the ADD module, each neuron will use one rule of the form \( a^r/a \to a \) or \( (a^s \to a) \), and at most two rules \( a^r \to \lambda \), with \( r, s \leq 3 \). Finally, the outdegree of each neuron is limited to two. Comparing this result with that of Theorem 2.1, one can notice that removing delays has some computational cost in terms of other parameters, as the maximum degree of forgetting rules, the number of rules per neuron and the maximum number of spikes consumed in a rule.

The main result on this section is based on the fact that neurons with delays, which make use of its refractory period to omit spikes received during this period, can be simulated by certain modules of neurons without delay (and with simple regular expression). The construction of these modules are given in the following lemma.

**Lemma 1.**

(i) The input–output behavior of a neuron with the single rule 
\( (a \to a; 1) \) is precisely simulated by the module \( \Pi_{d_1} \) in Figure 1.

(ii) The input–output behavior of a neuron with the single rule \( (a \to a; 2) \) 
is precisely simulated by the module \( \Pi_{d_2} \) in Figure 2.

**Proof.** Let us assume X emits two spikes consecutively to \( \Pi_{d_1} \) in \( t \) and \( t + 1 \). The spike received by \( c_1 \) and \( c_2 \) in \( t \) is emitted to \( c_3 \) in \( t + 1 \) (meanwhile the second spike emitted by X reaches \( c_1 \) and \( c_2 \)). Neuron \( c_3 \) fires, consuming just one of its two spikes, and spikes in \( t + 2 \). In that moment \( c_1 \) and \( c_2 \) also spike to \( c_3 \) which now contains three spikes, which are forgotten in \( t + 3 \) (using the rule \( a^3 \to \lambda \)).
FIGURE 1
Module $\Pi_{d_1}$ simulating a neuron with delay 1.

FIGURE 2
Module $\Pi_{d_2}$ simulating a neuron with delay 2.

By inserting another neuron $c_4$ with the rule $a \rightarrow a$ between $c_3$ and the output $Y$, and adding a feedback from $c_4$ to $c_3$, we obtain a module $\Pi_{d_2}$ simulating a neuron with delay 2. Its function is analogous to that of $\Pi_{d_1}$.

**Theorem 3.1.** $\text{Spik}_2^\beta P_s(\text{rule}_3, \text{cons}_3, \text{forg}_3, \text{dley}_0, \text{outd}_2) = \text{NRE}$, where either $\beta = h$ or $\beta$ is omitted.

**Proof.** This proof is based on the construction in the proof of Theorem 7.1 in [3]. In this construction we replace all the neurons having delays, depending on the case, by either a chain of basic neurons (those which use the delay just to postpone the emission of a spike) or by one of the modules $\Pi_{d_1}$ and $\Pi_{d_2}$.
In the proof of Theorem 7.1 of [3] the dynamic register stores a number equal to the number of spikes that are continuously circulating in the close circuit $r - s - t - u$. Due to the above explanation, we replace some of these neurons by modules $\Pi_{d_1}$ and $\Pi_{d_2}$ with two input neurons $c_1$ and $c_2$. Hence the input synapses to the registers have to be doubled. The need to maintain the outdegree $\leq 2$ forces us to replicate some cells (with their respective synapses) in order to keep the same functionality. Figure 3 shows the resulting dynamic register after introducing all these changes.

† if the register stores a number $n > 1$
†† if the register stores a number $n \leq 1$

FIGURE 3
A register with dynamical circulation of spikes without using delays.
The reader can note that one spike circulates from module $U$ to module $R$ (actually a group of 6 parallel spikes). Both $r_1$ and $r_2$ need to receive three spikes to ensure that $R$ spikes every three steps (as long as it holds any spike). Thus, to perform the ADD operation three spikes have also to be sent to both neurons $r_1$ and $r_2$. Module $R$ is connected to modules $S$ and $T$, as well. Hence, when $R$ spikes, one spike is sent simultaneously to neurons $s_1$, $s_2$, $t_1$ and $t_2$. Finally, a synapse also connects modules $S$ and $T$ with module $U$. In this case, three spikes have to be sent from each $S$ and $T$ to feed the three equal neurons in $U$.

Hence, the computation in our dynamic register is cyclic every six steps when it stores the value $n = 1$ (all 6 spikes present at step $t$ in $R$ are consumed to make it spike once to $S$ and $T$ at $t + 3$). In turn, $S$ and $T$ send three spikes (one to each neuron in $U$) at step $t + 5$. Finally, $U$ emits again six spikes to $R$ at step $t + 6$ and the cycle is complete. Moreover, the six-step computation cycle actually consists of two identical halves of three steps when the stored value is $n > 1$.

**Simulating an ADD instruction $l_i$:** $(\text{ADD}(r), l_j, l_k)$ (Figure 4).

To avoid the use of delays in the module ADD, we replace the former nondeterministic neuron $c_4$ by a new one $c_3$ containing three rules. At step one, neurons $l_i$, $l'_i$, $l''_i$ and $l'''_i$ send one spike to $c_1$ and $c_2$ and a group of 6 spikes to the module $R$ of the dynamic register (incrementing by one the stored value). In the next step, two spikes reach neurons $c_3$ and $c_4$. Then, if rule $a^2/a \rightarrow a$ of $c_3$ is chosen, one spike is sent to $c_6$ and $c_7$ while another one still remains in $c_3$. In the following step, $c_3$ uses its rule $a \rightarrow a$ and two more spikes arrive to $c_6$ and $c_7$ (one from $c_3$ and another from $c_5$). This makes $c_7$ fire (leading the computation to $l_k$) while $c_6$ forgets its three spikes. On the other hand, if $c_3$ first chooses rule $a^2 \rightarrow a$, then it just emits one spike to $c_6$ and $c_7$ which will receive another one from $c_5$ in the next step. This situation makes $c_6$ fire (leading now the computation to $l_j$) while $c_7$ forgets its two spikes. Finally, neurons $c_8$ and $c_9$ can be replaced by a chain of six basic zero-delay neurons.

**Simulating a SUB instruction $l_i$:** $(\text{SUB}(r), l_j, l_k)$ (Figure 5).

The module SUB contains only one neuron $c_5$ using its refractory period. It is replaced by a module of type $\Pi_d$. The rest of neurons that have delays are replaced by a chain of basic neurons with delay 0 (except, for the sake of clarity, in the case of $c_6$). Some neurons are also replicated in order to maintain outdegree $\leq 2$. This module is initiated when a spike is sent to neuron $l'_i$. Then, two spikes are sent to module $V$ at step three and the de-synchronization of the dynamic register starts, decrementing its value by 1. This forces $T$ to send a spike to $c_4$ at step 6. After that, $c_4'$ spikes at step 8 and the computation continues by instruction $l_j$. If the register stored zero, then neuron $c_4''$ do not spike at step 8 ($c_4'$ forgets the spike emitted by $c_1$ at 6). Then, two spikes reach
c_7 at step 11 and the computation continues by \( l_k \). It is important to note that if there exists more than one instruction SUB decrementing the same register, then we would need more connections from \( T \) to the neurons \( c_4 \) corresponding to these instructions.

**Ending a Computation** – module FIN (Figure 6).

The module FIN has also the same behavior as that of Theorem 7.1 in [3]. However, as in the case of module SUB, some structural changes have been made to eliminate the delays of neurons \( l_h, c_9, c_{10}, d_9 \) and \( d_{10} \). On the one hand, both neurons \( c_9 \) and \( d_9 \) are substituted by two basic cells with delay zero. On the other hand, as neurons \( c_{10} \) and \( d_{10} \) make use of their refractory period, we replace each of them by one of our modules of type \( \Pi_{d_1} \). Finally, neurons \( c_2, c_7, d_7, c_8 \) and \( d_8 \) are duplicated to keep outdegree \( \leq 2 \). Again, for the sake of clarity, we don’t replace \( c_3 \) with its corresponding four basic neurons.  \( \square \)
4 SIMPLIFIED REGULAR EXPRESSIONS REVISITED

Theorem 7.1 in [3] (recalled here as Theorem 2.1) considers neither the case of strong halting, nor the case of accepting SN P systems. This section extends Theorem 2.1 and shows that it remains valid even if these additional restrictions are imposed. First we deal with the strong halting case.

**Theorem 4.1.** $\text{Spik}^b_2 P_s (\text{rule}^s_2, \text{cons}_2, \text{forg}_2, \text{dley}_2, \text{outd}_2) = \text{NRE}$. 
Proof. Recall the assumption that all the registers of a register machine except register 1 are empty at the end of computation. Then one can verify by inspection of the proof of Theorem 7.1 in [3] that the only neurons containing spikes at the moment of halting are in the module FIN. The new module FIN which satisfies the strong halting condition is presented in Figure 7. Note that, to achieve this stronger condition (and simplify the construction), another restriction originally stated in [3] had to be released: the rules of the form $a' \rightarrow a; t$ and $a^r \rightarrow \lambda$ in the same neuron satisfy $s < r$.

Let the output register 1 hold a value $n > 1$ and hence the cycle of neurons 1 and $c_1$ contain $n$ spikes ($n - 1$ in neuron 1 and one spike in neuron $c_1$). Both neurons fire at every step.

FIGURE 6
Module FIN ending the computation.
At step 1 neuron $l_h$ receives one spike and fires. The output neuron out fires the first time at step 8, receiving a spike from neuron $c_5$. At step 3 both neurons $c_2$ and $c_3$ receive spikes and start to fire at every step. Neuron $c_4$ receives at every step two spikes which are removed. Simultaneously from step 5 the neuron $c_1$ starts to receive spikes from neuron $c_7$ which decreases the number of spikes in the cycle $1 - c_1$ by one at every step. (Spikes are consumed in neuron $c_1$.) After $n$ steps all spikes in the cycle $1 - c_1$ are removed. At step $n + 5$ neuron $c_2$ receives no spike and does not fire. Consequently, at step $n + 7$ neuron $c_4$ receives only one spike from neuron $c_8$ and fires. Finally, at step $n + 8$, exactly $n$ steps after its first firing, neuron out fires for the second time.

After removing all spikes in the cycle $1 - c_1$, neuron $c_7$ sends two more spikes to $c_1$. To remove these two spikes which start to circulate in the cycle $1 - c_1$ is the role of neurons $c_9$ and $c_{10}$. They send two additional spikes to neuron $c_1$ which, thanks to the rule $a^2 \rightarrow \lambda$, clear the cycle $1 - c_1$ at the end of computation.

In the case $n = 1$ neuron 1 fires at every even step and neuron $c_1$ fires at every odd step. To remove this spike, neuron $c_1$ must receive its first spike from $c_7$ at an odd step. Going back in the cascade, neuron $l_h$ must receive spike at an odd step, too. Indeed, in the proof of Theorem 7.1 in [3], each instruction of the register machine is simulated in exactly 6 steps of the SNP system. Hence neuron $l_h$ receives spike at a step $6k + 1$, $k \geq 0$, and the module FIN works correctly also in this case.

Another extension of the previously known normal forms is the case of accepting SNP systems. In the accepting mode [4], the SNP system obtains...
an input in the form of an interval between two consecutive spikes sent from outside to the input neuron \( i_0 \). Therefore, we need a module INPUT which translates this input value into the number of spikes present in a neuron labeled 1. Furthermore, the SN P system must behave deterministically. Both conditions can be satisfied and the resulting statement is given below.

**Theorem 4.2.** \( \text{DSpik}^\beta_{2\text{acc}}(\text{rule}^*_{2}, \text{cons}^2_{2}, \text{forg}_\alpha_{2}, \text{dley}_{2}, \text{outd}_{2}) = \text{NRE} \) where (i) \( \beta = h, \alpha = 1 \), or (ii) \( \beta = h, \alpha = 2 \), or (iii) \( \beta \) is omitted and \( \alpha = 1 \).

**Proof.** Considering the proof of Theorem 7.1 in [3], we can observe that the module SUB is already deterministic. Then it remains only to “determinize” the module ADD by removing the rule \( a \rightarrow a; 1 \) from neuron \( c_{14} \) in the above mentioned module ADD, and the whole module becomes deterministic.

It remains to construct the above described module INPUT which would pass the number of spikes corresponding to the value of input to the module ADD attached to register 1. However, as the module ADD works in a synchronized cycle of the length three, we have to send spikes to register 1 each three computational steps (or its multiple). Otherwise the spikes might be lost (consumed) within the module ADD.

The module INPUT solving this task is presented in Figure 8. Three steps after neuron \( i_0 \) receives the first input spike, neurons \( c_3 - c_6 \) start to fire at each step. Similarly, three steps after neuron \( i_0 \) receives the second input spike, neurons \( c_3 - c_6 \) stop firing and remove all their spikes. Therefore, neuron \( c_7 \)

![Diagram](image-url)  
**FIGURE 8**  
The Module INPUT with simplified regular expressions.
will receive exactly $3n$ spikes, where $n$ is the period between the first and the second input spike. Neuron $c_7$ emits one spike at each step but neuron $c_8$ lets pass only each third spike. Hence neuron 1 corresponding to the input register receives spikes each three steps and the number of spikes is exactly $n$.

Finally, observe that there are no spikes left in neurons after finishing the computation. As this is also the case in the modules ADD and SUB described in the proof of Theorem 7.1 in [3] (provided that all the registers are empty), the system halts always in the strong halting state. If we did not require strong halting, the rules $a_2 \rightarrow \lambda$ in neurons $c_3 - c_6$ could be omitted. Therefore, the parameter $\text{forg}$ would be reduced to 1 in this case.

Therefore, the described SN P system correctly simulates a register machine in the accepting mode and the inclusion $NRE \subseteq DSpik_{\text{acc}}^b \times (\text{rule}_2^*, \text{cons}_2, \text{forg}_2, \text{dlev}_2, \text{outd}_2)$ holds. The converse inclusion follows by the Church-Turing thesis.

5 FINAL REMARKS

In this paper we have proven that SN P systems remain computationally universal even when (1) simplifying regular expressions and eliminating delays, (2) using simple regular expressions and the strong halting condition and (3) using simple regular expressions in the accepting mode. We conjecture that in the cases (2) and (3), also delays could be removed without loss of computational universality. The cases of simultaneously removing delays and forgetting rules, or removing forgetting rules and simplifying regular expressions remain open.

In all these results one can observe a trade-off between some other computational parameters, such as the number of neurons, the maximum number of firing rules per neuron, the complexity of regular expressions, the maximum number of spikes consumed in a firing rule or the maximum number of spikes removed in a forgetting rule. In all the above mentioned cases, however, the outdegree of neurons has been bounded by two.

It remains an open problem whether these results can be further improved. Would it be possible, for instance, to eliminate some more features of the model (three of them simultaneously) while keeping universality? If not, which would be the computational power of such a restricted model? Another open question would be, naturally, whether we can still achieve lower bounds for some of the computational parameters in our current proofs, as the number of rules in neurons, number of spikes consumed in one rule etc.

ACKNOWLEDGEMENTS

This research has been partially funded by the Spanish Ministry of Science and Education under projects TIN2006-15595 and DEP2005-00232-C03-03,
REFERENCES
