How Redundant Is Your Universal Computation Device?

Alberto Leporati, Claudio Zandron, and Giancarlo Mauri

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,zandron,mauri}@disco.unimib.it

Abstract. Given a computational model $\mathcal{M}$, and a "reasonable" encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ that encodes any computation device M of $\mathcal{M}$ as a finite bit string, we define the *description size* of M (under the encoding $\mathcal{C}$) as the length of $\mathcal{C}(M)$. The description size of the entire class $\mathcal{M}$ (under the encoding $\mathcal{C}$) can then be defined as the length of the shortest bit string that encodes a *universal* device of $\mathcal{M}$. In this paper we propose the description size as a complexity measure that allows to compare different computational models. We compute upper bounds to the description size of deterministic register machines, Turing machines, spiking neural P systems and UREM P systems. By comparing these sizes, we provide a first partial answer to the following intriguing question: what is the minimal (description) size of a universal computation device?

1 Introduction

Looking for small universal computing devices is a natural and well investigated topic in computer science: see e.g., [24,12,20,21,22,9,10] for classic computational models, [23,6,4] for tissue and symport/antiport P systems, [26,3] for cellular automata, and [7,17] for spiking neural P systems.

A related question that we investigate in this paper is: What is the size of the *smallest* among all possible universal computation devices? Of course we must agree on the meaning of the term "size", since the size of a given device may depend on several parameters (for example, the number of registers and the number of program instructions when speaking of register machines), whose number and possible values vary depending on the device under consideration. Trying to find a common unit to measure the size of different computation devices, in section 3 we will define the *description size* of a computation device as the number of bits which are needed to describe it. Precisely, for a given model of computation $\mathcal{M}$ (for example, register machines), we will define an encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ that associates a bit string to every computing device M taken from $\mathcal{M}$; the description size $ds_{\mathcal{C}}(M)$ of M (under the encoding $\mathcal{C}$) will be the length of the bit string $\mathcal{C}(M)$, whereas the description size of the entire class $\mathcal{M}$ will be the minimum between the description sizes $ds_{\mathcal{C}}(M)$ for M that varies over the set of *universal* computing

D. Corne et al. (Eds.): WMC9 2008, LNCS 5391, pp. 274–291, 2009.

devices contained in $\mathcal{M}$. Then, we start a quest for the shortest possible bit string that describes a universal computation device: that is, we look for a computational model $\mathcal{M}$ and an encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ such that $\mathcal{M}$ contains at least one universal computation device and $ds_{\mathcal{C}}(\mathcal{M})$ is as low as possible. To this aim, we will compute the description size of randomly generated deterministic register machines, Turing machines, spiking neural P systems [8] and UREM P systems [5], as well as the size of a specific *small* universal instance of each of these computational models. Then, by taking deterministic register machines as the reference model, we will compute the redundancy of the other computing devices here considered as the ratio between their description size and the description size of the smallest (to the best knowledge of the authors) universal deterministic register machine currently known. As a result, we will have an idea about how verbose are such models of computation in catching the notion of universality.

A word of caution is due: with our work, we are not saying that the most compact computational model is the best: a computation device that requires a lot of features to perform its computations may be more interesting than others because of many reasons. A notable example is given by traditional P systems [18,19], whose structure and behavior are inspired from the functioning of living cells; the amount of theoretical results and applications reported in the bibliography of [27] is certainly an indication of how interesting is such a model of computation.

The paper is structured as follows. In Section 2 we briefly recall the definition of the computational models we will work upon: deterministic register machines, spiking neural P systems and UREM P systems. Since several variants of Turing machines have been defined in the literature, we will later refer the reader to the bibliography for the details on the these machines. In Section 3 we will define and compute the description size of *randomly chosen* instances of all these models. We will also consider a small universal instance (taken from the literature) of each of these models, and we will compute both its description size and the redundancy with respect to the smallest (to the best knowledge of the authors) currently known deterministic register machine. In Section 4 we draw some conclusions and we propose some directions for future research.

2 Some (Universal) Models of Computation

2.1 Deterministic Register Machines

A *deterministic n–register machine* is a construct $M = (n, P, m)$, where $n > 0$ is the number of registers, P is a finite sequence of instructions bijectively labeled with the elements of the set $\{0, 1, \ldots, m-1\}$, 0 is the label of the first instruction to be executed, and $m-1$ is the label of the last instruction of P. Registers contain non-negative integer values. The instructions of P have the following forms:

- $j : (INC(r), k)$, with $j, k \in \{0, 1, \ldots, m-1\}$ and $r \in \{0, 1, \ldots, n-1\}$
  This instruction, labeled with j, increments the value contained in register r, and then jumps to instruction k.

– $j : (DEC(r), k, l)$, with $j, k, l \in \{0, 1, \ldots, m-1\}$ and $r \in \{0, 1, \ldots, n-1\}$
If the value contained in register r is positive then decrement it and jump to instruction k. If the value of r is zero then jump to instruction l (without altering the contents of the register).

Computations start by executing the first instruction of P (labeled with 0), and terminate when the instruction currently executed tries to jump to label m.

For a formal definition of *configurations* and *computations* of M we refer the reader to [5]. Here we just recall that deterministic register machines provide a simple universal computational model, as stated in [5, Proposition 1].

2.2 Spiking Neural P Systems

Spiking neural P systems (SN P systems, for short) have been introduced in [8] as a class of synchronous, parallel and distributed computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses along axons to other neurons.

Formally, a *spiking neural membrane system* (SN P system, for short) of degree $m \geq 1$, as defined in [7] in the computing version (i.e., able to take an input and provide and output), is a construct of the form $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ 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) *firing* (also *spiking)* rules $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers;
      (2) *forgetting* rules $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (the regular language defined by E);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π, respectively.

A firing rule $E/a^c \rightarrow a; d \in R_i$ can be applied in neuron σ_i if it contains $k \geq c$ spikes, and $a^k \in L(E)$. The execution of this rule removes c spikes from σ_i (thus leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$ then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. During these d computation steps 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 that particular spike is lost), and cannot fire (and even select) rules. A *forgetting* rule $a^s \rightarrow \lambda$ can be applied in neuron σ_i if it contains *exactly* s spikes; the execution of this rule simply removes all the s spikes from σ_i.

Extended rules provide a common generalization of firing rules. These rules are of the form $E/a^c \rightarrow a^p; d$, where $c \geq 1$, $p \geq 1$ and $d \geq 0$ are integer numbers.

The semantics of these rules is the same as above, with the difference that now p spikes are delivered (after d time steps) to all neighboring neurons.

We refer the reader to [7] for a formal definition of *configurations* and *computations*; here we just recall that (many variants of) SN P systems have proven to be universal. In what follows we will use the two small deterministic universal SN P systems which are defined in [17]: one of them uses 84 neurons, each one containing only standard rules; the other uses 49 neurons, but in this case extended rules are needed.

2.3 UREM P Systems

P systems with unit rules and energy assigned to the membranes (UREM P systems, for short) have been introduced in [5] as a variant of P systems in which a non–negative integer value (regarded as an amount of energy) is assigned to each membrane of the system. The rules are assigned to the membranes rather than to the regions of the system, and operate like filters that control the movement of objects (symbols of an alphabet) across the membranes.

Formally, a UREM P system [5] of degree $d+1$ is a construct Π of the form $\Pi = (A, \mu, e_0, \ldots, e_d, w_0, \ldots, w_d, R_0, \ldots, R_d)$, where:

- A is an alphabet of *objects*;
- μ is a *membrane structure*, with the membranes labeled by numbers $0, \ldots, d$ in a one-to-one manner;
- $e_0, \ldots, e_d$ are the initial energy values assigned to the membranes $0, \ldots, d$. We assume that $e_0, \ldots, e_d$ are non–negative integers;
- $w_0, \ldots, w_d$ are multisets over A associated with the regions $0, \ldots, d$ of μ;
- $R_0, \ldots, R_d$ are finite sets of *unit rules* associated with the membranes $0, \ldots, d$. Each rule of R_i has the form $(\alpha_i : a, \Delta e, b)$, where $\alpha \in \{in, out\}$, $a, b \in A$, and $|\Delta e|$ is the amount of energy that — for $\Delta e \geq 0$ — is added to or — for $\Delta e < 0$ — is subtracted from e_i (the energy assigned to membrane i) by the application of the rule.

A computation step is performed by *non-deterministically* choosing one rule from some R_i and applying it (hence in a *sequential* way, as opposed to the maximally parallel way often required in P systems). Applying $(in_i : a, \Delta e, b)$ means that an object a (being in the membrane immediately outside of i) is changed into b while entering membrane i, thereby changing the energy value e_i of membrane i by Δe. On the other hand, the application of a rule $(out_i : a, \Delta e, b)$ changes object a into b while leaving membrane i, and changes the energy value e_i by Δe. The rules can be applied only if the amount e_i of energy assigned to membrane i fulfills the requirement $e_i + \Delta e \geq 0$. Moreover, a sort of local priorities is assumed: if there are two or more applicable rules in membrane i, then one of the rules with $\max |\Delta e|$ has to be used.

If we consider the distribution of energy values among some predefined membranes as the input to be processed and the resulting output (a non–halting computation does not produce a result) we obtain a universal model of computation, as proved in [5, Theorem 1]. The proof is obtained by simulating deterministic register machines by *deterministic* UREM P systems which contain

one elementary membrane into the skin for each register of the simulated machine. The contents of each register are expressed as the energy value assigned to the corresponding membrane. A single object is present in the system at every computation step, which stores the label of the instruction of the program P currently simulated. Increment instructions of the kind $j : (INC(i), k)$ are simulated in two steps by using the rules $(in_i : p_j, 1, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, 0, p_k)$. Decrement instructions of the kind $j : (DEC(i), k, l)$ are also simulated in two steps, by using the rules $(in_i : p_j, 0, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, -1, p_k)$ or $(out_i : \widetilde{p_j}, 0, p_l)$. The use of priorities associated to these last rules is *necessary* to correctly simulate a decrement instruction, and hence to reach the computational power of Turing machines, as proved in [5, Theorem 2].

3 Description Size

As stated in the Introduction, we define the *description size* of a given computation device M as the length of the binary string which encodes the structure of M. Since this is an informal definition, we have to discuss some technical difficulties that immediately arise. First of all, for any given computational model $\mathcal{M}$ (register machines, SN P systems, etc.) we have to find a "reasonable" encoding function $\mathcal{C} : \mathcal{M} \to \{0, 1\}^*$, in the sense given in [2]. Such a function should be able to encode *any* computation device M of $\mathcal{M}$ as a finite bit string. When this string is interpreted (that is, decoded) according to a specified set of rules (the *decoding algorithm*), the decoder unambiguously recovers the structure of M. In order to avoid cheating — by hiding information into the encoding or decoding algorithms — we ask to consider only reasonable encodings that satisfy the following requirements.

1. For each model of computation, the encoding and decoding algorithms are fixed a priori, and their representation as a program for a deterministic register machine or as a deterministic Turing machine have a fixed *finite length*. Note that, when computing the description size of a given device, we will not count the size of the encoding and decoding algorithms; moreover, instead of formally specifying such algorithms, we will only provide *informal* instructions on how to encode and decode our computation devices. An alternative approach, not followed in this paper, consists of minimizing the size of the decoding (and, possibly, encoding) algorithm together with the length of the encoded strings.
2. With the selected encoding algorithm it should be possible to describe *any* instance of the computational model under consideration (for example, any deterministic register machine). Encodings that allow to represent in a very compact form only one or a few selected instances of the computational model (for example, all the register machines whose program P contains exactly five instructions) are not considered acceptable.

We can look at any computational model as a family of computation devices, whose size depends upon a predefined collection of parameters. For example, the

class of all deterministic register machines is composed by machines which have n registers and whose programs are composed by m instructions, for all possible integers $n \geq 1$ and $m \geq 0$. Denoted by $\mathcal{M}$ a computational model, and by $(n_1, n_2, \ldots, n_k)$ the non–negative integer parameters upon which the size of the computing devices of $\mathcal{M}$ depend, we can write $\mathcal{M} = \bigcup_{n_1,\ldots,n_k} \{M(n_1, \ldots, n_k)\}$, where $M(n_1, \ldots, n_k)$ is the subclass of $\mathcal{M}$ that is composed of those devices $M \in \mathcal{M}$ for which the parameters have the indicated values $n_1, \ldots, n_k$. An encoding function $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ can thus be viewed as a family $\{\mathcal{C}(n_1, \ldots, n_k)\}_{n_1,\ldots,n_k}$, where the function $\mathcal{C}(n_1, \ldots, n_k)$ encodes any instance $M \in M(n_1, \ldots, n_k)$. Note that the values $n_1, \ldots, n_k$ need not to be encoded, since they can be determined by the sub-function of $\mathcal{C}$ we are using. In what follows we will propose specific encodings $\mathcal{C}(n_1, \ldots, n_k)$ for each of the computational models considered in this paper, both for generic and for specific values of $n_1, \ldots, n_k$; with a little abuse of notation, we will sometimes indicate the functions $\mathcal{C}(n_1, \ldots, n_k)$ as the encodings (that is, $\mathcal{C}$) of our models.

Let $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ denote a fixed encoding of $\mathcal{M}$, and let M be a computation device from $\mathcal{M}$. By $ds_{\mathcal{C}}(M) = |\mathcal{C}(M)|$ (the length of $\mathcal{C}(M)$) we will denote the description size of M, obtained by using the encoding $\mathcal{C}$, and by $ds_{\mathcal{C}}(\mathcal{M})$ we will denote the length of the most compact representation — produced by the encoding algorithm of $\mathcal{C}$ — of a *universal* computing device taken from the class $\mathcal{M}$, that is, $ds_{\mathcal{C}}(\mathcal{M}) = \min\{ds_{\mathcal{C}}(M) \ : \ M \in \mathcal{M} \text{ is universal}\}$. By definition, for any fixed universal computing device $M \in \mathcal{M}$ the value $ds_{\mathcal{C}}(M)$ is an upper bound for $ds_{\mathcal{C}}(\mathcal{M})$. We say that a universal computing device $M^* \in \mathcal{M}$ is *optimal* (referred to the description size, for a prefixed encoding $\mathcal{C}$) if $ds_{\mathcal{C}}(M^*) = ds_{\mathcal{C}}(\mathcal{M})$. Given two classes of computation devices $\mathcal{M}$ and $\mathcal{M}'$ (with possibly $\mathcal{M} = \mathcal{M}'$), we define the *redundancy* of a universal computation device $M' \in \mathcal{M}'$, with respect to the computational model $\mathcal{M}$ and the encoding $\mathcal{C}$, as $R_{\mathcal{M},\mathcal{C}}(M') = \frac{ds_{\mathcal{C}}(M')}{ds_{\mathcal{C}}(\mathcal{M})}$. Similarly, we define the redundancy of a computational model $\mathcal{M}'$ (with respect to $\mathcal{M}$ and $\mathcal{C}$) as $R_{\mathcal{M},\mathcal{C}}(\mathcal{M}') = \frac{ds_{\mathcal{C}}(\mathcal{M}')}{ds_{\mathcal{C}}(\mathcal{M})}$. Finally, by letting $\mathcal{C}$ vary on the class of all possible "reasonable" encodings, for any computational models $\mathcal{M}$ and $\mathcal{M}'$ we can define:

$$ds(\mathcal{M}) = \min_{\mathcal{C}}\{ds_{\mathcal{C}}(\mathcal{M})\} \qquad \text{and} \qquad R_{\mathcal{M}}(\mathcal{M}') = \frac{ds(\mathcal{M}')}{ds(\mathcal{M})}$$

that is, the description size complexity of $\mathcal{M}$ and the redundancy of $\mathcal{M}'$ with respect to $\mathcal{M}$, respectively.

Let us note that the quantities $ds(\mathcal{M})$ and $ds_{\mathcal{C}}(\mathcal{M})$, for some fixed computational model $\mathcal{M}$ and encoding $\mathcal{C}$, may be difficult to find, as it usually happens with theoretical bounds. Hence in general we will obtain upper bounds to these quantities, and thus lower bounds for the corresponding redundancies. The final goal of the research line set out with this paper is to find a universal computational class $\mathcal{M}$ and an encoding $\mathcal{C} : \mathcal{M} \to \{0,1\}^*$ whose description size $ds_{\mathcal{C}}(\mathcal{M})$ is as low as possible, and eventually an optimal instance $M \in \mathcal{M}$. In this way, no other model $\mathcal{M}'$ that contains a universal computation device would have $ds(\mathcal{M}') < ds_{\mathcal{C}}(\mathcal{M})$, and hence the value $ds_{\mathcal{C}}(M) = ds_{\mathcal{C}}(\mathcal{M})$ could be regarded

as the *description size complexity* of *universality*: in other words, it would be the minimal number of bits which are needed to describe the ability to compute Turing computable (that is, partial recursive) functions.

In the next subsections we will compute the description size of *randomly generated* computing devices taken from each of the classes mentioned in section 2: deterministic register machines, Turing machines, SN P systems and UREM P systems (each with respect to an appropriate predefined encoding). Then, we will also compute the description size of a small universal device taken from each of these classes, thus providing upper bounds to the description sizes of the whole classes.

In what follows, for any natural number n we will simply denote by $\lg n$ the number $\lfloor \log_2 n \rfloor + 1$ of bits which are needed to represent n in binary form.

3.1 Deterministic Register Machines

Denoted by $\mathrm{DRM}(n, m)$ the subclass of register machines that have n registers and programs composed of m instructions, we can express the class DRM of deterministic register machines as: $\mathrm{DRM} = \bigcup_{n \geq 1, m \geq 0} \mathrm{DRM}(n, m)$. Let us describe a function $\mathcal{C}(n, m)$ that encodes any machine from the subclass $\mathrm{DRM}(n, m)$.

Let $M \in \mathrm{DRM}(n, m)$, and let P denote its program. To describe each instruction of P, we need 1 bit to say whether it is an INC or a DEC instruction, $\lg n$ bits to specify the register which is affected, and $\lg m$ bits (resp., $2 \cdot \lg m$ bit) to specify the jump label (resp., the two jump labels) for the INC (resp., DEC) instruction. A simple encoding of M is a sequence of m blocks, each composed of $1 + \lg n + \lg m$ or $1 + \lg n + 2 \cdot \lg m$ bits, encoding the corresponding instruction of P.

If M is a *randomly chosen* (and thus, possibly, non–universal) machine, then about half of the instructions of P will be INCs and half will be DECs; hence the description size of M, with respect to the encoding $\mathcal{C}$ we have just defined, will be:

$$\begin{aligned} ds_{\mathcal{C}}(M) &= \frac{m}{2} \left[1 + \lg n + \lg m\right] + \frac{m}{2} \left[1 + \lg n + 2 \cdot \lg m\right] \\ &= m \cdot \left[1 + \lg n\right] + \frac{3m}{2} \lg m \end{aligned}$$

In order to compute an upper bound to $ds(DRM)$ we have instead to restrict our attention to *universal* register machines. In [9], several universal register machines are described and investigated. In particular, the *small* universal register machine illustrated in Figure 1 is defined. This machine has $n = 8$ registers and $m = 22$ instructions. However, recall that we need a further label (22) to halt the execution of P anytime by simply jumping to it, and thus we put $m = 23$. The number of bits required to store these values are 3 and 5, respectively. The encoding of this machine produces a bit string which is composed of 22 blocks, one for each instruction of P. Each register will require 3 bits to be specified, and each label will require 5 bits. If we denote INC instructions by a 0, and DEC instructions by a 1, then the first block will be 1 001 00001 00010, where

$0 : (DEC(1), 1, 2)$ $1 : (INC(7), 0)$
$2 : (INC(6), 3)$ $3 : (DEC(5), 2, 4)$
$4 : (DEC(6), 5, 3)$ $5 : (INC(5), 6)$
$6 : (DEC(7), 7, 8)$ $7 : (INC(1), 4)$
$8 : (DEC(6), 9, 0)$ $9 : (INC(6), 10)$
$10 : (DEC(4), 0, 11)$ $11 : (DEC(5), 12, 13)$
$12 : (DEC(5), 14, 15)$ $13 : (DEC(2), 18, 19)$
$14 : (DEC(5), 16, 17)$ $15 : (DEC(3), 18, 20)$
$16 : (INC(4), 11)$ $17 : (INC(2), 21)$
$18 : (DEC(4), 0, 22)$ $19 : (DEC(0), 0, 18)$
$20 : (INC(0), 0)$ $21 : (INC(3), 18)$

Fig. 1. The small universal deterministic register machine defined in [9]

we have put a small space to make clear how the block is formed: the first 1 denotes a DEC instruction, which has to be applied to register number 1 (= 001), and the two labels to jump to when we have executed the instruction are 1 (= 00001) and 2 (= 00010). Similarly, the block that encodes the second instruction is 011100000 (here we have omitted the unnecessary spaces), whereas the string that encodes the whole machine M is:

10010000100010 011100000 011000011 11010001000100
11100010100011 010100110 11110011101000 000100100
11100100100000 011001010 11000000001011 11010110001101
11010111001111 10101001010011 11011000010001 10111001010100
010001011 001010101 11000000010110 10000000010010
000000000 001110010

Here the spaces denote a separation between two consecutive blocks; of course these spaces are put here only to help the reader, but are not necessary to decode the string. We can thus conclude that, referring to the encoding $\mathcal{C}$ given above:

$$ds_{\mathcal{C}}(M) = 14 * 13 + 9 * 9 = 182 + 81 = 263$$

Since our final goal is to find the shortest bit string that encodes a universal computation device, we could wonder how many bits we would save by *compressing* the above sequence. This means, of course, that the encoding algorithm will have to produce a compressed representation of M, that will be decompressed by the decoding algorithm. Many compression algorithms exist, that yield different results. For simplicity here we just consider entropy-based compressors, such as the Huffman algorithm, and we compute a bound on the length of the compressed string. If we look at the above bit string, we can see that it contains 154 zeros and 109 ones. Hence in each position of the string we have the probability $p_0 = \frac{154}{263}$ that a 0 occurs, and the probability $p_1 = \frac{109}{263}$ that a 1 occurs. By

looking at the output of the encoding algorithm as a *memoryless* information source, we can compute the entropy of the above sequence, that measures the average amount of information carried by each bit of the string:

$$H(M) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \approx 0.979$$

Now, by applying an optimal entropy-based compressor we would obtain a compressed string whose length is approximately equal to the length of the uncompressed string times the entropy, that is, $\lceil 263 \cdot 0.979 \rceil = 258$ bits. Such a quantity is less than 263, but of course is still an upper bound to $ds(\mathrm{DRM})$, the (unknown, and possibly very difficult to determine) description size complexity of deterministic register machines.

As stated above, the choice of a different category of (lossless) compression algorithms may yield to different string lengths. Moreover, even if we restrict our attention to entropy-based compressors the fact that we have modeled the output of the encoding algorithm as a memoryless information source is questionable. In fact, when we encounter a 0 at the beginning of a new block that encodes an instruction of the register machine then it is clear that 8 bits will follow, instead of 13, since we are reading an INC instruction. This means that the occurrence of the bits in the sequence depends somehow upon the bits which have already been emitted by the source, and to capture this dependence we should use a source endowed with memory, such as a Markovian source.

3.2 Turing Machines

Let $\mathrm{TM}(n, m)$ be the class of Turing machines with n symbols and m states. Several definitions of Turing machines, all equivalent from the computational power point of view, have been given in the literature. Here we refer to the traditional (standard) one, with a bi-infinite tape and one read/write head. We refer to [25] for a formal definition of Turing machines, configurations and computations.

Several authors have investigated small universal Turing machines. For example, Rogozhin [22] constructed small universal machines in the classes $\mathrm{TM}(5, 5)$, $\mathrm{TM}(6, 4)$, $\mathrm{TM}(10, 3)$ and $\mathrm{TM}(18, 2)$, Kudlek and Rogozhin [11] constructed a machine in $\mathrm{TM}(9, 3)$, and Baiocchi [1] constructed machines in $\mathrm{TM}(2, 19)$ and $\mathrm{TM}(4, 7)$. The smallest machine, both in terms of number of instructions (22) and number of symbol–state pairs (24) is Rogozhin's machine in $\mathrm{TM}(6, 4)$. Let us note in passing that, by slightly modifying the definition of Turing machines (and, consequently, the notion of universality) it is possible to obtain even smaller machines [15,16,3,26]. Concerning traditional machines, assuming that a single instruction is reserved for halting, it is known that there are no universal machines in $\mathrm{TM}(2, 2)$, $\mathrm{TM}(2, 3)$, $\mathrm{TM}(3, 2)$, $\mathrm{TM}(n, 1)$ and $\mathrm{TM}(1, n)$ for $n \geq 1$. See [14] for further details and references.

Given $M \in \mathrm{TM}(n, m)$, we need $\lg n$ and $\lg m$ bits to represent each symbol and each state, respectively. The read/write head can only move to the cell on its left, to the cell on its right, or not move; two bits are thus needed to represent the head movement. A simple encoding of M is a sequence of blocks of $2 \lg n + 2 \lg m + 2$

bits, each encoding an instruction of the kind *(current state, current symbol, new state, new symbol, head movement)*. The maximum number of blocks is $n \cdot m$. By writing the 2 head movement bits at the beginning of each block, we can encode the empty instruction (for those symbol–state pairs that do not have an instruction) as the unused head movement 2-bit configuration. In this way, the description size (under this encoding $\mathcal{C}$) of any Turing machine $M \in \mathrm{TM}(n, m)$ having $k \leq n \cdot m$ instructions will be $ds_{\mathcal{C}}(M) = k \cdot (2 \lg n + 2 \lg m + 2) + (n \cdot m - k) \cdot 2$ bits.

Using this encoding, each non-empty instruction of Rogozhin's 6-symbol 4-state 22-instruction universal machine requires 12 bits. Hence the description size of the machine is 268 bits, which is an upper bound to the description size $ds(\mathrm{TM})$ of the class of Turing machines. To limit the length of the paper, we do not show the explicit bit string that encodes Rogozhin's machine; we just note that its size is just a little bit higher than the (uncompressed) size of the smallest currently known register machine.

3.3 Spiking Neural P Systems

Let $\mathrm{SNP}(m, R, C, D)$ denote the class of SN P systems having degree m and a total number R of rules, where each rule consumes a maximum number C of spikes and has a maximum delay D.

Let $\Pi \in \mathrm{SNP}(m, R, C, D)$. In order to describe the synapse graph of Π (which is a directed graph, without self–loops) we need $m^2 - m$ bits. To describe a forgetting rule $a^s \to \lambda$ we need 1 bit to distinguish it from spiking rules, and $\lg C$ bits to represent the value of s. On the other hand, to describe a firing rule $E/a^c \to a; d$ we need 1 bit to distinguish it from forgetting rules, $\lg C$ bits to represent c and $\lg D$ bits to represent d; moreover, we need some bits to describe the regular expression E. In general, there are no limitations to the length of a regular expression, but by observing the systems described in [17] we note that the expressions $a, a^2, a^3, a(aa)^+$ and $a(aa)^*$ suffice to reach computational completeness, and thus we will restrict our attention to systems that contain only these kinds of regular expressions. Of course this restriction will influence our results; any different choice of the set of regular expressions is valid, provided that the class of SN P systems thus obtained contains at least one universal computation device. To specify one of the above five expressions we need 3 bits, and hence we need a total of $1 + \lg C$ bits to describe a forgetting rule, and $1 + 3 + \lg C + \lg D$ bits to describe a firing rule. On average, a *randomly generated* SN P system with R rules will contain about $\frac{R}{2}$ firing rules and $\frac{R}{2}$ forgetting rules, and thus we will need $\frac{R}{2}[1 + \lg C] + \frac{R}{2}[4 + \lg C + \lg D] = R[1 + \lg C] + \frac{R}{2}[3 + \lg D]$ bits to encode it.

A simple encoding of Π is a sequence of m blocks — one for each neuron — followed by the $m^2 - m$ bits that encode the structure of the synapse graph. For each neuron we have to specify the list of its rules; since each neuron may have a different number of rules (possibly zero), we will put an additional bit equal to 1 in front of the encoding of each rule, and a 0 at the end of the list. In this way, when decoding, the presence of a 1 means that the next bits encode a rule of the neuron, whereas a 0 means that the next bits encode a different neuron. Using

this encoding $\mathcal{C}$, the description size of a *randomly chosen* (and thus, possibly, non–universal) system $\Pi \in \mathrm{SNP}(m, R, C, D)$ is:

$$ds_{\mathcal{C}}(\Pi) = \frac{R}{2}[2 + \lg C] + \frac{R}{2}[5 + \lg C + \lg D] + R + m^2 - m = \\ = \frac{9R}{2} + R \lg C + \frac{R}{2} \lg D + m^2 - m$$

As we did with register machines, in order to determine an upper bound to the description size $ds(SNP)$ of the entire class of SN P systems we now turn our attention to *universal* systems. In [17], a *small* universal SN P system is obtained by simulating a slightly modified version of the small universal deterministic register machine described in section 3.1. The modification is needed for a technical reason due to the behavior of SN P systems (see [17] for details); the modified version of the register machine has 9 registers, 24 instructions and 25 labels. Each instruction is simulated through an appropriate subsystem; moreover, an INPUT module is needed to read the input spike train from the environment and initialize the simulation, and an OUTPUT module is needed to produce the output spike train if and when the computation of the simulated register machine halts. As a result, the universal SN P system is composed of 91 neurons, which are subsequently reduced to 84 by simulating in a different way one occurrence of two consecutive $INCs$ and two occurrences of an INC followed by a DEC. These 84 neurons are used as follows: 9 neurons for the registers, 22 neurons for the labels, 18 auxiliary neurons for the 9 INC instructions, 18 auxiliary neurons for the 9 DEC instructions, 2 auxiliary neurons for the simulation of two consecutive INC instructions, $3 \cdot 2 = 6$ auxiliary neurons for the two simulations of consecutive $INC - DEC$ instructions, 7 neurons for the INPUT module, and finally 2 neurons for the OUTPUT module. Considering all these neurons, the system contains a total number $R = 117$ of rules.

For such a system it is uncomfortable to make a detailed analysis of the encoded string as we did for register machines, and thus we will just determine its length. Let us first note that in such a system we have $D = 1$ and $C = 3$, and thus 1 and 2 bits will suffice to represent any delay and any number of consumed spikes, respectively. The 9 neurons that correspond to the registers contain two firing rules each, and thus require 15 bits each, for a total of 135 bits. The 22 neurons associated with the labels contain each one firing and one forgetting rule, for a total of 11 bits that, multiplied by 22, makes 242 bits. Each auxiliary neuron involved in the simulation of the 9 INC instructions contains one firing rule, and thus requires 8 bits to be described; all the 18 neurons require 144 bits. The same argument applies to the 18 auxiliary neurons involved in the simulation of the 9 DEC instructions, thus adding further 144 bits. The two auxiliary neurons used to simulate two consecutive INC instructions also contain one firing rule each, thus contributing with 16 bits. The same applies to the 6 auxiliary neurons used to simulate (two instances of) an INC followed by a DEC, thus adding 48 bits, as well as to the 7 neurons that are used in the INPUT module (56 bits) and the 2 auxiliary neurons of the OUTPUT module (16 bits).

All considered, we need 801 bits to describe the rules contained in the neurons. To these we must add the $m^2 - m = 6972$ bits needed to describe the structure of the synapse graph. We thus obtain a total of 7773 bits to encode the universal standard SN P system presented in [17]. This quantity is an upper bound to $ds_{\mathcal{C}}(\Pi)$, the description size of the system under the proposed encoding, which in turn is an upper bound to $ds(\text{SNP})$, the description complexity of the class of SN P systems. Tighter bounds can be obtained by explicitly computing the encoding of Π and then compressing it by means of a lossless compressor.

By assuming $ds(\text{DRM}) = 263$ and $ds(\text{SNP}) = 7773$, an approximated value of the *redundancy* of spiking neural P systems with respect to deterministic register machines, is:

$$R_{\text{DRM}}(\text{SNP}) = \frac{ds(\text{SNP})}{ds(\text{DRM})} = \frac{7773}{263} \approx 29.56$$

On the other hand, by assuming $ds(\text{DRM}) = 258$ (that results from the computation of the entropy of the string that encodes the universal deterministic register machine depicted in Figure 1) the redundancy becomes $R_{\text{DRM}}(\text{SNP}) = \frac{ds(\text{SNP})}{ds(\text{DRM})} = \frac{7773}{258} \approx 30.13$. These results suggest that the description of a universal SN P system is at least 29 or 30 times more verbose with respect to the description of a universal deterministic register machine.

In [17] it is also shown that by allowing firing rules of the extended type it is possible to build a universal SN P system by using only 49 neurons. However this time many neurons have 7 rules instead of 2, and to describe every extended rule $E/a^c \to a^p; d$ we also need some bits to specify the value of p, that does not occur in standard rules. As a result, there may be some doubts about what, among the two systems, is smaller. To find out the winner of this competition, let us compute the description size of the extended SN P system. As reported in [17], this time the system is able to simulate the universal register machine which is composed of $n = 8$ registers and $m = 23$ instructions. The rules contain 12 different regular expressions, do not contain delays, and the maximum number of spikes produced or consumed is 13. Thus we will need 4 bits to specify a regular expression, 0 bits to represent the delays, and 4 bits to represent each number of produced/consumed spikes. The 49 neurons are used as follows: 8 neurons for the registers, 22 neurons for the labels, 13 for the DEC instructions, 5 for the INPUT module, and 1 for the OUTPUT module. Each extended firing rule requires $1+4+4+4 = 13$ bits to be encoded, whereas a forgetting rule requires $1+4 = 5$ bits. Recall that each rule is preceded by a 1 in a list of rules, while the list itself is terminated with a 0. Each of the 8 neurons used for the registers contains 2 firing rules ($2 \cdot 13 + 3 = 29$ bits), for a total of 232 bits. Each of the 22 neurons used for the labels contains 3 firing rules and 4 forgetting rules (67 bits), for a total of 1474 bits. Each of the 13 neurons which are used in the simulation of the DEC instructions contains 1 firing rule (15 bits), for a total of 195 bits. Each of the 5 neurons used in the INPUT module also contains 1 firing rule (total: 75 bits), whereas the neuron used in the OUTPUT module contains 2 firing rules and 1 forgetting rule (35 bits). To all these bits we must add the $49^2 - 49 = 2352$ bits which are needed to encode the synapse graph. All considered, we obtain

4361 bits, which is well less than the 7773 bits obtained with the first universal SN P system. Hence this is a tighter upper bound to $ds(\text{SNP})$ and, assuming $ds(\text{DRM}) = 263$ or $ds(\text{DRM}) = 258$, we obtain

$$R_{\text{DRM}}(\text{SNP}) = \frac{4361}{263} \approx 16.58 \qquad \text{and} \qquad R_{\text{DRM}}(\text{SNP}) = \frac{4361}{258} \approx 16.90$$

respectively. Also in this case, tighter bounds can be obtained by explicitly computing the bit string that encodes the universal extended SN P system and then compressing it using a lossless compressor.

3.4 UREM P Systems

Let UREM denote the class of UREM P systems. As proved in [5], in order to reach computational completeness we can restrict our attention to *deterministic* systems in which the skin membrane contains one elementary membrane for each register of the (possibly universal) simulated deterministic register machine. This means getting rid of the membrane structure, saving a lot of bits when describing the system. Similarly, we can restrict our attention to UREM P systems in which the amounts Δe of energy that occur in each rule are taken from the set $\{-1, 0, 1\}$. This means that for each rule 2 bits will suffice to encode the actual value of Δe.

Under these assumptions, we can define the subclass $\text{UREM}(n, m, R)$ of UREM P systems having R rules, an alphabet of m symbols, and n elementary membranes contained into the skin. Given $\Pi \in \text{UREM}(n, m, R)$, for each membrane we have to specify the list of its rules. Just like it happens with SN P systems, in general every membrane will have a different number of rules, and thus we will append the description of each rule by a bit equal to 1, and we will conclude each list of rules with a 0. To encode each rule $(op_i : a, \Delta e, b)$ we need 1 bit to specify whether $op = in$ or $op = out$, $2 \cdot \lg m$ bits to specify the alphabet symbols a and b, and 2 bits to express the value of Δe. A simple encoding of Π is composed of $n+1$ blocks, one for each membrane. Each block encodes the sequence of rules associated with the corresponding membrane, listing the rules as described above. Each rule requires $4 + 2 \lg m$ bits to be encoded (one bit is used to indicate that we have not yet reached the end of the list of rules), for a total of $2R\,[2 + \lg m]$ bits. One bit is needed to terminate each of the $n + 1$ lists, and thus the description size of the whole system under the encoding $\mathcal{C}$ just proposed is $ds_{\mathcal{C}}(\Pi) = 2R\,[2+\lg m]+n+1$.

A small universal UREM P system (here proposed for the first time) can be obtained by simulating the small universal deterministic register machine described in section 3.1. Such a small UREM P system contains $n = 8$ elementary membranes. As stated in Section 2.3, to simulate the instructions of the register machine we need the objects p_j and $\widetilde{p}_j$, for all $j \in \{0, 1, \ldots, 21\}$ (see [5] for details), as well as the object p_{22} to simulate the jump to the non-existent instruction number 22 (to halt the computation), for a total of $m = 45$ alphabet symbols. Each INC instruction of the register machine requires 2 rules to be simulated, whereas each DEC instruction requires 3 rules, for a total of $R = 57$ rules, reported in Figure 2.

$0 : (in_1 : p_0, 0, \widetilde{p}_0),\ (out_1 : \widetilde{p}_0, -1, p_1),\ (out_1 : \widetilde{p}_0, 0, p_2)$
$1 : (in_7 : p_1, 1, \widetilde{p}_1),\ (out_7 : \widetilde{p}_1, 0, p_0)$
$2 : (in_6 : p_2, 1, \widetilde{p}_2),\ (out_6 : \widetilde{p}_2, 0, p_3)$
$3 : (in_5 : p_3, 0, \widetilde{p}_3),\ (out_5 : \widetilde{p}_3, -1, p_2),\ (out_5 : \widetilde{p}_3, 0, p_4)$
$4 : (in_6 : p_4, 0, \widetilde{p}_4),\ (out_6 : \widetilde{p}_4, -1, p_5),\ (out_6 : \widetilde{p}_4, 0, p_3)$
$5 : (in_5 : p_5, 1, \widetilde{p}_5),\ (out_5 : \widetilde{p}_5, 0, p_6)$
$6 : (in_7 : p_6, 0, \widetilde{p}_6),\ (out_7 : \widetilde{p}_6, -1, p_7),\ (out_7 : \widetilde{p}_6, 0, p_8)$
$7 : (in_1 : p_7, 1, \widetilde{p}_7),\ (out_1 : \widetilde{p}_7, 0, p_4)$
$8 : (in_6 : p_8, 0, \widetilde{p}_8),\ (out_6 : \widetilde{p}_8, -1, p_9),\ (out_6 : \widetilde{p}_8, 0, p_0)$
$9 : (in_6 : p_9, 1, \widetilde{p}_9),\ (out_6 : \widetilde{p}_9, 0, p_{10})$
$10 : (in_4 : p_{10}, 0, \widetilde{p}_{10}),\ (out_4 : \widetilde{p}_{10}, -1, p_0),\ (out_4 : \widetilde{p}_{10}, 0, p_{10})$
$11 : (in_5 : p_{11}, 0, \widetilde{p}_{11}),\ (out_5 : \widetilde{p}_{11}, -1, p_{12}),\ (out_5 : \widetilde{p}_{11}), 0, p_{13})$
$12 : (in_5 : p_{12}, 0, \widetilde{p}_{12}),\ (out_5 : \widetilde{p}_{12}, -1, p_{14}),\ (out_5 : \widetilde{p}_{12}), 0, p_{15})$
$13 : (in_2 : p_{13}, 0, \widetilde{p}_{13}),\ (out_2 : \widetilde{p}_{13}, -1, p_{18}),\ (out_2 : \widetilde{p}_{13}), 0, p_{19})$
$14 : (in_5 : p_{14}, 0, \widetilde{p}_{14}),\ (out_5 : \widetilde{p}_{14}, -1, p_{16}),\ (out_5 : \widetilde{p}_{14}), 0, p_{17})$
$15 : (in_3 : p_{13}, 0, \widetilde{p}_{13}),\ (out_3 : \widetilde{p}_{13}, -1, p_{18}),\ (out_3 : \widetilde{p}_{13}), 0, p_{20})$
$16 : (in_4 : p_{16}, 1, \widetilde{p}_{16}),\ (out_4 : \widetilde{p}_{16}, 0, p_{11})$
$17 : (in_2 : p_{17}, 1, \widetilde{p}_{17}),\ (out_2 : \widetilde{p}_{17}, 0, p_{21})$
$18 : (in_4 : p_{18}, 0, \widetilde{p}_{18}),\ (out_4 : \widetilde{p}_{18}, -1, p_0),\ (out_4 : \widetilde{p}_{18}), 0, p_{22})$
$19 : (in_0 : p_{19}, 0, \widetilde{p}_{19}),\ (out_0 : \widetilde{p}_{19}, -1, p_0),\ (out_3 : \widetilde{p}_{19}), 0, p_{18})$
$20 : (in_0 : p_{20}, 1, \widetilde{p}_{20}),\ (out_0 : \widetilde{p}_{20}, 0, p_0)$
$21 : (in_3 : p_{21}, 1, \widetilde{p}_{21}),\ (out_3 : \widetilde{p}_{21}, 0, p_{18})$

Fig. 2. A small universal deterministic UREM P system. In each row, the number on the left refers to the label of the simulated instruction of the register machine depicted in Figure 1.

The skin membrane does not contain any rule, and thus the first block of the encoding of Π is 0. The elementary membrane that simulates register 0 contains the five rules that correspond to the INC (line 19 in Figure 2) and to the DEC (line 20) instructions that affect the contents of register 0. Since $n = 8$, we will need 3 bits to encode a register number; similarly, to encode an alphabet symbol we will need $\lg m = \lg 45 = 6$ bits. The bit string that encodes membrane 0 is thus:

$$1\underbrace{0}_{in}\underbrace{010011}_{p_{19}}\underbrace{00}_{0}\underbrace{110011}_{\widetilde{p}_{19}}1\underbrace{1}_{out}\underbrace{110011}_{\widetilde{p}_{19}}\underbrace{11}_{-1}\underbrace{000000}_{p_0}$$
$$1\underbrace{1}_{out}\underbrace{110011}_{p_{19}}\underbrace{00}_{0}\underbrace{010010}_{p_{18}}1\underbrace{0}_{in}\underbrace{010100}_{p_{20}}\underbrace{01}_{1}\underbrace{110100}_{\widetilde{p}_{20}}$$
$$1\underbrace{1}_{out}\underbrace{110100}_{\widetilde{p}_{20}}\underbrace{00}_{0}\underbrace{000000}_{p_0}0$$

where we have encoded in as 0, out as 1, $\Delta e = 0$ as 00, $\Delta e = 1$ as 01, $\Delta e = -1$ as 11, p_k as the 5-bit binary encoding of $k \in \{0, 1, \ldots, 22\}$ with an additional leading

0, and $\widetilde{p}_k$ as the 5-bit binary encoding of $k \in \{0, 1, \ldots, 21\}$ with an additional leading 1. Operating in a similar way for all the elementary membranes of Π, we obtain the following binary string (the spaces denote a separation between consecutive rules; they are put here to help the reader, but are not necessary to decode the string):

(Skin membrane)

0

(Membrane 0)

1 001001100110011 1 111001111000000 1 111001100010010
1 001010001110100 1 111010000000000 0

(Membrane 1)

1 000000000100000 1 110000011000001 1 110000000000010
1 000011101100111 1 110011100000100 0

(Membrane 2)

1 000110100101101 1 110110111010010 1 110110100010011
1 001000101110001 1 111000100010101 0

(Membrane 3)

1 000110100101101 1 110110111010010 1 110110100010100
1 001010101110101 1 111010100010010 0

(Membrane 4)

1 000101000101010 1 110101011000000 1 110101000001011
1 101000001110000 1 111000000001011 1 001001000110010
1 111001011000000 1 111001000010110 0

(Membrane 5)

1 000001100100011 1 110001111000010 1 110001100000100
1 000010101100101 1 110010100000110 1 000101100101011
1 110101111001100 1 110101100001101 1 000110000101100
1 110110011001110 1 110110000001111 1 000111000101110
1 110111011010000 1 110111000010001 0

(Membrane 6)

1 000001001100010 1 110001000000011 1 000010000100100
1 110010011000101 1 110010000000011 1 000100000101000
1 110100011001001 1 110100000000000 1 000100101101001
1 110100100001010 0

(Membrane 7)

1 000000101100001 1 110000100000000 1 000011000100110
1 110011011000111 1 110011000001000 0

We can thus conclude that $ds_{\mathcal{C}}(\Pi) = 921$, where $\mathcal{C}$ denotes our encoding.

Since UREM P systems are a relatively compact model of P systems, it is interesting to ask how many bits we would further save by compressing the above bit string. By operating like we did with register machines, if we look at such a string we can see that it contains 516 zeros and 405 ones. Hence the probability that a 0 occurs in any given position is $p_0 = \frac{516}{921}$, whereas the probability that a 1 occurs is $p_1 = \frac{405}{921}$. The entropy of the above sequence is thus $H(\Pi) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \approx 0.990$, and we can conclude that a compressed string produced by an optimal entropy-based compressor, whose length is approximately equal to the length of the uncompressed string times the entropy, would contain $\lceil 911.79 \rceil = 912$ bits. Such a quantity is an upper bound to ds(UREM), the theoretical description size complexity of the class of UREM P systems.

By assuming ds(DRM) = 263 and ds(UREM) = 921, an approximated value of the *redundancy* of UREM P systems with respect to deterministic register machines is:

$$R_{\text{DRM}}(\text{UREM}) = \frac{ds(\text{UREM})}{ds(\text{DRM})} = \frac{921}{263} \approx 3.50$$

On the other hand, by assuming ds(DRM) = 258 and ds(UREM) = 912 (that result by considering the entropies of the corresponding encoded strings) the redundancy becomes $R_{\text{DRM}}(\text{UREM}) = \frac{912}{258} \approx 3.53$. These results suggest that the description of a universal UREM P system is at least 3.5 times more verbose with respect to the description of a universal deterministic register machine.

4 Conclusions and Directions for Further Research

Trying to find a common measure for the size of different computation devices, we have introduced the *description size* of a device M as the length of the binary string produced by a "reasonable" encoding of M. For four classes of computation devices (deterministic register machines, Turing machines, SN P systems and UREM P systems) we have computed the description size of randomly chosen devices, as well as of a universal device taken from each class. In this way we have observed that the smallest universal SN P system currently known has a description which is about 16.58 times as verbose as the the description of the smallest deterministic register machine (currently known), while the smallest universal deterministic UREM P system (here described for the first time) is only about 3.5 times more verbose with respect to the register machine. Further, we have seen that the smallest (standard) universal Turing machine currently known has about the same size of the smallest deterministic register machine. An intriguing question that naturally arises after these calculations is: What is the minimum theoretical description size that can be obtained by considering *all* possible universal computing devices (including, for example, the small universal tissue and antiport P systems considered in [4,23,6])? In other words: What is the minimum number of bits which are necessary to describe the structure of a universal computing device?

Different encoding functions could produce shorter strings than those we have presented in this paper, thus showing tighter bounds to the theoretical values of description size complexities. Improving our results under this point of view is a direction of research of a clear interest. Let us also note that we did not formally specify the encoding and decoding functions for our computational models. Finally, we did not count the size of these functions when calculating the description size of our computation devices. Whether this choice is correct or not is questionable, but let us note that also representing the decoding algorithm as a string of bits would require a decoding process, and so on in an endless sequence of encodings and decodings.

Acknowledgments

The work of the authors was supported by MiUR under the project "Azioni Integrate Italia-Spagna - Theory and Practice of Membrane Computing" (Acción Integrada Hispano-Italiana HI 2005-0194), and by project "Models of natural computing and applications" — FIAR 2007 — University of Milano–Bicocca.

References

1. Baiocchi, C.: Three small universal turing machines. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 1–10. Springer, Heidelberg (2001)
2. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity, vol. I and II. Springer, Heidelberg (1988–1990)
3. Cook, M.: Universality in elementary cellular automata. Complex Systems 15, 1–40 (2004)
4. Csuhaj–Varjú, E., Margenstern, M., Vaszil, G., Verlan, S.: On small universal antiport P systems. Theoretical Computer Sci. 372, 152–164 (2007)
5. Freund, R., Leporati, A., Oswald, M., Zandron, C.: Sequential P systems with unit rules and energy assigned to membranes. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 200–210. Springer, Heidelberg (2005)
6. Freund, R., Oswald, M.: Small universal antiport P systems and universal multiset grammars. In: Proc. 4$^{\text{th}}$ Brainstorming Week on Membrane Computing, RGNC Report 03/2006, Fénix Editora, Sevilla, vol. II, pp. 51–64 (2006)
7. Ionescu, M., Păun, A., Păun, Gh., Jesús Pérez-Jímenez, M.: Computing with spiking neural P systems: Traces and small universal systems. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 1–16. Springer, Heidelberg (2006)
8. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71, 279–308 (2006)
9. Korec, I.: Small universal register machines. Theoretical Computer Sci. 168, 267–301 (1996)
10. Kudlek, M.: Small deterministic Turing machines. Theoretical Computer Sci. 168, 241–255 (1996)
11. Kudlek, M., Rogozhin, Y.: A universal turing machine with 3 states and 9 symbols. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 311–318. Springer, Heidelberg (2002)

12. Minsky, M.L.: Size and structure of universal Turing machines using Tag systems. In: Recursive Function Theory, Symp. in Pure Mathematics, pp. 229–238. American Mathematical Society, Providence (1962)
13. Neary, T., Woods, D.: Small fast universal Turing machines. Theoretical Computer Sci. 362, 171–195 (2006)
14. Neary, T., Woods, D.: Four small universal turing machines. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 242–254. Springer, Heidelberg (2007)
15. Woods, D., Neary, T.: Small semi-weakly universal turing machines. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 303–315. Springer, Heidelberg (2007)
16. Neary, T., Woods, D.: Small weakly universal Turing machines, arXiv [cs.CC]: 0707.4489v1 (2007)
17. Păun, A., Păun, Gh.: Small universal spiking neural P systems. BioSystems 90, 48–60 (2007)
18. Păun, Gh.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
19. Păun, Gh.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
20. Rogozhin, Y.: Seven universal Turing machines. Math. Issled 69, 76–90 (1982)
21. Rogozhin, Y.: A universal Turing machine with 10 states and 3 symbols. Izv. Akad. Nauk. Respub. Moldova Mat. 4, 80–82, 95 (1992)
22. Rogozhin, Y.: Small universal Turing machines. Theoretical Computer Sci. 168, 215–240 (1996)
23. Rogozhin, Y., Verlan, S.: On the rule complexity of universal tissue P systems. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 356–362. Springer, Heidelberg (2006)
24. Shannon, C.E.: A universal Turing machine with two internal states. Automata Studies, Ann. Math. Stud. 34, 157–165 (1956)
25. Sipser, M.: Introduction to the Theory of Computation. PWS Publishing Co. (1997)
26. Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign (2002)
27. The P systems Web page, `http://ppage.psystems.eu/`