Available online at www.sciencedirect.com
ScienceDirect
BioSystems 90 (2007) 48–60
BioSystems
www.elsevier.com/locate/biosystems

Small universal spiking neural P systems

Andrei Păun[a], Gheorghe Păun[b,c,*]

[a] Department of Computer Science, Louisiana Tech University, PO Box 10348, Ruston, LA 71272, USA
[b] Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 Bucharest, Romania
[c] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda Reina Mercedes s/n, 41012 Sevilla, Spain

Received 27 March 2006; received in revised form 22 June 2006; accepted 22 June 2006

Abstract

In search for small universal computing devices of various types, we consider here the case of spiking neural P systems (SN P systems), in two variants: as devices that compute functions and as devices that generate sets of numbers. We start with the first case and we produce a universal spiking neural P system with 84 neurons. If a slight generalization of the used rules is adopted, namely, we allow rules for producing simultaneously several spikes, then a considerable reduction, to 49 neurons, is obtained. For SN P systems used as generators of sets of numbers, we find a universal system with restricted rules having 76 neurons and one with extended rules having 50 neurons.

Keywords: Membrane computing; Spiking neural P system; Universality; Register machine

1. Introduction

Looking for small universal computing devices is a natural and well investigated topic in computer science, see e.g. Korec (1996) and Rogozhin (1996), and the references therein. Recently, this issue started to be considered also in membrane computing; first contributions of this kind can be found in Rogozhin and Verlan (2006) (for tissue P systems with string objects processed by splicing operations) and Csuhaj-Varjú et al. (in press) (for symport/antiport P systems).

In the present paper, we address the case of the recently introduced (see Ionescu et al., 2006; Ibarra et al., in press; Păun et al., 2006) *spiking neural P systems* (in short, SN P systems), computing models which bring into membrane computing (see Păun, 2002 for an introduction and http://www.psystems.disco.unimib.it for updated information) ingredients from neural computing by spiking (see e.g. Maass, 2002; Maass and Bishop, 1999).

In short, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph (whose arcs represent *synapses*) and sending to each other *spikes*, identical electrical impulses. The distance between consecutive spikes is the main way to encode information. The neurons contain rules for emitting spikes and for forgetting spikes. One neuron is distinguished as the *output* neuron and its spikes also exit into the environment, thus producing a *spike train*.

Such systems can be used as computing devices in various ways; generating sets of numbers (encoded in the number of steps between consecutive spikes sent into the environment by the output neuron), generating strings (the spike train itself is such a string over the binary alphabet), or computing functions (an input neuron "reads" the arguments of the function from the environment and the output neuron "writes" the function value, in all cases with the numbers being encoded in

* Corresponding author.
E-mail addresses: apaun@latech.edu (A. Paun), george.paun@imar.ro, gpaun@us.es (Gheorghe P).

0303-2647/$ – see front matter © 2006 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2006.06.006

the distance between consecutive spikes). More precise definitions will be given in Section 2.2.

Here we deal with the first and the third cases, namely with SN P systems that generate numbers and compute functions.

Already in Ionescu et al. (2006), the SN P systems used for computing sets of numbers were proved to be computationally complete (able to compute all Turing computable sets of numbers), but no bound on the number of used neurons was found. The proof from Ionescu et al. (2006) is based on simulating register machines with SN P systems, hence the same strategy as that followed in Csuhaj-Varjú et al. (in press) in search of small universal P systems is useful also here: starting from a small universal register machine as those constructed in Korec (1996), we can get a small universal SN P system, with the important mentioning that in Korec (1996) one works with register machines that compute functions. That is why we also start here with this case.

We work with the so-called strong universality (without encodings of the input and output) and we use the standard type of register machines (using ADD and SUB instructions). For this case, a universal register machine with 8 registers and 23 instructions (the halting one included) was constructed in Korec (1996). (The "code" of the particular partial recursive function and the argument of the function are introduced in registers 1 and 2 of the universal machine and the value of the function for that argument, if defined, is found in register 0 when/if the machine halts.) Following then the construction of an SN P system that simulate a register machine from Ionescu et al. (2006), with some improvements inspired from Ibarra et al. (in press) and some additional "code optimization", as well as a suitable INPUT module, we get an SN P system with 84 neurons simulating the register machine from Korec (1996).

We may formulate this result as follows: there is a universal "brain" (in the form of an SN P system) with only 84 neurons.

Of course, this number needs to be checked for optimality, but it is our expectation that in the framework used here (with the type of SN P systems considered, i.e. with standard rules) it is not possible to significantly decrease the number of neurons.

However, if we allow rules which produce two or more spikes, then a considerable improvement of the previous result is obtained: 49 neurons are sufficient for universality.

We consider then the case when SN P systems are used for generating sets of numbers, starting from the observation that a set $Q \subseteq N$ is recursively enumerable if and only if the characteristic function of Q (equal to 1 for elements of Q and undefined otherwise) is a partial recursive function. Similar results as for the previous case are obtained: 76 neurons are sufficient for universality when using restricted rules and 50 when using extended rules.

In the next section we introduce all necessary prerequisites related to register machines, universality, and standard SN P systems. Section 3 gives the first universal SN P system, and in Section 4 we introduce the extended spiking rules and we produce the universal computing SN P system with 49 rules. In Section 5 we consider the case of universal SN P systems that work as generators of sets of numbers.

2. Prerequisites

The reader is assumed to have some elementary knowledge in theoretical computer science as available from the many monographs in the field (e.g. from van Leeuwen, 1990; Wood, 1987), so that we specify here only some notations and basic definitions.

For an alphabet V, V^* denotes the set of all strings over V, with the empty string denoted by λ.

A *regular expression* over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. Clearly, we assume that the parentheses are not in V; as a matter of fact, we will often omit "unnecessary parentheses". Also, $E_1^+ \cup \lambda$ can be written as E_1^*. With each expression E we associate its language $L(E)$ as follows: (i) $L(\lambda) = \{\lambda\}$, $L(a) = \{a\}$, for $a \in V$, (ii) $L((E_1)(E_2)) = L(E_1)L(E_2)$, $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, and $L((E_1)^+) = L(E_1)^+$, for all regular expressions E_1, E_2.

The operations used here are the standard union, concatenation, and Kleene +. We also need below the operation of *right derivative* of a language $L \subseteq V^*$ with respect to a string $x \in V^*$, which is defined by $L/x = \{y \in V^* | yx \in L\}$.

We proceed now to introducing the necessary notions related to register machines and universality, then the spiking neural P systems.

2.1. Universal register machines

We use here register machines given in the form $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H the set of instruction labels, l_0 the start label, l_h the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H is associated with