Acta Informatica manuscript No.
(will be inserted by the editor)

On the Power of Enzymatic Numerical P Systems

Cristian Ioan Vasile · **Ana Brânduşa Pavel** ·
Ioan Dumitrache · **Gheorghe Păun**

Received: date / Accepted: date

Abstract We study the computing power of a class of numerical P systems introduced in the framework of autonomous robot control, namely enzymatic numerical P systems. Three ways of using the evolution programs are investigated: sequential, all-parallel and one-parallel (with the same variable used in *all* programs or in only *one*, respectively); moreover, both deterministic and non-deterministic systems are considered. The Turing universality of some of the obtained classes of numerical P systems is proved (for polynomials with the smallest possible degree, one, also introducing a new proof technique in this area, namely starting the universality proof from the characterization of computable sets of numbers by means of register machines). The power of many other classes remains to be investigated.

Keywords Membrane computing · Numerical P System · Turing universality · Register machine

C.I. Vasile, A.B. Pavel, I. Dumitrache
Department of Automatic Control and Systems Engineering
Politehnica University of Bucharest
Splaiul Independenţei 313, 060042 Bucharest, Romania
E-mail: {cvasile, apavel, idumitrache}@ics.pub.ro
{cristian.vasile, ana.pavel, ioan.dumitrache}@acse.pub.ro

Gh. Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
E-mail: george.paun@imar.ro

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: gpaun@us.es

1 Introduction

In the last years, several classes of computing devices – called *P systems* – were introduced inspired from the cell architecture and functioning, and vividly investigated in the framework of *membrane computing* – general references will be given below.

Numerical P systems are a class of computing models inspired both from the cell structure and from the economics: numerical variables evolve in the compartments of a cell-like structure by means of so-called *production–repartition programs*. The variables have a given initial value and the production function is usually a polynomial, whose value for the current values of variables is distributed among variables in certain compartments (close to the place where the polynomial is evaluated – see precise definitions in the next section) according to the "repartition protocol". In this way, the values of variables evolve; all values taken by a specified variable are said to be computed by the P system.

These computing devices were introduced in [9], where their computational completeness (equivalence with the computing power of Turing machines) was proven, by making use of the characterization of Turing computable sets of numbers as the positive values of polynomials with integer coefficients, [5]. The results in [9], as well as further connections between membrane computing and economics, are recalled in Chapter 23.6 of [10]. This Handbook provides an overall image of membrane computing at the level of year 2009; further details can be found in [8], with the updated information available at the website [16]. A simulator for numerical P systems, together with relevant examples, are presented in [1] and [3].

Note the important difference between numerical P systems and all classes of P systems usually investigated in membrane computing: in general, multisets of symbol objects are processed in the membranes, by means of biochemically inspired rules, while in numerical P systems we deal with numerical variables, processed in an algebraic way. At a more technical level, there also are several other differences between (the way of working of) numerical P systems and usual P systems: (i) the evolution programs are used in [9] *in the sequential way* (in each step, each membrane uses only one program), while in the membrane computing the most investigated semantics is the maximally parallel one (at a given step one uses a multiset of rules which is maximal in the inclusion sense), and (ii) the result of a computation *is not defined by halting*, but by collecting all values assumed by a specified variable, in all computations, with the computations not necessarily halting.

In [9] and [10] one uses the numerical P systems only in the generating mode. However, numerical P systems were recently used in a series of papers (see references in [3]) for implementing controllers for mobile robots (an idea first mentioned in [2]), and in this framework the P systems work in the computing mode: an input is introduced in the form of the values of some variables and an output is produced, as the value of other variables. Clearly, the systems should behave in a deterministic way – another difference from the case of [9], where one deals with the non-deterministic case.

Furthermore, in the robot control context, the so-called *enzymatic* numerical P systems were introduced and used, [12], [13], [14]. Such systems correspond to *catalytic P systems* in the "general" membrane computing: a reaction takes place only

in the presence of a catalyst. Here, the catalyst (enzyme) is a variable attached to an evolution program; the program is used only if the value of the enzyme is strictly greater than the smallest value of variables involved in the production polynomial.

In the robot control context, the production-repartition programs are used in parallel – actually, in a way which will be called here *all-parallel*: all programs in a compartment of the membrane structure are used simultaneously, with each variable participating in all programs where it appears. A variant, closer to the maximal parallelism in membrane computing, is to select the programs to be used in parallel in such a way that each variable participates in only one of the chosen programs – we call this *one-parallel* evolution.

A large variety of classes of numerical P systems appears in this way: (1) enzymatic or non-enzymatic, (2) deterministic or non-deterministic, (3) sequential, all-parallel, one-parallel. Still, we can add: (4) generating, computing, accepting (a number is introduced in the system, as the value of a variable, and it is accepted if a certain condition is met, e.g., a specified variable gets the value 0), and (5) selecting as results only the positive values of the output variable or accepting all values, but making sure that the output variable assumes only positive values. This is a subtle difference: in the first case, we just intersect the set of values of a variable with $\mathbf{N}$, the set of natural numbers, in the second case we have a *property* of the system. The former case reminds of the Chomsky grammars and extended Lindenmayer systems, where a terminal alphabet is used in order to squeeze the generated language from the language of sentential forms, but an important difference is that here "the terminal" sets of numbers, $\mathbf{N}$, is fixed, is not at our choice, as in Chomsky grammars and L systems.

A plethora of classes of numerical P systems appears, waiting for a systematic investigation. We settle here only a few cases, but we bring into the stage some proof techniques which will probably be useful in investigating also other cases.

Two are the main new ideas: (i) working in the deterministic mode even in the generative case (and this is possible, because we do not define successful computations by halting), and (ii) using register machines, [6], as the starting characterization of Turing computable sets of numbers (instead of their characterization as sets of positive solutions of Diophantine equations); this proof strategy, of simulating register machines, is widely used in membrane computing, but it is the first time used for numerical P systems.

In all cases discussed here, we make an essential use of the enzymatic control. Interesting enough, when the enzymatic control is supplemented with the maximal use of productions, in the two variants of parallelism suggested above, we can improve the universality results from [9]: polynomials of degree at most two for the deterministic all-parallel case and of degree one for the non-deterministic one-parallel case are sufficient.

Besides the study of the many classes of numerical P systems whose power is not characterized here, two important directions of research remain to be explored: bringing to numerical P systems other notions from the general membrane computing area (e.g., other ways to control the rule application, or ways to also evolve the membrane structure), and to find non-universal classes of numerical P systems (e.g.,

with decidable properties). Of course, the usefulness of such numerical P systems, with an enhanced structure, for the robots' control, also remains to be examined.

2 Numerical P Systems

We do not recall here elements of membrane computing, the reader is assumed to have some familiarity with this research area, from [7], [8], [10], but we introduce formally the numerical P systems and then the enzymatic version of them.

As basic notations, we use $\mathbf{N}$ to denote the set $\{0,1,2,3,\dots\}$ of natural numbers, and *NRE* the family of computable sets of numbers, zero excluded (RE comes from "recursively enumerable").

In order to define numerical P systems we need a series of ingredients.

The basic one is the cell-like *membrane structure* (hierarchical, hence described by a tree and represented mathematically by a well-formed expression of matching labeled parentheses), with the membranes labeled in a one-to-one manner with elements of an alphabet H. In the compartments of the membrane structure, we have *variables*; those from region i are written in the form $x_{j,i}$, $j \geq 1$. The value of $x_{j,i}$ at time $t \in \mathbf{N}$ is denoted by $x_{j,i}(t)$. In general, these values can be real numbers, but here we only work with integers, positive or negative.

In order to evolve the values of variables, we use *programs*, composed of two components, a *production function* and a *repartition protocol* (to make the use of programs more explicit, we separate the two components by an arrow, like in rewriting rules). The former can be any function with variables from a given region – here we consider only polynomials with integer coefficients. Using such a function we compute a *production value* of the region at a given time, depending on the values of variables at that time. This value is distributed to variables from the region where the program resides, and to variables in its upper and lower compartments (for a given region i, let $v_1,\dots,v_{n_i}$ be all these variables) according to the repartition protocol associated with the used production function. The repartition protocols are of the form

$$c_1|v_1 + c_2|v_2 + \cdots + c_{n_i}|v_{n_i},$$

where $c_1,\dots,c_{n_i}$ are natural numbers. The idea is that the coefficients $c_1,\dots,c_{n_i}$ specify the proportion of the current production value distributed to each variable $v_1,\dots,v_{n_i}$.

Formally, for a program (the lth program from region i)

$$F_{l,i}(y_{1,i},\dots,y_{k_l,i}) \to c_{l,1}|v_1 + c_{l,2}|v_2 + \cdots + c_{l,n_i}|v_{n_i}, \text{ for } \{y_{1,i},\dots,y_{k_l,i}\} \subseteq Var_i,$$

let

$$C_{l,i} = \sum_{s=1}^{n_i} c_{l,s}.$$

For $t \geq 0$, we compute the *production value* at time t

$$\tau_{l,i}(t) = F_{l,i}(y_{1,i}(t),\dots,y_{k_l,i}(t))$$

and then

$$q_{l,i}(t) = \frac{\tau_{l,i}(t)}{C_{l,i}}.$$

The value $q_{l,i}(t)$ represents the "unitary portion" to be distributed to variables $v_1, \ldots, v_{n_i}$ proportionally with $c_{l,1}, \ldots, c_{l,n_i}$. Thus, v_s will receive $q_{l,i}(t) \cdot c_{l,s}, 1 \leq s \leq n_i$. The variables involved in the production function are reset to zero after computing the production value; a variable not involved in a production function retains its value. After repartition, the quantities assigned to each variable from the repartition protocol are added to the current value of these variables (starting with 0 for the variables which were reset by a production function).

Thus, a *numerical P system* is a construct of the form

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

where m is the degree of the system (the number of membranes), H is an alphabet of labels for membranes in μ (when possible, we use $H = \{1, 2, \ldots, m\}$), μ is a membrane structure with m membranes labeled injectively by elements of H, Var_i is the set of variables from region i, Pr_i is the set of programs from region i (all sets Var_i, Pr_i are finite), $Var_i(0)$ is the vector of initial values for the variables in region i, and x_{j_0,i_0} is a distinguished variable (from a distinguished region i_0), which provides the result of computations.

Each program is of the form specified above: $pr_{l,i} = (F_{l,i}(y_{1,i}, \ldots, y_{k_l,i}) \rightarrow c_{l,1}|v_1 + c_{l,2}|v_2 + \cdots + c_{l,n_i}|v_{n_i})$ denotes the l-th program from region i, where $\{y_{1,i}, \ldots, y_{k_l,i}\} \subseteq Var_i$ and $v_1, \ldots, v_{n_i}$ are all variables from region i, the upper region, and the immediately inner regions.

Such a system evolves in the way informally described before. Initially, the variables have the values specified by $Var_i(0), 1 \leq i \leq m$. A transition from a configuration at time instant t to a configuration at time instant $t+1$ is made by (i) choosing non-deterministically one program from each region, (ii) computing the value of the respective production function for the values of local variables at time t, and then (iii) computing the values of variables at time $t+1$ as indicated by repartition protocols. A sequence of such transitions forms a computation. For a computation σ, let $N^+(\sigma)$ be the set of positive values assumed by x_{j_0,i_0} during the computation σ. (Note that the computation may continue forever.) Let $N^+(\Pi)$ be the union of $N^+(\sigma)$, for all computations σ in Π. (Also the result 0 can be considered, but we proceed here as usual in language and automata theory, where the empty word is omitted when comparing the power of two computing devices – this extends to membrane computing to omitting the number 0.)

A delicate problem concerns the issue whether the production value is divisible by the total sum of coefficients c_j from the corresponding repartition protocol. In this paper, we assume that this is the case, i.e., we only deal with systems whose programs have this property – this corresponds to the *div* case in [9], but we do not further specify the parameter *div* in what follows. Variants were introduced in [9]; taking this aspect into consideration introduces one further classification criterion for numerical P systems, besides those mentioned in the Introduction.

The family of sets of numbers $N^+(\Pi)$ computed by numerical P systems Π 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 $N^+P_m(poly^n(r),seq),\ m \geq 1, n \geq 0, r \geq 0$, where the fact that we work in the sequential mode (in each step, only one program is applied in each region) is indicated by *seq*. If one of the parameters m,n,r is not bounded, then it is replaced by $*$. (Both in $N^+(\Pi)$ and in $N^+P_m(poly^n(r),seq)$, the superscript $+$ indicates the fact that as the result of a computation we only consider positive natural numbers, zero excluded. If any value of x_{j_0,i_0} is accepted, then we remove the superscript $+$.)

We only recall here the following result from [9] (called Corollary 4.1 there).

Theorem 1 $NRE = N^+P_8(poly^5(5),seq) = N^+P_7(poly^5(6),seq)$.

The proof is based on the characterization of recursively enumerable sets of numbers as positive values of polynomials with integer coefficients, [5]. It is formulated as an open problem in [9] the question "whether or not the positive values of a variable can be selected inside the system (changing in a minimal way the definition), not by means of an external condition." In the next section we show that this question can be affirmatively answered, by using enzymes in controlling the application of evolution programs.

3 Enzymatic Numerical P Systems

Enzymatic numerical P systems (in short, EN P systems) use both evolution programs as introduced above and programs of the form

$$F_{l,i}(y_{1,i},\ldots,y_{k_l,i})|_{e_{j,i}} \to c_{l,1}|v_1 + c_{l,2}|v_2 + \cdots + c_{l,n_i}|v_{n_i},$$

where $e_{j,i}$ is a variable from Var_i different from $y_{1,i},\ldots,y_{k_l,i}$, and from $v_1,\ldots,v_{n_i}$. Such a program can be applied at a time t only if $e_{j,i}(t) > \min(y_{1,i}(t),\ldots,y_{k_l,i}(t))$. (A slightly more complex definition is considered in [12] and [14] where: $e_{j,i}(t) > \min(|y_{1,i}(t)|,\ldots,|y_{k_l,i}(t)|)$. Considering the absolute value of the variables, instead of their real values, simplifies the design of the membrane structures used to compute certain functions, such as *cos* and *sin*, as power series, but here we work only with the simpler case defined above. We also use here a notation different from that in [12], writing the enzyme in the same way as the promoters are written in multiset rewriting rules, see [8].) Note that in order to apply the program it is sufficient that *one variable* involved in a production function has the current value strictly smaller than the value of the enzyme variable. The enzyme cannot evolve by means of the associated program, but it can evolve by means of other programs, and can receive "contributions" from other programs and regions.

Because the enzymes are usual variables, playing a different role only "locally", in specified programs, we do not consider their set separated, hence the general writing of an enzymatic numerical P systems is the same as that of a numerical P system – only the form of programs can be different.

Using enzymes introduces a checking possibility in our systems (we compare the value of the enzyme with the values of variables from the associated program), and this suggests the possibility of choosing the positive values of the output variable

"inside the system". Indeed, the following result holds true (for the non-deterministic sequential case; *enz* indicates the use of enzymes):

Theorem 2 $NRE = NP_7(poly^5(5), enz, seq)$.

Proof We will modify the proof given in [9] to the previous Theorem 1; the computations proceed in the same way, but at the end we use the enzyme control in order to select as results only the positive values of the output variable. Specifically, following [5], for each set $Q \in NRE$ we consider a polynomial $f_Q(x_1,\ldots,x_n)$ with integer coefficients such that $r \in Q$ iff $r = f_Q(a_1,\ldots,a_n) \cap \mathbf{N}$ for some $a_1,\ldots,a_n \in \mathbf{N}$. It is known that polynomials f_Q exists of degree 5 using 5 variables (see, e.g., page 109 of [15]; polynomials with other combinations *degree – number of variables* can be chosen, with a trade-off between the two parameters, but the point here is not the value of parameters, but the fact that we can select the positive values of the output variable in an internal way, not in an external one). Let us consider such a polynomial f_Q.

We construct the following enzymatic numerical P system (for the sake of the readability, we also present it graphically, in Figure 1; note in this figure the way the initial values of variables are given in square brackets for each variable):

$$
\begin{aligned}
\Pi &= (7, H, \mu, (Var_s, Pr_s, Var_s(0)), (Var_0, Pr_0, Var_0(0)),\\
&\quad (Var_1, Pr_1, Var_1(0)), (Var_2, Pr_2, Var_2(0)), \ldots, (Var_5, Pr_5, Var_5(0)), x_{2,0}),\\
H &= \{s, 0, 1, 2, \ldots, 5\},\\
\mu &= [_s [_0]_0 [_1]_1 [_2]_2 \cdots [_5]_5]_s,\\
Var_s &= \{x_{1,s}, x_{2,s}, \ldots, x_{5,s}\},\\
Pr_s &= \{-f_Q(x_{1,s}, x_{2,s}, \ldots, x_{n,s}) \to 1 | x_{1,0}\},\\
Var_s(0) &= (0, 0, \ldots, 0),\\
Var_0 &= \{x_{1,0}, x_{2,0}, e_{1,0}, g_0\},\\
Pr_0 &= \{-x_{1,0}|_{e_{1,0}} \to 1 | x_{2,0},\ \ x_{1,0} \to 1 | g_0,\ x_{2,0} \to 1 | g_0\},\\
Var_0(0) &= (0, 0, 0, 0),\\
Var_i &= \{x_{1,i}\},\\
Pr_i &= \{x_{1,i} \to 1 | x_{1,i},\ \ x_{1,i} + 1 \to 1 | x_{1,i},\ \ x_{1,i} \to 1 | x_{i,s}\},\\
Var_i(0) &= (0),\ \text{for all } i = 1, 2, \ldots, 5.
\end{aligned}
$$

This system works as follows (we recall also a few details from [9]). All variables starts from 0. In each membrane $1, 2, \ldots, 5$ there are three programs; the first one leaves the local variable unchanged, the second one increases it by one, the third program moves the value of the local variable to the corresponding variable from compartment s. Specifically, compartment i provides a value to variable $x_{i,s}, 1 \leq i \leq 5$. By non-deterministically choosing the programs to apply, in compartments $1, 2, \ldots, 5$ we can produce all vectors $(a_1, a_2, \ldots, a_5)$ of natural numbers.

In each step, in compartment s we compute a value of polynomial f_Q, therefore we can compute such values for all vectors of natural numbers. The respective values are immediately transferred to variable $x_{1,0}$ from compartment 0, multiplied with -1.

The only enzymatic program is here. The value of $e_{1,0}$ is zero and it is never changed. If the value of $x_{1,0}$ is strictly smaller – which means that $f_Q(x_{1,s}, \ldots, x_{5,s})$

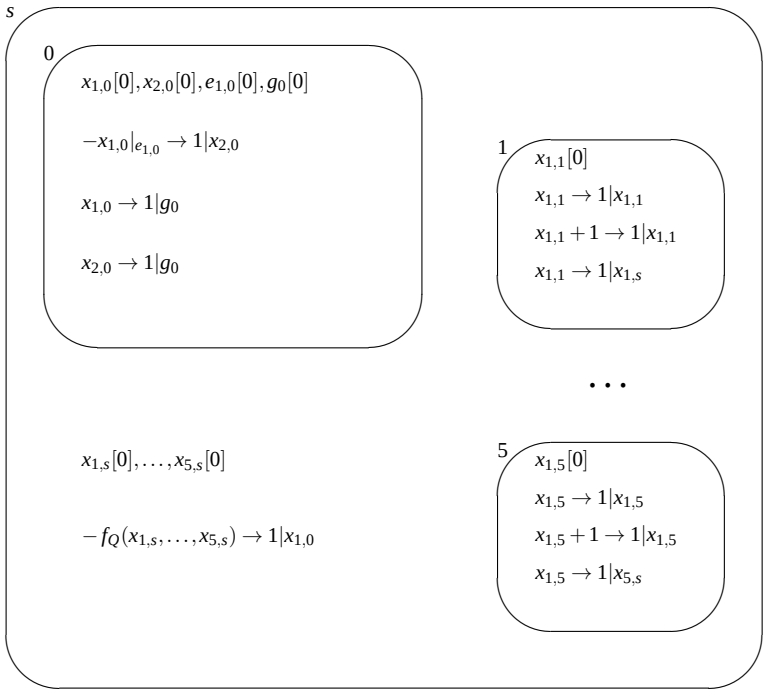

Fig. 1 The P system from the proof of Theorem 2

was strictly positive – then its value is transferred, again with a changed sign, to the output variable $x_{2,0}$. In the opposite case, the program $x_{1,0} \to 1|g_0$ must be used, and the value of $x_{1,0}$ is "lost" in the "garbage variable" g_0. If this program is used also for the case of having a positive value for $x_{1,0}$, then again the result is "lost", nothing wrong happens. Also the variable $x_{2,0}$ should be reset to zero, and this is done again with the help of variable g_0.

The work of the system continues forever, and variable $x_{2,0}$ takes all positive values of f_Q, which means that $N(\Pi) = Q$, hence $Q \in NP_7(poly^5(5), enz, seq)$. This concludes the proof. □

Starting from a polynomial of degree four (there are characterizations of *RE* by such polynomials, see [15]), the degree of the polynomial used in the previous proof can be decreased to four. However, if we pass to numerical P systems working in a parallel manner, then the maximal improvement (from this point of view) can be obtained: polynomials of degree one suffice. As expected, this is obtained at the expense of increasing other parameters – in this case, the number of membranes (however, pleasantly enough, we need only polynomials with at most two variables).

4 Parallel EN P Systems

We continue to work in the "classic" setup (the non-deterministic generative mode), but we introduce a change in the way of using the programs, namely, we consider

the "standard" way in membrane computing, i.e., the maximal parallelism: in each membrane, at each step, we use a maximal set of programs (programs are selected non-deterministically, and a set of programs is applied only if it is maximal, no further program can be added to it in such a way that the new set is still applicable), with the restriction that a variable can appear only in *one* production function of a program which is applied.

In the enzymatic case, it is important to note that an enzyme which controls the use of a program can evolve at the same time by means of another program (but not by the program whose application it makes possible).

The functioning of such systems is the same as above: we start from the initial values of variables and we evolve them according to the programs; the values assumed by a distinguished variable during any computation form the set computed by the system. We denote as above by $N(\Pi)$ the set of numbers computed by a system Π and by $NP_m(poly^n(r), enz, oneP),\ m \geq 1,\ n \geq 0,\ r \geq 0$, the family of sets of numbers $N(\Pi)$ computed by enzymatic numerical P systems working in the parallel mode described above, 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; if one of the parameters m, n, r is not bounded, then it is replaced by $*$. (We no longer write $N^+(\Pi)$, because, as in Theorem 2, we can select inside the system the positive values of the output variable.)

The following somewhat surprising result holds true:

Theorem 3 $NRE = NP_*(poly^1(2), enz, oneP)$.

Proof We start from the characterization of sets in *NRE* by means of register machines. Such a device is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the finite set of instruction labels, l_0 is the start label (associated with an ADD instruction), l_h is the halt label (assigned to instruction `HALT`), and I is the set of instructions; each element of H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

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

A register machine M computes (generates) a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by labels (and made possible by the content of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize *NRE*; moreover, machines with three registers suffice (see, e.g., [6]).

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty.

Consider a register machine $M = (m, H, l_0, l_h, I)$. We construct an enzymatic numerical P system Π in the following way. Let us assume that $H = \{l_0, l_1, \ldots, l_n\}$, for some $n \geq 1$, and that $l_n = l_h$. The membrane structure of Π will be

$$\mu = [_s [_{l_0}]_{l_0} [_{l_1}]_{l_1} \ldots [_{l_n}]_{l_n}]_s,$$

that is, we associate one membrane l_i to each label $l_i \in H$ (hence to each instruction in I; these membranes are also called *modules*), all of them placed in the skin membrane, labeled with s. Each membrane $l_i, 0 \leq i \leq n$, simulates the corresponding instruction in I. To this aim, membrane l_i contains so-called *register variables* (denoted by x with subscripts) associated with the values of the three registers of M (for instance, variables $x_{r,i}$, for $r = 1,2,3$, present in each membrane $l_i, 0 \leq i \leq n$, represent the values of registers $r = 1,2,3$ of M), *enzymes* (denoted by e with subscripts; each membrane l_i contains a "trigger enzyme" e_i), so-called *dummy variables* (denoted by w and z with subscripts), and *garbage variable* (denoted by g with subscripts). Also the skin region contains register variables, enzymes, and dummy variables. Initially, all these variables are zero, with the exception of enzyme e_0 (from membrane l_0), which is equal to 1. In general, when the enzyme e_i associated with a membrane l_i is 1, that membrane/module is *active*, its rules can be applied and local variables can evolve. Because the registers of M take only values in $\mathbf{N}$, all variables of our system will have values in $\mathbf{N}$.

A general trick for using a program in the presence of an enzyme with value 1 is to involve in the production function a dummy variable, which is always equal to zero. In this way, as long as the enzyme is 0, the program cannot be used, but when the enzyme becomes 1, the program can be applied.

After using a module (hence simulating an ADD or SUB instruction), all variables are reset to zero, with the exception of the variables $x_{r,i}$ (the register values), and of one enzyme which determines the next instruction to simulate.

The modules communicate to each other by means of variables placed in the skin region: if membrane l_i needs to move $x_{r,i}, 1 \leq r \leq 3$, to membrane l_j, then membrane l_i first move $x_{r,i}$ to $x_{(r,j),s}$, and from membrane s this variables moves its value to $x_{r,j}$; simultaneously, e_i increases a variable $e_{j,s}$ to 1, which moves its value to e_j in membrane l_j, thus activating this membrane.

When the computation in M halts, we reach the membrane associated with l_h, coming from an ADD or SUB module which moved the values of all variables $x_{r,i}, 1 \leq r \leq 3$, into membrane l_h (only $x_{1,i}$ is different from zero); in this way, $x_{1,n}$ – the output variable – gets the value of the first register of M. Immediately, $x_{1,n}$ is reset to 0.

With these explanations in mind, we give now formally the system Π. (Note that the values of the registers are always positive, hence we do not have to take care of selecting the positive values of the output variable.)

$$\begin{aligned}
\Pi &= (n+2, H', \mu, (Var_s, Pr_s, Var_s(0)), (Var_0, Pr_0, Var_0(0)),\\
&\quad (Var_1, Pr_1, Var_1(0)), \ldots, (Var_n, Pr_n, Var_n(0)), x_{1,n}),\\
H' &= \{s, l_0, l_1, l_2, \ldots, l_n\},\\
\mu &= [_s [_{l_0}]_{l_0} [_{l_1}]_{l_1} \ldots [_{l_n}]_{l_n}]_s,
\end{aligned}$$

$$
\begin{aligned}
Var_s &= \{x_{(t,i),s},\ w_{(t,i),s}\ :\ 1 \le t \le 3, 0 \le i \le n\} \cup \{e_{i,s}\ :\ 0 \le i \le n\},\\
Pr_s &= \{x_{(t,i),s} + w_{(t,i),s}|_{e_{i,s}} \to 1 | x_{t,i}\ :\ 1 \le t \le 3, 0 \le i \le n\}\\
&\quad \cup \{e_{i,s} \to 1 | e_i\ :\ 0 \le i \le n\},\\
Var_s(0) &= (0,0,\ldots,0),\\
Var_i &= \{x_{t,i}, w_{t,i}\ :\ 1 \le t \le 3\} \cup \{e_i, e_i', e_i'', z_i\}, \text{for all } 0 \le i \le n-1,\\
Var_i(0) &= (0,0,\ldots,0), \text{ for all } i = 0,1,2,\ldots,n-1, \text{ with the exception of}\\
e_0 &= 1,\\
Var_n &= \{x_{r,n}\ :\ 1 \le r \le 3\} \cup \{e_n\},\\
Pr_n &= \emptyset,\\
Var_n(0) &= (0,0,0,0),
\end{aligned}
$$

with the programs in membranes l_i as follows:

1. If $l_i : (\texttt{ADD}(r), l_j, l_k)$, then

$$
\begin{aligned}
Pr_i = \{&e_i \to 1 | e_i',\ e_i \to 1 | e_i'',\\
&x_{r,i} + w_{r,i} + 1|_{e_i'} \to 1 | x_{(r,j),s},\ x_{r,i} + w_{r,i} + 1|_{e_i''} \to 1 | x_{(r,k),s}\}\\
\cup\ &\{x_{t,i} + w_{t,i}|_{e_i'} \to 1 | x_{(t,j),s},\ x_{t,i} + w_{t,i}|_{e_i''} \to 1 | x_{(t,k),s}\ :\ t \in \{1,2,3\} - \{r\}\}\\
\cup\ &\{e_i' \to 1 | e_{j,s},\ e_i'' \to 1 | e_{k,s}\},
\end{aligned}
$$

2. If $l_i : (\texttt{SUB}(r), l_j, l_k)$, then

$$
\begin{aligned}
Pr_i = \{&z_i + 1|_{x_{r,i}} \to 1 | e_i',\ x_{r,i} + 1|_{e_i} \to 1 | e_i'',\ e_i \to 1 | g_i,\\
&x_{r,i} + w_{r,i} - 1|_{e_i'} \to 1 | x_{(r,j),s}\}\\
\cup\ &\{x_{t,i} + w_{t,i}|_{e_i'} \to 1 | x_{(t,j),s}\ :\ t \in \{1,2,3\} - \{r\}\}\\
\cup\ &\{x_{t,i} + w_{t,i}|_{e_i''} \to 1 | x_{(t,k),s}\ :\ 1 \le t \le 3\}\\
\cup\ &\{e_i' \to 1 | e_{j,s},\ e_i'' \to 1 | e_{k,s}\}.
\end{aligned}
$$

The membrane l_i associated with an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ works as follows. In each step, irrespective of the stage of the computation, the programs $e_i \to 1|e_i'$, $e_i \to 1|e_i''$ can be used, but they change nothing as long as the value of e_i is zero. When the module is activated, and e_i becomes 1, by non-deterministically using one of the above mentioned programs (and only one, as we work in the *oneP* mode), one of the enzymes e_i', e_i'' becomes 1, and this makes possible the use of the subsequent programs. Corresponding to the choice of e_i' or e_i'', the value of $x_{r,i}$, augmented with 1, passes to $x_{(r,j),s}$ or to $x_{(r,k),s}$, respectively, and, at the same time, $e_{j,s}$ or $e_{k,s}$ becomes 1. Concomitantly, the variables $x_{t,i}, t \in \{1,2,3\} - \{r\}$, pass their values to $x_{(t,j),s}$ or $x_{(t,k),s}$. From the skin region, these variables pass their values to the corresponding variables from membranes l_j or l_k. All variables of membrane l_i, are reset to zero. The ADD instruction is correctly simulated, only one enzyme continues to have the value 1, namely in that membrane l_j, l_k where also the values of the registers were moved.

For an easier reference, the module associated with a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is given also in a graphical form in Figure 2.

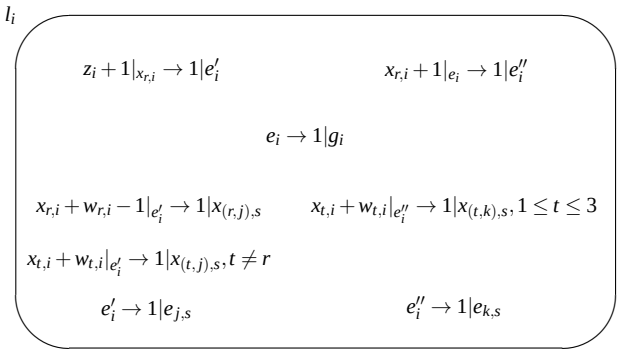

Fig. 2 The SUB module.

As long as both $x_{r,i}$ and e_i are zero, no rule can be applied in membrane l_i. If $x_{r,i}$ becomes at least one, the rule $z_i + 1|_{x_{r,i}} \to 1|e'_i$ can be used. Similarly, if e_i becomes 1 and at the same time, $x_{r,i} = 0$, then the rule $x_{r,i} + 1|_{e_i} \to 1|e''_i$ can be used. In this way, the rules in membrane l_i written in the first row in Figure 2 check whether $x_{r,i}$ is greater than zero or not; the left column of rules in this figure corresponds to the case $x_{r,i} > 0$, and the right column corresponds to the case $x_{r,i} = 0$. In the first case, the variable (enzyme) e'_i becomes 1, in the latter case the variable e''_i becomes 1. Simultaneously, e_i is returned to zero.

In the case $x_{r,i} > 0$, we subtract 1 from $x_{r,i}$ (and move the result to $x_{(r,j),s}$), we move $x_{t,i}$ for $1 \le t \le 3, t \neq r$, to $x_{(t,j),s}$, and, simultaneously, e'_i moves its value to $e_{j,s}$ and it is reset to zero.

In the case $x_{r,i} = 0$ we simply move the variables $x_{t,i}$ to $x_{(t,k),s}$, for all $t = 1,2,3$, and we make $e_{k,s} = 1$, while returning e''_i to zero.

The variables from the skin region move now their values to the corresponding variables from membrane l_j or l_k.

The simulation of the SUB instruction is correct, we continue as in a computation of the register machine.

The combination of all these modules ensures the correct simulation of all computations in M (if the computation in M does not halt, then the variable $x_{1,n}$ remains always zero), therefore, $N(M) = N(\Pi)$. The observation that we use only linear production mappings involving at most two variables completes the proof. □

Starting the previous proof from a universal register machine (as given, e.g., in [4]), the number of membranes can be bounded – finding a precise (small) bound remains as a research issue.

5 Deterministic EN P Systems

As mentioned in the Introduction, when used for robot control, EN P systems should work in the computing mode (certain parameters concerning the relations between

the robot and its environment, other robots included, are provided as an input, and the P system produces an output which is used for specifying the robot action), and this input-output passage is deterministic. The determinism is obtained (e.g., in [12]) by two means: making use of the enzymatic control and making use of a "total parallelism" – all programs are used, with a variable which occurs in several programs being used with its current value in all of them. The enzymatic mechanism assures that each computational step is completely determined by the previous steps. The enzymes control the program flow, they are used in stop conditions and for synchronization between membranes. The deterministic EN P systems model was used to design and implement robot behaviors like obstacle avoidance and odometric localization (see, e.g., [13] and [14]). Some robot behaviors were also implemented using classical numerical P systems with only one rule per membrane, [3], but the membrane structures obtained were far more complicated and less efficient from the computational point of view, compared to the ones modeled with EN P systems. Anyhow, the deterministic behavior of P systems is essential when they have to be implemented as computer programs or robot controllers.

Let us use *allP* to indicate the use of evolution programs in the parallel mode mentioned above (if two or more programs which are enabled at a computation step, i.e., they satisfy the condition imposed by the associated enzymes, share variables in their production functions, then they will all use the current values of those variables), and *det* to indicate that the systems we use are deterministic.

We prove now that EN P systems are universal (for production polynomials of degree two) also for the deterministic case, when working in the *allP* mode. Such universality results are also interesting from the robot control points of views, as they guarantee that any computer program or robot behavior can be implemented using EN P systems.

Theorem 4 $NRE = NP_{254}(poly^2(253), enz, allP, det)$.

Before proceeding to the proof of Theorem 4, we give two auxiliary helpful results.

Lemma 1 *Let f be a polynomial of degree n with k variables. The maximum number of terms of f is $a_{k+1}(n)$, given by the recurrence formula:*

$$a_k(n) = \sum_{i=0}^{n} a_{k-1}(i), where\, a_1(n) = 1, n \in \mathbf{N}.$$

Proof The general form of polynomial f is:

$$f = \sum_{s_1+\cdots+s_k \leq n} \alpha_{s_1,\ldots,s_k} \cdot x_1^{s_1} \cdot \ldots \cdot x_k^{s_k}.$$

The first thing to note is that the problem can be simplified by considering the polynomial g with $k+1$ variables obtained from f by multiplying each term with a power of the $(k+1)$-th variable such that each term has degree n. In this case, the powers of the variables in each term of g must satisfy a linear equality, i.e., their sum must be n. Also, f can be recovered from g by noting that $f(x_1,\ldots,x_k) = g(x_1,\ldots,x_k,1)$. All that is left is to count the maximum number of the terms of g.

The second important observation is that the coefficients of f, and therefore g, are irrelevant to counting problem. They only have to be non-zero. Thus, we can choose $g = (x_1 + \cdots + x_{k+1})^n$. The recurrence formula follows immediately from the binomial formula applied on g:

$$(x_1 + \cdots + x_{k+1})^n = \sum_{i=0}^{n} \binom{n}{i} (x_1 + \cdots + x_k)^i \cdot x_{k+1}^{n-i}. \qquad \square$$

The second result used in the proof of Theorem 4 concerns the form of the multivariate polynomial f.

Lemma 2 *If f is a polynomial of degree 5 with 5 variables, then f can be put in the following form:*

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

where $m = 252$ and represents the maximum number of terms of f in the general form, β_i are polynomial specific coefficients, and $a_{j,i}$ are some constants.

Remark 1 The maximum number of terms of a polynomial f in the general form of degree 5 with 5 variables is 252, computed using Lemma 1. The general form of a multivariate polynomial is given in the proof of Lemma 1.

Proof The transformation of polynomial f from the general form into the one in the theorem is in fact a change of base in the vector space R^m/R, where R is the set of real numbers. The constants $a_{j,i}$ must satisfy the following condition, obtained by writing polynomial f in both forms and identifying the coefficients of each term:

$$\left| \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{1,1}^{s_1} \cdot \ldots \cdot a_{k,1}^{s_k} & \cdots & a_{1,m}^{s_1} \cdot \ldots \cdot a_{k,m}^{s_k} \\ \vdots & \ddots & \vdots \\ a_{k,1}^{n} & \cdots & a_{k,m}^{n} \end{pmatrix} \right| \neq 0.$$

For polynomials with 5 variables and degree 5 a matrix of constants $a_{j,i}$ was found using Matlab. Moreover, non-negative integer values have been found for the $a_{j,i}$ constants. $\square$

With the two lemmas above, we can now proceed to the proof of Theorem 4.

Proof As in the proof of Theorem 2 we will use the characterization of recursively enumerable sets by polynomials of degree 5 with 5 variables and integer coefficients. From Lemma 1 we know that polynomials of degree 5 with 5 variables have at most 252 terms. Also, every multivariate polynomial can be written as the sum of powers of linear combinations of the polynomials variables. The previous two lemmas can

be used to construct the following EN P system (it is also presented graphically in Figure 3):

$$
\begin{aligned}
\Pi &= (254, H, \mu, (Var_{Generate}, Pr_{Generate}, Var_{Generate}(0))\\
&\quad (Var_{Compute}, Pr_{Compute}, Var_{Compute}(0)),\\
&\quad (Var_{Pow5_1}, Pr_{Pow5_1}, Var_{Pow5_1}(0)), \ldots,\\
&\quad (Var_{Pow5_{252}}, Pr_{Pow5_{252}}, Var_{Pow5_{252}}(0)), enum),\\
H &= \{Generate, Compute, Pow5_1, \ldots, Pow5_{252}\},\\
\mu &= [_{Generate}[_{Compute}[_{Pow5_1}\ ]_{Pow5_1} \cdots [_{Pow5_{252}}\ ]_{Pow5_{252}}]_{Compute}]_{Generate},\\
Var_{Generate} &= \{x_i, e_j, ez_k, er_i, n, e_t, g, gc \ :\ 1 \le i \le 5, 1 \le j \le 7, 1 \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 \ :\ 2 \le i \le 5\}\\
&\cup \{2 + e_t|_{er_5} \to \sum_{i=1}^{5} 1|x_i + 1|n,\ er_5|_{e_6} \to 1|gc,\\
&\quad e_6 \to 1|e_7,\ 2 \cdot e_7 \to 1|e_1 + 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_k, g^*, f_Q, enum \ :\ 1 \le i \le 5, 1 \le j \le 2, 1 \le k \le 252\},\\
Pr_{Compute} &= \{g^* + 2 \cdot \left(\sum_{i=1}^{5} a_{i,k} \cdot x_i^c + a_{6,k} \right) |_{e_1^c} \to 1|s_{1,k} + 1|s_{2,k} \ :\ 1 \le k \le 252\}\\
&\cup \{252 \cdot e_1^c \to \sum_{k=1}^{252} 1|ep_{1,k},\ g^* - \sum_{k=1}^{252} \beta_k \cdot t_k|_{e_2^c} \to 1|f_Q,\\
&\quad -f_Q|_{g^*} \to 1|enum,\ enum + f_Q + e_2^c \to 1|gc\}\\
Var_{Compute}(0) &= (0,0,\ldots,0),\\
Var_{Pow5_k} &= \{s_{1,k}, s_{2,k}, z, ep_{i,k} \ :\ 1 \le i \le 3\},\\
Pr_{Pow5_k} &= \{z + s_{1,k} \cdot s_{1,k}|_{ep_{1,k}} \to 1|s_{1,k},\ ep_{1,k} \to 1|ep_{2,k},\\
&\quad z + s_{1,k} \cdot s_{1,k}|_{ep_{2,k}} \to 1|s_{1,k},\ ep_{2,k} \to 1|ep_{3,k},\\
&\quad z + s_{1,k} \cdot s_{2,k}|_{ep_{3,k}} \to 1|t_k,\ ep_{3,k} \to 1|e_2^c\},\\
Var_{Pow5_k}(0) &= (0,0,\ldots,0),\ \text{for all } k = 1,2,\ldots,252.
\end{aligned}
$$

The system has two parts: membrane *Generate* generates all 5-tuples of natural numbers, in a deterministic way, and membrane *Compute* computes the value of polynomial f_Q for all generated 5-tuples. The generation of the 5-tuples is done by simulating a 5 position $(x_1, \ldots, x_5)$ countdown timer with automatic reset. However, instead of resetting to the same initial state, the countdown timer resets all positions to the next natural number. The initial state of the timer is $(5,5,5,5,5)$ and when the

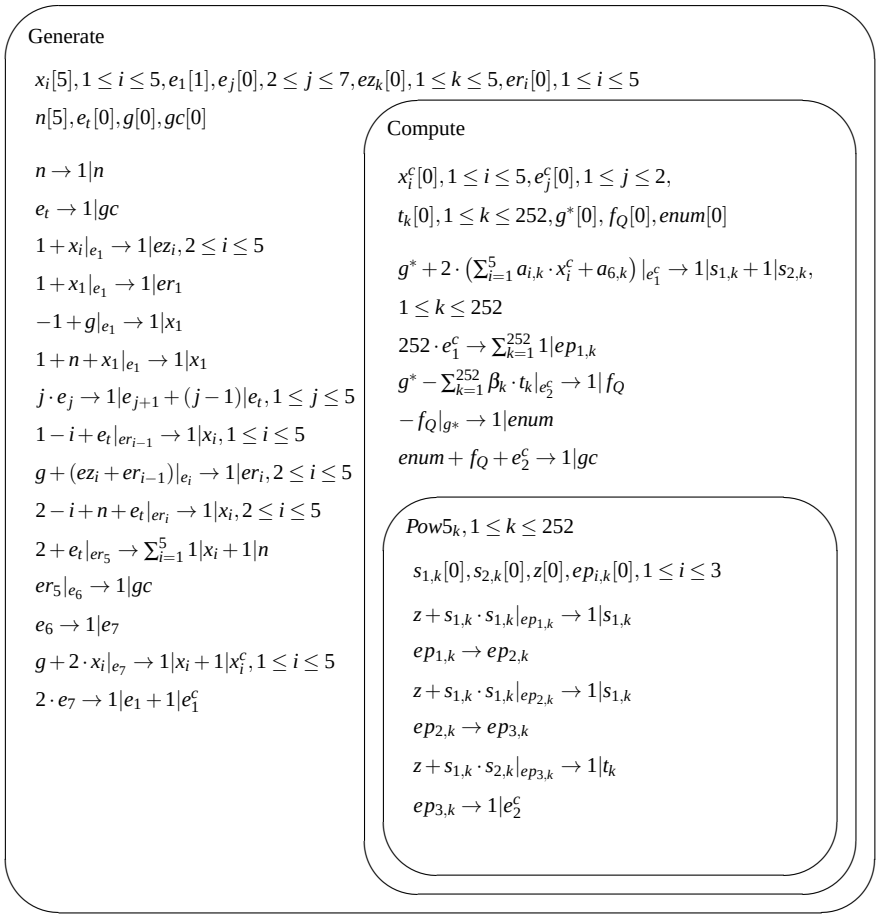

Fig. 3 The EN P system from the proof of Theorem 4

timer becomes zero, it resets to $(6,6,6,6,6)$. In this way all 5-tuples of natural numbers are generated. This functionality is achieved by first testing each position if it is zero, then if it needs to be reset, and finally it is updated, decreased by one or reset. This process is sequential, updating the position in order from x_1 to x_5. A 5-tuple is generated every 7 time steps and transferred to the *Compute* membrane.

Specifically, the ordering of 5-tuples of natural numbers is the following: the initial tuple is $(5,5,5,5,5)$, the next generated tuples are $(5,5,5,5,4)$, $\ldots$, $(5,5,5,5,0)$, $(5,5,5,4,5)$, $\ldots$, $(5,5,5,0,0)$, $(5,5,4,5,5)$, $\ldots$ and so on until $(0,0,0,0,0)$ is generated; the next tuple will be $(6,6,6,6,6)$, and then $(6,6,6,6,5)$, $(6,6,6,6,4)$, $\ldots$, $(6,6,6,6,0)$, $(6,6,6,5,6)$, $\ldots$ are generated; when the generated tuple becomes again $(0,0,0,0,0)$, the next tuple will be $(7,7,7,7,7)$. This process continues forever and generates all 5-tuples of natural numbers in a deterministic way. The initial tuple was chosen $(5,5,5,5,5)$ in order to activate the correct rules in the first computational step of the P system (see Figure 3).

The same tuple of numbers is generated arbitrarily many times, but this entails no problem (except the non-efficiency, but this is not of concern here): the result of the computation is the set of all generated numbers, so the repeats do not matter.

The values of f_Q are enumerated in the variable *enum*.

We will briefly show that all programs in all membranes satisfy the divisibility property which means that the unitary portion of the production value at each execution step is an integer. First of all, the production functions of all rules are polynomials with integer coefficients. Therefore, if the variables in the membrane system are integers, then the production values are also integers. For the programs which only have one variable in the repartition protocol the divisibility property is trivially satisfied. Next, we show that all the other programs satisfy the property *div*.

- *Generate*
  - programs $j \cdot e_j \rightarrow 1|e_{j+1} + (j-1)|e_t \ : \ 1 \leq j \leq 5$ — in this case the unitary portion is an integer and equal to e_j, because the sum of the repartition coefficients is j and the production value is $j \cdot e_j$.
  - programs $g + 2 \cdot x_i|_{e_7} \rightarrow 1|x_i + 1|x_i^c \ : \ 1 \leq i \leq 5$ — in these programs the production value will always be even because the variable g is always 0 (has initial value 0 and it is never changed). The sum of the repartition coefficients is 2, therefore the unitary portion is an integer.
  - program $2 \cdot e_7 \rightarrow 1|e_1 + 1|e_1^c$ — this case is similar to the one above, the production value is always an even number and the sum of the repartition coefficients is 2.
  - program $2 + e_t|_{er_5} \rightarrow \sum_{i=1}^{5} 1|x_i + 1|n$ — in this case, we have to take into account when the program is applied. Because the system is deterministic, we know exactly the execution sequence of the programs. In order for the unitary portion to be an integer we will show that the variable e_t has the value 4 when this program is applied. With this value the production value is 6 which is divisible by the sum of the repartition coefficients which is also 6. The program is executed when the enzyme er_5 gets the positive value 5. The variable er_i is produced by the program $g + (ez_i + er_{i-1})|_{e_i} \rightarrow 1|er_i$, for $i = 5$. However, this program is applied when enzyme e_5 is 1. In parallel, the program $j \cdot e_j \rightarrow 1|e_{j+1} + (j-1)|e_t$, for $j = 5$, is also applied and distributes the value 4 to variable e_t. The value of the variable e_t is consumed by other programs in that execution step. Therefore, e_t is 4 when the program $2 + e_t|_{er_5} \rightarrow \sum_{i=1}^{5} 1|x_i + 1|n$ is applied.
- *Compute*
  - programs $g^* + 2 \cdot \left(\sum_{i=1}^{5} a_{i,k} \cdot x_i^c + a_{6,k}\right)|_{e_1^c} \rightarrow 1|s_{1,k} + 1|s_{2,k} \ : \ 1 \leq k \leq 252$ — in this case the production value is always even because g^* is always 0 (it is initially 0 and it is not modified). The sum of the repartition coefficients is 2, therefore the unitary portion is an integer.

The P system was also simulated and tested using the *SimP* simulator, described in [11], and the simulation confirmed its correct functioning (in particular, the fact that it has the "divisibility property", that is, it is of *div* type).

The computation of the value of the polynomial f_Q for the current 5-tuple is done using only production functions of degree at most 2. This was achieved by

first considering f_Q in the form from Lemma 2. First, the linear combinations are computed and transferred to the $Pow5_k$ membranes. Then, the fifth power of the sums $a_{1,i}x_1 + \cdots + a_{5,i}x_5 + a_{6,i}$ is computed in $Pow5_k$ and the result is stored in the variables t_k. In the following step, the membrane *Compute* computes the value of f_Q as a linear combination of the terms t_k. It can be observed that polynomials of degree 2 are only used in $Pow5_k$ membranes. At the end of the computation, the values are filtered and only positive values are transferred to the output variable *enum*. □

We are currently working on proving that the computational power of cell-like EN P systems with the *allP* parallel execution mechanism can be further improved to the optimal degree of the polynomial production functions (degree one). This would be the best universality result from this point of view that can be obtained.

6 Final Remarks

Numerical P systems were only scarcely investigated from a computational point of view, but their usefulness in devising controllers for robots both suggests new variants (deterministic, with enzymatic control, working in the computing mode) and motivate further theoretical research.

These systems also raise interesting technical problems, mainly related to the fact that we do not use the halting condition in defining the result of a computation; the computations can go forever, and the values assumed by a specified variable during the computation form the generated set of numbers.

The present paper introduces a large numbers of classes of numerical P systems: deterministic – non-deterministic; enzymatic – non-enzymatic; generative, accepting, computing; sequential, all-parallel, one-parallel; with a "terminal" set of numbers or without such a squeezing mechanism. Only a few of the many cases obtained by combining these properties are investigated here.

In particular, we prove that enzymes improve the universality results in terms of the complexity of used polynomials, provided that the evolution programs are used in a parallel manner (different types of parallelism were used in the non-deterministic and the deterministic case).

Similar extensions of "general" notions in membrane computing to numerical P systems remain to be examined, and this is a rich research topic. For instance, other ways of using the programs can be considered: minimally parallel, with bounded parallelism, asynchronously. Then, we can also consider rules for handling membranes, such as membrane division and membrane creation. These operations are the basic tools by which polynomial solutions to computationally hard problems, typically, **NP**-complete problems, are obtained in the framework of P systems with symbol objects, by a time-space trade-off. Is this possible also for numerical P systems? In the previous investigations of numerical P systems, the computations were not halting, as usual in membrane computing; does the halting condition makes any difference? (Note that in the enzymatic case the halting is possible: no evolution program can be applied, because its associated enzyme forbids it.) What about numerical P systems used in the accepting mode?

A current research issue in membrane computing is to find classes of P systems which are not universal. This extends also to numerical P systems.

Of course, an important research topic is to further explore the use of numerical P systems in controlling robots or in other applications where functions from vectors of numbers to vectors of numbers should be computed in an efficient way.

And so on and so forth, a wealth of research ideas, which supports our belief that numerical P systems deserve further research efforts.

Acknowledgements The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

Useful remarks by two anonymous referees are gratefully acknowledged.

References

1. Arsene, O., Buiu, C., Popescu, N.: SNUPS – A simulator for numerical membrane computing. Intern. J. of Innovative Computing, Information and Control **7** (6), 3509–3522 (2011)
2. Buiu, C.: Towards integrated biologically inspired cognitive architectures, keynote talk. In: Proc. of the Int. Conf. on Electronics, Computers, and AI - ECAI' 09, Pitesti, Romania, **I**, pp. 2–9 (2009)
3. Buiu, C., Vasile, C.I., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences **187**, 33–51 (2012)
4. Korec, I.: Small universal register machines. Theoretical Computer Sci. **168**, 267–301 (1996)
5. Matijasevitch, Y. (ed.): Hilbert's Tenth Problem. MIT Press, Cambridge, London (1993)
6. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
7. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61** 108–143 (2000)
8. Păun, Gh.: Membrane Computing - An Introduction. Springer-Verlag, Berlin (2002)
9. Păun, Gh., Păun, R.A.: Membrane computing and economics: Numerical P systems. Fundamenta Informaticae **73** 213–227 (2006)
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
11. Pavel, A.B.: Membrane controllers for cognitive robots. Master's thesis, Department of Automatic Control and System Engineering, Politehnica University of Bucharest, Romania (2011)
12. Pavel, A.B., Arsene, O., Buiu, C.: 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, pp. 1331–1336 (2010)
13. Pavel, A.B., Buiu, C.: Using enzymatic numerical P systems for modeling mobile robot controllers. Natural Computing (in press). Doi: 10.1007/s11047-011-9286-5
14. Pavel, A.B., Vasile, C.I., Dumitrache, I.: Robot controllers implemented with enzymatic numerical P systems. Proceedings of Living Machines 2012, Lecture Notes in Artificial Intelligence **7375**, pp. 204–215, Springer-Verlag Berlin Heidelberg (2012)
15. Rozenberg, G., Salomaa, A.: Cornerstones of Undecidability. Prentice Hall. New York (1994)
16. The P Systems Web Page: http://ppage.psystems.eu (2012)