International Journal of Computer Mathematics
Vol. 00, No. 00, June 2012, 1–12

RESEARCH ARTICLE

Universality of Enzymatic Numerical P Systems

Cristian Ioan Vasile[a]* and Ana Brânduşa Pavel[a] and Ioan Dumitrache[a]

[a] *Department of Automatic Control and Systems Engineering*
Politehnica University of Bucharest
Splaiul Independenţei 313, 060042 Bucharest, Romania
email: {cristian.vasile, ana.pavel, ioan.dumitrache}@acse.pub.ro

(*October 2012*)

This paper provides the proof that Enzymatic Numerical P systems with deterministic, but parallel, execution model are universal, even when the production functions used are polynomials of degree 1. This extends previous known results and provides the optimal case in terms of polynomial degree.

Keywords: Membrane computing, Numerical P systems, Enzymatic Numerical P systems, Turing universality, Bio-inspired computing

1. Introduction

P systems represent a computational paradigm inspired by the cell architecture and functioning. Several classes of P systems have been introduced in the framework of membrane computing [11]. Numerical P systems (NP systems) are a type of P systems, inspired by the cell structure, in which numerical variables evolve inside the compartments by means of programs; a program (or rule) is composed of a production function and a repartition protocol. The variables have a given initial value and the production function is a multivariate polynomial. The value of the production function for the current values of the variables is distributed among variables in certain compartments according to a repartition protocol. The formal definition of NP systems can be found in [10] where the authors introduce this type of P systems and show some possible applications in economics.

NP systems were designed both as deterministic and non-deterministic systems [10]. Non-deterministic NP systems allow the existence of more rules per each membrane and the rule to apply in any step is selected by an "oracle",while in deterministic NP systems the state at time $t+1$ is completely determined by the state at time t (one way to achieve this is by having at most one rule per membrane). NP systems were used as a naturally parallel and distributed modeling tool for the design of robot controllers [2], [6], [7]. Designing robot controllers requires deterministic mechanisms. Therefore, an extension of NP systems, Enzymatic Numerical P systems (ENP systems), in which enzyme-like variables allow the existence of more than one program (rule) in each membrane, while keeping the deterministic

*Corresponding author. Email: cristian.ioan.vasile@gmail.com

ISSN: 0020-7160 print/ISSN 1029-0265 online

DOI: 0020716YYxxxxxxxx
http://www.informaworld.com

nature of the system, were introduced in [5]. Due to their properties, ENP systems represent a more powerful modeling tool for robot behaviors than classical NP systems [6], [7].

In this paper we extend the results obtained in [13] regarding the power of ENP systems. In [13] it is shown that ENP systems are universal for both non-deterministic and deterministic case (for production polynomials of degree two). Such universality results are also interesting from the robot control point of view, as they guarantee that any computer program or robot behavior can be implemented using ENP systems. Therefore, in this paper we improve the universality result obtained in [13] for deterministic ENP systems, working in parallel execution mode, by reducing the degree of the polynomials of the production functions from 2 to 1 and the number of membranes from 253 to 6.

The paper is structured as follows. In the next section the authors provide the formal definition of ENP systems. In section 3, the previous results obtained regarding the power of ENP systems are presented and explained. Section 4 provides the proof of the new result regarding universality of ENP systems. Finally, section 5 is dedicated to the conclusions of the paper.

2. Definitions

An ENP system is defined as an NP system with special enzyme-like variables which control the execution of the rules. The reader is assumed familiar with basic ideas of membrane computing, e.g., [8, 9, 11].

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0)), v_{j_0,i_0}) \quad (1)$$

where:

- m is the degree of the membrane system (the number of membranes), $m \geq 1$;
- H is an alphabet of membrane labels;
- μ represents the tree structure of the membrane system;
- v_{j_0,i_0} is a distinguished variable from a compartment i_0 which provides the results of the computation;
- Each membrane is defined by a 3-tuple:
  (1) Var_i is a (finite) set of variables from compartment i;
  (2) $Var_i(0)$ are the initial values of the variables from compartment i;
  (3) Pr_i is the set of programs from compartment i. Programs have one of the two following forms:
    a) non-enzymatic form, which is exactly like the one from the standard NP systems:

$$Pr_{j,i} = F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}) \rightarrow c_{j,1}|v_1 + \ldots + c_{j,n_i}|v_{n_i} \quad (2)$$

    b) enzymatic form

$$Pr_{j,i} = F_{j,i}(x_{1,i}, \ldots, x_{k_i,i})|_{e_j} \rightarrow c_{j,1}|v_1 + \ldots + c_{j,n_i}|v_{n_i} \quad (3)$$

    where $e_j \in Var_i$ is an enzyme-like variable which controls the activation of the rule.

A program is composed of a production functions, a repartition protocol and optionally an enzyme-like variable. Each program is evaluated in three steps, activation-production-distribution. First of all, it is established which rules are active. There can be more than one active rule in a membrane or none. A rule is active if it is in the non-enzymatic form or if the associated enzyme has a greater value than one of the variables involved in the production function. All active rules in the membrane system are executed in parallel in one computational step. This parallel execution mechanism is denoted by *allP*.

The actual execution of the programs is done in the following to steps. At any time t, every production function $F_{j,i}(x_{1,i}(t), ..., x_{k_i,i}(t))$ from every active program is computed. The current values of the variables are used, even if they appear in more than one production function. The variables $x_{1,i}, ..., x_{k_i,i}$ belong to compartment i. A production function is not required to use all variables from the compartment the corresponding program belongs to. However, the variables which are used by a production function of an active program will be reset to zero before the distribution step.

In the final step, the value $F_{j,i}(x_{1,i}(t), ..., x_{k_i,i}(t))$ is distributed to variables $v_1, \ldots, v_{n_i}$, according to coefficients $c_{j,1}, \ldots, c_{j,n_i}$ in order to obtain the values of these variables at time $t+1$. The set of variables $\{v_1, \ldots, v_{n_i}\}$ is the union of the set of variables from the membrane the program belongs to, the parent membrane and the children membranes. Formally, $\{v_1, \ldots, v_{n_i}\} = Var_i \cup Var_{par(i)} \cup \left(\bigcup_{ch \in Ch(i)} Var_{ch}\right)$, where $par(i)$ is the parent of membrane i and $Ch(i)$ is the set of children of membrane i from μ. The coefficients $c_{j,1}, \ldots, c_{j,n_i}$ are natural numbers (they may be also 0, in which case the terms "$+0|x$" are omitted) [10] which specify the proportion of the current production distributed to each variable $v_1, ..., v_{n_i}$. Denote with $C_{j,i} = \sum_{s=1}^{n_i} c_{j,s}$ the sum of all coefficients of the repartition protocol. The value

$$q_{j,i}(t) = \frac{F_{j,i}(x_{1,i}(t), ..., x_{k_i,i}(t))}{C_{j,i}}$$

represents the "unitary portion". The value $q_{j,i}(t) * c_{j,s}, 1 \leq s \leq n_i$, will be added to the value of the variable v_s which belongs to the repartition protocol of program j.

If after applying all the rules, a variable receives "contributions" from several programs, then they are all added in order to produce the next value of the variable.

The execution of the model is synchronized, meaning that each of the three described steps performed at time t is done for all programs at the same time.

3. The power of Enzymatic Numerical P systems

In [13] the authors prove and analyze the universality of ENP systems. The main results in [10] and [13] regarding the power of NP systems and ENP systems are the following (the notations are immediately explained):

THEOREM 3.1 $NRE = N^+P_8(poly^5(5), seq) = N^+P_7(poly^5(6), seq) = NP_7(poly^5(5), enz, seq) = NP_*(poly^1(2), enz, oneP) = NP_{254}(poly^2(253), enz, allP, det)$.

NRE represents the family of computable (recursively enumerable) sets of natural numbers. Denote by $N(\Pi)$ the set of values the distinguished variable v_{j_0,i_0}

of membrane system Π takes during a computation, a (finite or infinite) sequence of transitions. When the superscript $+$ is used, $N^+(\Pi)$, it indicates that only the positive values of v_{j_0,i_0} are considered. The family of sets of numbers $N(\Pi)$ computed by NP systems with at most m membranes, production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial, is denoted by $NP_m(poly^n(r)),\ m \geq 1, n \geq 0, r \geq 0$ [13]. If one of the parameters m, n, r is not bounded, then it is replaced by $*$. The superscript $+$ used in $N^+P_m(poly^n(r))$ denotes that only the positive values are considered as results of the computations. *seq* indicates the fact that the system works in sequential mode, in each step only one program is chosen non-deterministically to be executed in every membrane. *oneP* indicates the maximally parallel mode. In this execution mechanism, in each membrane a maximal set of programs (in the sense of set inclusion) is chosen non-deterministically to be executed such that no two programs share variables in their production functions. Finally, *allP* indicates the deterministic parallel execution model described above, where all active programs are executed, regardless of whether they share variables in their production functions. To highlight that the execution model is deterministic, the indicator *det* is used. The absence of *det* implies that the systems may also be non-deterministic. *enz* is used to indicate that the enzymatic mechanism is used. Again, the absence of *enz* implies that the enzymatic mechanism is not used.

The enzymatic mechanism and the advantages of ENP systems are detailed in [6], [1].

4. An improved universality result

The following theorem improves the parameters in the equality $NRE = NP_{254}(poly^2(253), enz, allP, det)$, thus proving again the power of the enzymatic mechanism. The obtained result is optimal in what concerns the degree of the used polynomials - they are of degree one, but at the same time we improve the result from [13] in what concerns the number of membranes and the number of variables in each production polynomial.

THEOREM 4.1 $NRE = NP_4(poly^1(6), enz, allP, det)$.

Proof The proof is constructive and is based on the characterization of recursive-enumerable set (RE) using polynomials [3, 12]. It is proven that for every RE set S there is a multivariate polynomial Q with integer coefficients such that the set of positive values of Q, corresponding to tuples of natural numbers, is S.

$$\forall S \subseteq \mathbb{N}^*(\ S \text{ is a RE set}), \exists Q \in \mathbb{Z}_n[X]\ s.t.\ S = \{\, Q(x) |\, x \in \mathbb{N}^n, Q(x) > 0\},$$

where $\mathbb{N}^*$ is the set of natural numbers without 0, $\mathbb{Z}_n[X]$ is the set of polynomials of n variables with integer coefficients. It is also shown that polynomials of degree at most 5 and with 5 variables are sufficient to generate the elements of RE sets.

A computing device X is Turing universal if for every RE set S there is an instance of X which generates the elements of S. In our case, using the characterization by polynomials, it is suffice to show that for every polynomial Q of degree at most 5 and with 5 variables there is an ENP system which can enumerate all 5-tuples of natural numbers and compute the corresponding values of Q.

This technique was first employed to show that "standard" NP systems are universal in [10] and also in [13] to show the universality of ENP systems.

The ENP systems considered in the theorem are deterministic. As such, in order to construct a membrane system corresponding to some polynomial Q, we propose two simple deterministic methods of enumerating all 5-tuples of natural numbers (the method can be used to enumerate tuples of natural numbers of any length) and of computing a polynomial of degree at most 5 with 5 variables.

The generating method is the same as in the proof from [13]. The variables of Q forming a 5-tuple are regarded as a single number with 5 digits in a certain base. The algorithm continuously counts down from the highest 5-digit number to zero in ever increasing base. Therefore, if the current base is $b+1$, the membrane will generate the numbers from *bbbbb* to 00000. When the null tuple is reached the variables are reset to the highest 5-digit number of the next base, $b+2$. The algorithm starts with base 6 from the tuple $(5,5,5,5,5)$ and generates $(5,5,5,5,4)$, $\ldots$, $(5,5,5,5,0)$, $(5,5,5,4,5)$, $\ldots$, $(0,0,0,0,0)$. At this point it will move to the tuple $(6,6,6,6,6)$ which corresponds to the highest 5-digit number in base 7. The process repeats indefinitely, thus generating all 5-tuples of natural numbers in a deterministic way.

The computing method is based on the fact that every polynomial Q of degree 5 with 5 variables can be put in the following form (lemma from [13]):

$$Q(x_1,\ldots,x_5)=\sum_{i=1}^{m}\beta_i\cdot(a_{1,i}x_1+\cdots+a_{5,i}x_5+a_{6,i})^5 \tag{4}$$

where $m=252$ and represents the maximum number of terms of Q in the general form ($Q=\sum_{p_1+\ldots+p_5\leq 5}\prod_{i=1}^{5}x_i^{p_i}$) , β_i are polynomial-specific coefficients and $a_{j,i}$ are some integer constants.

The form (4) is well suited to compute the values of Q with only polynomials of degree 1 in the following way. First the values $v_i=a_{1,i}x_1+\cdots+a_{5,i}x_5+a_{6,i}$ are computed. Then v_i are raised to the fifth power. Computing the power of a number can be done using only multiplication; computing the 5-th power of v_i can be done by first computing $a=v_i\cdot v_i=v_i^2$, then $b=a\cdot a=v_i^4$ and finally $c=v_i\cdot b=v_i^5$. Since v_i are natural numbers, multiplication can be performed as repeated addition, $a\cdot b=\underbrace{a+\ldots+a}_{b}$. The final step is to sum all terms, $\sum_{i=1}^{m}\beta_i q_i$ with $q_i=v_i^5$.

The two methods were used to construct the following ENP system, also represented graphically in figure 1.

$$\begin{aligned}\Pi=(&4,H,\mu,(Var_{Generate},Pr_{Generate},Var_{Generate}(0))\\&(Var_{Compute},Pr_{Compute},Var_{Compute}(0),enum),\\&(Var_{Pow5},Pr_{Pow5},Var_{Pow5}(0)),\\&(Var_{Mult},Pr_{Mult},Var_{Mult}(0))),\\H=\{&Generate,Compute,Pow5,Mult\},\end{aligned}$$

$$
\begin{aligned}
\mu &= [_{Generate}[_{Compute}[_{Pow5}\ [_{Mult}]_{Mult}\ ]_{Pow5}]_{Compute}]_{Generate},\\
Var_{Generate} &= \{x_i, e_j, ez_k, er_i, n, e_t, g, gc :\ 1 \le i \le 5, 1 \le i \le 7, 2 \le k \le 5\},\\
Pr_{Generate} &= \{n \to 1|n,\ e_t \to 1|gc,\\
&\quad 1 + x_1|_{e_1} \to 1|er_1,\ -1 + g|_{e_1} \to 1|x_1,\ 1 + n + x_1|_{e_1} \to 1|x_1\}\\
&\cup \{j \cdot e_j \to 1|e_{j+1} + (j-1)|e_t :\ 1 \le j \le 5\}\\
&\cup \{1 + x_i|_{e_1} \to 1|ez_i,\ 1 - i + e_t|_{er_{i-1}} \to 1|x_i :\ 2 \le i \le 5\}\\
&\cup \{g + (ez_i + er_{i-1})|_{e_i} \to 1|er_i,\ 2 - i + n + e_t|_{er_i} \to 1|x_i\\
&\quad :\ 2 \le i \le 5\}\\
&\cup \{2 + e_t|_{er_5} \to \sum_{i=1}^{5} 1|x_i + 1|n,\ g + er_5|_{e_6} \to 1|gc,\ e_6 \to 1|e_7,\\
&\quad e_7 \to 1|e_1^c\}\\
&\cup \{g + 2 \cdot x_i|_{e_7} \to 1|x_i + 1|x_i^c\ :\ 1 \le i \le 5\},\\
Var_{Generate}(0) &= (5, 5, 5, 5, 5, 1, 0, \ldots, 0, 5, 0, 0, 0),\\
Var_{Compute} &= \{x_i^c, e_j^c, t, g^*, ep_t, f_Q, enum, aux, e_f :\ 1 \le i \le 5, 1 \le j \le 506\},\\
Pr_{Compute} &= \{g^* + \sum_{i=1}^{5} a_{i,k} \cdot x_i^c + a_{6,k}|_{e_{2k-1}^c} \to 1|s_1,\\
&\quad 3 \cdot e_{2k-1}^c \to 1|e_{2k}^c + 2|ep_1,\ e_{2k}^c|_{ep_t} \to 1|e_{2k+1}^c,\\
&\quad g^* - 2 \cdot \beta_k t|_{e_{2k+1}^c} \to 1|aux + 1|e_f\ :\ 1 \le k \le 252\}\\
&\cup \{2 \cdot e_{505}^c \to 1|e_f + 1|e_{506}^c,\ aux|_{e_f} \to 1|f_Q,\ -f_Q|_{g*} \to 1|enum,\\
&\quad -(f_Q + e_{506}^c) \to 1|e_f,\ enum + f_Q + ep_t \to 1|gc,\ e_{506}^c \to 1|e_1\}\\
Var_{Compute}(0) &= (0, 0, \ldots, 0),\\
Var_{Pow5} &= \{s_1, s_2, ep_i, e_M, gc^*, z\ :\ 1 \le i \le 7\},\\
Pr_{Pow5} &= \{z + 3 \cdot s_1|_{ep_1} \to 1|a + 1|b + 1|s_1,\ z + 2 \cdot s_2|_{ep_3} \to 1|a + 1|b,\\
&\quad z + s_1|_{ep_5} \to 1|a,\ z + s_2|_{ep_5} \to 1|b,\ z + s_2|_{ep_7} \to 1|t,\\
&\quad ep_7 \to 1|ep_t,\ e_M \to 1|gc^*\}\\
&\cup \{3 \cdot ep_{2k-1} \to 1|ep_{2k} + 2|e_s,\ ep_{2k}|_{e_M} \to 1|ep_{2k+1},\ 1 \le k \le 3\}\\
Var_{Pow5}(0) &= (0, 0, \ldots, 0),\\
Var_{Mult} &= \{a, b, z^*, d, u, e_s\},\\
Pr_{Mult} &= \{z^* + 1.5 \cdot a|_b \to 2|a + 1|s_2,\ z^* - (1 + d)|_b \to 1|d,\ d \to 1|b\\
&\quad e_s + b|_u \to 1|e_M,\ a + b|_u \to 1|gc^*\}\\
Var_{Mult}(0) &= (0, 0, 0, 0, 1, 0).
\end{aligned}
$$

The proposed ENP system is mainly composed of two parts: the 5-tuple generation part implemented in membrane *Generate* and the computation of the polynomial's value implemented in *Computation* membrane. When the *Generate* membrane generates a 5-tuple, it transfers it to the *Computation* membrane and waits for the computation to finish.

To illustrate the functioning of membrane *Generate* we will consider the gener-

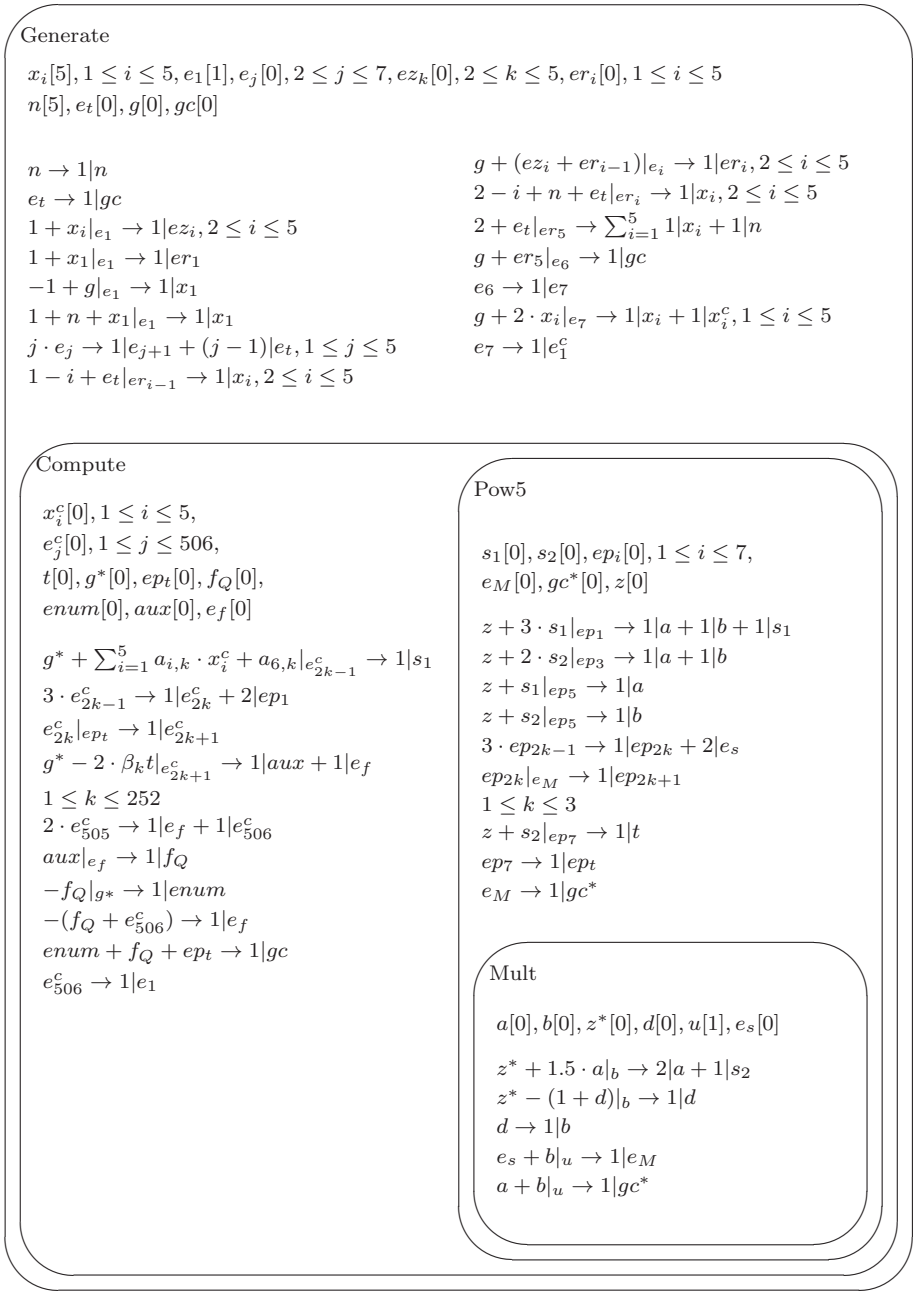

Figure 1. The ENP system from the proof of Theorem 4.1

ation of 3-tuples which is easier to follow. The construction can be extended to generate tuples of any length by the method described above.

The *Generate* membrane for the 3-tuple case will have the following variables and rules:

- Variables - the initial values are shown within the square brackets
  (1) $n[5]$ represents the current base minus one.
  (2) $x_1[5], x_2[5], x_3[5]$ are the digits of the number in base $n+1$. The rightmost digit is x_1.
  (3) $e_1[1], e_2[0], e_3[0], e_4[0], e_5[0]$ are variables used to control the execution flow of the system.
  (4) $er_1[0], er_2[0], er_3[0]$ correspond to the 3 digits and will store the number of zero valued digits to the right of the corresponding digit (including itself).

(5) $ez_2[0], ez_3[0]$ correspond to the 3 digits and store whether the digit is zero or not.
(6) $e_t[0]$ is an auxiliary variable. Let nrz be the number of zeros to the right of a digit needed in order to decrease or reset it. e_t will have at any given time either the value 0 or nrz of the current digit.
(7) $g[0]$ is an auxiliary variable which has the value 0 in all timesteps.
(8) $gc[0]$ is an auxiliary variable which acts as a garbage collector. Its value is not important and never used.

- The following 23 programs are used to generate 3-tuples

$$\{ \quad n \to 1|n,\ e_t \to 1|gc,$$
$$1 + x_1|_{e_1} \to 1|er_1,\ 1 + x_2|_{e_1} \to 1|ez_2,\ 1 + x_3|_{e_1} \to 1|ez_3,$$
$$-1 + g|_{e_1} \to 1|x_1,\ 1 - 2 + e_t|_{er_1} \to 1|x_2,\ 1 - 3 + e_t|_{er_2} \to 1|x_3,$$
$$1 + n + x_1|_{e_1} \to 1|x_1\ 2 - 2 + n + e_t|_{er_2} \to 1|x_2\ 2 - 2 + n + e_t|_{er_3} \to 1|x_3$$
$$g + (ez_2 + er_1)|_{e_2} \to 1|er_2,\ g + (ez_3 + er_2)|_{e_3} \to 1|er_3,$$
$$2 + e_t|_{er_3} \to 1|x_1 + 1|x_2 + 1|x_3 + 1|n,$$
$$g + er_3|_{er_4} \to 1|gc,$$
$$g + 2 \cdot x_i|_{e_5} \to 1|x_i + 1|x_i^c\ :\ 1 \le i \le 3,$$
$$e_1 \to 1|e_2,\ 2 \cdot e_2 \to 1|e_3 + 1|e_t,\ 3 \cdot e_3 \to 1|e_4 + 2|e_t,\ e_4 \to 1|e_5,\ e_5 \to 1|e_1^c\}$$

The process of generating a 3-tuple takes 5 timesteps (7 timesteps for the 5-tuple case and in general $k+2$ timesteps for tuples of length k). In the following we will describe the computation in each timestep. However, the programs $n \to 1|n$ and $e_t \to 1|gc$ which are always active will be omitted. $n \to 1|n$ ensures that the current value of the base is never lost, while $e_t \to 1|gc$ always resets the value of e_t to zero before any contributions are added.

The idea of the procedure is to process the digits from right to left. For each digit, it is determined if the digit needs to be decremented. A digit needs to be decremented if all the digits to its right are zero. The rightmost digit is always decremented. Also, if a digit and all the digits to its right are zero, then it has to be reset to the current value of the base minus 1, the value of n. When the leftmost digit is reset, the base is incremented and the values of all digits have to be incremented as well, in order to continue the generating process from the greatest number (tuple) in the new base.

- Step 1: $e_1 = 1$
  - Variables: $e_t = 0$, $er_1 = 0$, $er_2 = 0$, $er_3 = 0$, $ez_2 = 0$, $ez_3 = 0$.
  - The active programs are:
    (1) $1+x_1|_{e_1} \to 1|er_1, 1+x_2|_{e_1} \to 1|ez_2, 1+x_3|_{e_1} \to 1|ez_3$ are used to test what digits are zero. er_1, ez_2, ez_3 are 1 if the corresponding digit is 0, because the programs will not activate otherwise. Note that for the rightmost digit (x_1) the number of zeros to the right including itself (er_1) is the same as testing if the digit is 0. Therefore there is no ez_1 variable.
    (2) $-1 + g|_{e_1} \to 1|x_1$ decrements the value of x_1 regardless of its value.
    (3) $1 + n + x_1|_{e_1} \to 1|x_1$ resets the value of the rightmost digit x_1 if it is zero. It adds $n + 1$ to x_1 in order to compensate for the decrement of x_1 when it is 0.
    (4) $e_1 \to 1|e_2$ goes to the next step.
- Step 2: $e_2 = 1$

- Variables: $e_t = 0$, $er_1 \in \{0,1\}$, $er_2 = 0$, $er_3 = 0$, $ez_2 \in \{0,1\}$, $ez_3 \in \{0,1\}$.
- The active programs are:
  (1) $g+(ez_2+er_1)|_{e_2} \to 1|er_2$ is active, because of g, and computes the number of zeros (including itself) to the right of the second rightmost digit er_2. Note that ez_2 and er_1 are consumed in this program and are not produced by any other one. Thus, their value in the next timestep is 0.
  (2) $1-2+e_t|_{er_1} \to 1|x_2$ decrements digit x_2 if all digits to its right are zero, er_1 is 1. Because e_t is 0 the program activates only if the er_1 is 1.
  (3) $2 \cdot e_2 \to 1|e_3 + 1|e_t$ goes to the next step and computes e_t for the next digit.

- Step 3: $e_3 = 1$
  - Variables: $e_t = 1$, $er_1 = 0$, $er_2 \in \{0,1,2\}$, $er_3 = 0$, $ez_2 = 0$, $ez_3 \in \{0,1\}$.
  - The active programs are:
    (1) $g+(ez_3+er_2)|_{e_3} \to 1|er_3$ is active, because of g, and computes the number of zeros (including itself) to the right of the third rightmost digit er_3.
    (2) $1-3+e_t|_{er_2} \to 1|x_3$ decrements digit x_3 if all digits to its right are zero, er_2 is 2. Because e_t is 1 the program activates only if er_2 is 2.
    (3) $2-2+n+e_t|_{er_2} \to 1|x_2$ resets the second rightmost digit if all digits (including itself) to its right are 0, er_2 is 2. Again, because e_t is 1 the program activates only if er_2 is 2.
    (4) $3 \cdot e_3 \to 1|e_4 + 2|e_t$ goes to the next step and computes e_t for the next digit.
- Step 4: $e_4 = 1$
  - Variables: $e_t = 2$, $er_1 = 0$, $er_2 = 0$, $er_3 \in \{0,1,2,3\}$, $ez_2 = 0$, $ez_3 = 0$.
  - The active programs are:
    (1) $g + er_3|_{e_4} \to 1|gc$ erases the value of er_3 by transferring its value to the garbage collector variable gc. er_3 is not produced by any other program at this timestep.
    (2) $2-3+n+e_t|_{er_2} \to 1|x_2$ resets the third rightmost digit if all digits (including itself) to its right are 0, er_3 is 3. Again, because e_t is 2 the program activates only if er_3 is 3.
    (3) $2+e_t|_{er_3} \to 1|x_1+1|x_2+1|x_3+1|n$ activates if the null 3-tuple is reached, er_3 is 3. It changes the base and increases all digits in order to obtain the largest number (tuple) in the new base. Note that e_t is 2, therefore the $2+e_t$ is divisible by 4, the sum of contribution coefficients.
    (4) $e_4 \to 1|e_5$ goes to the next step.
- Step 5: $e_5 = 1$
  - Variables: $e_t = 0$, $er_1 = 0$, $er_2 = 0$, $er_3 = 0$, $ez_2 = 0$, $ez_3 = 0$.
  - The active programs are:
    (1) $g + 2 \cdot x_i|_{e_5} \to 1|x_i + 1|x_i^c \ : \ 1 \le i \le 3$ copy the generated 3-tuple to another membrane for processing.
    (2) $e_5 \to 1|e_1^c$ transfers the control to another membrane for processing.

It is easy to see that the construction can be extended to generate all tuples of natural numbers of any length. When a new tuple is generated it is transferred to the *Compute* membrane to be processed and it halts the generating process until e_1 is set to 1 again.

The computation part of the system is responsible for computing the value of a polynomial Q corresponding to a generated 5-tuple.

First, we need a membrane to multiply the two natural numbers as a repeated addition. This is implemented in *Mult* membrane. The best way to think of this membrane is as of a system with inputs and outputs. Variables a and b are the two inputs that must be multiplied and s_2 from the *Pow5* membrane is the output

variable. The computation is started by setting e_s to a positive value, and is finished when this value is transferred back to membrane *Pow5* in variable e_M. At every 2 timesteps the value of a is added to s_2 by program $z^* + 1.5a|_b \to 2|a + 1|s_2$. The addition is repeated b times in order to compute the product. b is decremented by using d, which alternatively takes the values 0 and -1. When b becomes zero programs $e_s + b|_u \to 1|e_M$, which transfers back the value of e_s to e_M, and $a + b|_u \to gc^*$, which erases the value of a, become active.

The *Pow5* membrane raises a value to its fifth power and uses the *Mult* membrane to perform the multiplication. This membrane can also be regarded as a system with inputs and outputs. Variable s_1 is the input, while the output is t. The computation starts when ep_1 becomes positive. The finish of the computation of the fifth power of s_1 is signaled by setting variable ep_t in the *Compute* membrane.

The square of the input value is computed by sending it as input to the *Mult* membrane. This is implemented by program $z + 3 \cdot s_1|_{ep_1} \to 1|a + 1|b + 1|s_1$ which also ensures that the value is not lost, but retained in variable s_1. Also, variable e_s is set by program $3 \cdot ep_1 \to 1|ep_2 + 2|e_s$ in order to start the multiplier membrane. Variable ep_2 and e_M are used to block the execution of the programs until the *Mult* membrane finishes. Please note that e_M will receive a value greater than ep_2, because e_M obtains its value from e_s, which in turn receives its value from ep_1. Thus, when the multiplication is finished, the value of e_M will be twice the value of ep_2. The program $ep_2|_{e_M} \to 1|ep_3$ is activated and ep_3 becomes 1. Setting ep_3 in turn triggers the next multiplication. Program $z + 2 \cdot s_2|_{ep_3} \to 1|a + 1|b$ is used to transfer the parameters to the *Mult* membrane. Since s_2 contained the square of the given input, the next multiplication will compute its 4th power. Again, the multiplier is started by setting e_s by program $3 \cdot ep_3 \to 1|ep_4 + 2|e_s$. The result is again collected in s_2 and variables ep_4 and e_M are used block the execution. When the computation of the 4th power is finished ep_5 is set by program $ep_4|_{e_M} \to 1|ep_5$. The last multiplication is performed at this stage between the value of s_1, which is the original one, and the value of s_2 which contains its 4th power. The two values are transferred again to the *Mult* membrane by programs $z + s_1|_{ep_5} \to 1|a$ and $z + s_2|_{ep_5} \to 1|b$. The end of the computation is signaled by setting e_M which together with ep_6 blocked the execution of programs. $ep_6|_{e_M} \to 1|ep_7$ is activated when the 5th power of the initial input value is obtained. The result is send back to the *Compute* membrane in variable t. The end of computation of membrane *Pow5* is signaled by setting ep_t by program $ep_7 \to 1|ep_t$. Some final clean up is performed by program $e_M \to 1|gc^*$ which erases e_M.

In turn, the *Pow5* membrane is used repeatedly by *Compute* membrane in order to compute the successive terms of the polynomial in the form (4). The process is the same as that used in the *Pow5* membrane. First, the linear combination $\sum_{i=1}^{5} a_{i,k} \cdot x_i^c + a_{6,k}$ is computed and send to be raised to the fifth power by program $g^* + \sum_{i=1}^{5} a_{i,1} \cdot x_i^c + a_{6,1}|_{e_1^c} \to 1|s_1$. g^* which is always zero ensures that the program is active. The start of the computation in *Pow5* is given by setting ep_1 to a positive value by program $3 \cdot e_1^c \to 1|e_2^c + 2|ep_1$. The final result is available when ep_t is set again. Note that again ep_t will have a greater value e_2^c because ep_t is equal to ep_1 which is twice the value of e_2^c. The result is stored in t, which is multiplied by a constant β_1 and added to the partial sum of the polynomial aux by program $g^* - 2 \cdot \beta_1 t|_{e_3^c} \to 1|aux + 1|e_f$. This process is repeated for all $m = 252$ terms of the polynomial. When the last term was computed and added to aux, e_{505}^c is set. Program $2 \cdot e_{505}^c \to 1|e_f + 1|e_{506}^c$ is used to unlock $aux|_{e_f} \to 1|f_Q$ which is used in testing of the computed value of the polynomial agaist zero. Note that the program is not active until $2 \cdot e_{505}^c \to 1|e_f + 1|e_{506}^c$ is executed (all terms were computed), because aux and e_f are equal. If the value of the polynomial f_Q is positive, it is

transferred in variable *enum* where the ENP system's results are collected. Some clean up is done by programs $-(f_Q + e^c_{506}) \to 1|e_f$ and $enum + f_Q + ep_t \to 1|gc$ to set the values of all variables in membrane *Compute* to the initial ones. Also, control is transferred back to the *Generate* membrane by $e^c_{506} \to 1|e_1$ in order to start generating the next 5-tuple.

The proposed construction uses only polynomials of degree 1 with at most 6 variables. Also, the number of membranes has been reduced to 4 due to repeated reuse of the *Pow5* and *Mult* membranes. Thus, the proof is complete. ■

Parts of the proposed membrane system, *Generate* membrane and *Pow5* membrane, were simulated and verified using SimP, an ENP systems simulator proposed in [4].

5. Conclusions

In the proof of Theorem 4.1 a method of reusing membranes was used in order to reduce the number of membranes in the system. It is, however, important to notice that it also constrained the system to perform the most important computations in a serial manner. In practice, it may be more convenient to have more membranes that compute in parallel, because it allows the underlying runtime environment to perform optimizations based on available hardware and software platform. It is also important to note that there are more rules dedicated to program control flow in the membrane system from Theorem 4.1 than there are in the one from Theorem 4 in [13].

Another important observation is that although computation can be done with polynomial production functions of degree 1, in some cases it is more convenient to use higher degree polynomials. However, most rules used for program flow control are of degree 1 and also, most rules with higher degree polynomial productions functions have few terms. These observations are relevant for optimizing the data structures and algorithms used for simulating ENP systems.

Acknowledgments.

The authors would like to thank Gheorghe Păun for useful discussions and suggestions regarding the universality of ENP systems. The authors would also like to thank the anonymous reviewers for useful comments and suggestions.

This work is supported by the Sectorial Operational Program Human Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the contract number SOP HRD/107/1.5/S/82514.

References

[1] C. Buiu, A. B. Pavel, C. I. Vasile, and I. Dumitrache. Perspectives of Using Membrane Computing in the Control of Mobile Robots. In *Proceedings of Beyond AI: Interdisciplinary Aspects of Artificial Intelligence(BAI 2011), Pilsen, Czech Republic*, pages 21–26, December 2011.

[2] C. Buiu, C. I. Vasile, and O. Arsene. Development of Membrane Controllers for Mobile Robots. *Information Sciences*, 187:33–51, March 2012. doi: 10.1016/j.ins.2011.10.007.

[3] Y. Matijasevitch, editor. *Hilbert's Tenth Problem* . MIT Press, Cambridge, London, 1993.

[4] A. B. Pavel. Membrane Controllers for Cognitive Robots. Master's thesis, Department of Automatic Control and System Engineering, Politehnica University of Bucharest, Romania, February 2011.

[5] A. B. Pavel, O. Arsene, and C. Buiu. Enzymatic Numerical P Systems - a New Class of Membrane Computing Systems. In *The IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010) Liverpool*, pages 1331–1336, September 2010.

[6] A. B. Pavel and C. Buiu. Using Enzymatic Numerical P Systems for Modeling Mobile Robot Controllers. *Natural Computing*, in press. doi: 10.1007/s11047-011-9286-5.
[7] A. B. Pavel, C. I. Vasile, and I. Dumitrache. Robot Localization Implemented with Enzymatic Numerical P Systems. In *Living Machines 2012: The International Conference on Biomimetic and Biohybrid Systems*, Proceedings in Lecture Notes in Artificial Intelligence, Barcelona, Spain, July 2012. Springer.
[8] Gh. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.
[9] Gh. Păun. *Membrane Computing - An Introduction*. Springer-Verlag, Berlin, 2002.
[10] Gh. Păun and R. Păun. Membrane Computing and Economics: Numerical P Systems. *Fundamenta Informaticae*, pages 213–227, 2004.
[11] Gh. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
[12] G. Rozenberg and A. Salomaa. *Cornerstones of Undecidability* . Prentice-Hall, 1994.
[13] C. I. Vasile, A. B. Pavel, I. Dumitrache, and Gh. Păun. On the Power of Enzymatic Numerical P Systems. *Acta Informatica*, in press. doi: 10.1007/s00236-012-0166-y.