# Improving Universality Results on Parallel Enzymatic Numerical P Systems

Alberto Leporati, Antonio E. Porreca, Claudio Zandron, Giancarlo Mauri

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

**Summary.** We improve previously known universality results on enzymatic numerical P systems (EN P systems, for short) working in all-parallel and one-parallel modes. By using a flattening technique, we first show that any EN P system working in one of these modes can be simulated by an equivalent one-membrane EN P system working in the same mode. Then we show that linear production functions, each depending upon at most one variable, suffice to reach universality for both computing modes. As a byproduct, we propose some small deterministic universal enzymatic numerical P systems.

## 1 Introduction

Numerical P systems have been introduced in [10] as a model of membrane systems inspired both from the structure of living cells and from economics. Each region of a numerical P system contains some numerical variables, that evolve from initial values by means of *programs*. Each program consists of a *production function* and a *repartition protocol*; the production function computes an output value from the values of some variables occuring in the same region in which the function is located, while the repartition protocol distributes this output value among the variables in the same region as well as in the neighbouring (parent and children) ones.

In [10], and also in Chapter 23.6 of [11], some results concerning the computational power of numerical P systems are reported. In particular, it is proved that nondeterministic numerical P systems with polynomial production functions characterize the recursively enumerable sets of natural numbers, while deterministic numerical P systems, with polynomial production functions having non-negative coefficients, compute strictly more than semilinear sets of natural numbers.

Enzymatic Numerical P systems (EN P systems, for short) have been introduced in [13] as an extension of numerical P systems in which some variables, named the *enzymes*, control the application of the rules, similarly to what happens in P systems with promoters and inhibitors [1]. Although in [10] it is claimed

that numerical P systems have been inspired by economic and business processes, the most promising application of their enzymatic version seems to be the simulation of control mechanisms of mobile and autonomous robots [12, 2, 14, 15].

In [17, 16] some results concerning the computational power of enzymatic P systems are reported. In particular, in [17] it is shown that EN P systems with 7 membranes and polynomial production functions of degree 5 involving at most 5 variables, working in the *sequential* mode (at each step, only one of the active programs is applied in each membrane) are universal. The computational power of EN P systems working in the so called *one-parallel* mode — programs are applied in parallel in each membrane, but each variable can appear only in *one* of the production functions — is also investigated, showing universality of these systems with an unlimited number of membranes and *linear* production functions (that is, polynomial functions of degree 1), each involving at most 2 variables. Finally, the universality of (deterministic) EN P systems working in the *all-parallel mode* — in each membrane all programs which can be applied are applied, possibly using the same variable in many production functions — having 254 membranes and polynomial production functions of degree 2 involving at most 253 variables, is established. A considerable improvement of the last result has subsequently been presented in [16], where it is proved that 4 membranes and linear production functions involving at most 6 variables suffice to obtain universal deterministic EN P systems working in the all-parallel mode.

In this paper we continue the study of the computational power of enzymatic numerical P systems. In particular we first show that, given any EN P system $\Pi$ working either in the one-parallel or in the all-parallel mode, it is possible to build an equivalent EN P system $\Pi'$ whose structure consists of a single membrane. This *flattening* technique already improves some of the above mentioned results, reducing to 1 the number of membranes required by all-parallel or one-parallel EN P systems to reach universality — albeit, despite this transformation, one-parallel EN P systems still require an *unbounded number of variables*. Then, we prove that for EN P systems working either in the all-parallel or in the one-parallel mode one membrane and linear production functions — each involving at most 1 variable — suffice to reach universality. These results are all obtained by simulating deterministic and/or nondeterministic register machines; by considering a small deterministic universal register machine described in [6], we obtain as byproducts some small deterministic universal EN P systems, working in the all-parallel mode.

A point to be considered is that the output of our EN P systems is defined as the value of some specified variables in a *final configuration*, that is, a configuration which is not changed by further applying programs. This allows us to simplify some of our constructions, but it is a bit different from the way EN P systems produce their output in most existing papers, where some specified output variables are considered, and the output of the system is the set of all values assumed by these variables during the entire computation. However, we prove that each of our EN P systems can be easily modified in order to produce its output according to the latter mode.

The rest of the paper is organized as follows. In section 2 we recall the definitions of EN P systems and register machines, along with the terms, tools and notation that will be used in the following. In section 3 we first show that any EN P system working either in the all-parallel or in the one-parallel mode can be "flattened" to one membrane, and then we prove our universality results on one-membrane EN P systems working in all-parallel or in one-parallel modes. In section 4 we show that the EN P systems used to obtain these results can be modified in order to produce their output into separate variables, as it is usually done in the literature. The conclusions and some directions for further work are given in section 5.

## 2 Definitions and Mathematical Preliminaries

We denote by $\mathbb{N}$ the set of non-negative integers. An *alphabet* $A$ is a finite non-empty set of abstract *symbols*. Given $A$, the free monoid generated by $A$ under the operation of concatenation is denoted by $A^*$; the *empty string* is denoted by $\lambda$, and $A^* - \{\lambda\}$ is denoted by $A^+$. By $|w|$ we denote the length of the word $w$ over $A$. If $A = \{a_1, \ldots, a_n\}$, then the number of occurrences of symbol $a_i$ in $w$ is denoted by $|w|_{a_i}$; the *Parikh vector* associated with $w$ with respect to $a_1, \ldots, a_n$ is $(|w|_{a_1}, \ldots, |w|_{a_n})$. The *Parikh image* of a language $L$ over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in $L$. For a family of languages **FL**, the family of Parikh images of languages in **FL** is denoted by **PsFL**. The family of recursively enumerable languages is denoted by **RE**; the family of all recursively enumerable sets of $k$-dimensional vectors of non-negative integers can thus be denoted by **Ps**$(k)$**RE**. Since numbers can be seen as one-dimensional vectors, we can replace **Ps**$(1)$ by $\mathbb{N}$ in the notation, thus obtaining $\mathbb{N}$**RE**.

### 2.1 Enzymatic Numerical P Systems

An *enzymatic numerical P system* (EN P system, for short) is a construct of the form:

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0)))$$

where $m \geq 1$ is the degree of the system (the number of membranes), $H$ is an alphabet of labels, $\mu$ is a tree-like membrane structure with $m$ membranes injectively labeled with elements of $H$, $Var_i$ and $Pr_i$ are respectively the set of variables and the set of programs that reside in region $i$, and $Var_i(0)$ is the vector of initial values for the variables of $Var_i$. All sets $Var_i$ and $Pr_i$ are finite. In the original definition of EN P systems [13] the values assumed by the variables may be real, rational or integer numbers; in what follows we will allow instead only integer numbers. The variables from $Var_i$ are written in the form $x_{j,i}$, for $j$ running from 1 to $|Var_i|$, the cardinality of $Var_i$; the value assumed by $x_{j,i}$ at time $t \in \mathbb{N}$ is

denoted by $x_{j,i}(t)$. Similarly, the programs from $Pr_i$ are written in the form $P_{l,i}$, for $l$ running from 1 to $|Pr_i|$.

The *programs* allow the system to evolve the values of variables during computations. Each program is composed of two parts: a *production function* and a *repartition protocol*. The former can be any function using variables from the region that contains the program. Usually only polynomial functions are considered, since these are sufficient to reach the computational power of Turing machines, as proved in [17]. Using the production function, the system computes a *production value*, from the values of its variables at that time. This value is distributed to variables from the region where the program resides, and to variables in its upper (parent) and lower (children) compartments, as specified by the repartition protocol. Formally, for a given region $i$, let $v_1, \ldots, v_{n_i}$ be all these variables; let $x_{1,i}, \ldots, x_{k_i,i}$ be some variables from $Var_i$, let $F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})$ be the production function of a given program $P_{l,i} \in Pr_i$, and let $c_{l,1}, \ldots, c_{l,n_i}$ be natural numbers. The program $P_{l,i}$ is written in the following form:

$$F_{l,i}(x_{1,i}, \ldots, x_{k_i,i}) \to c_{l,1}|v_1 + c_{l,2}|v_2 + \cdots + c_{l,n_i}|v_{n_i} \tag{1}$$

where the arrow separates the production function from the repartition protocol. Let $C_{l,i} = \sum_{s=1}^{n_i} c_{l,s}$ be the sum of all the coefficients that occur in the repartition protocol. If the system applies program $P_{l,i}$ at time $t \geq 0$, it computes the value

$$q = \frac{F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))}{C_{l,i}}$$

that represents the "unitary portion" to be distributed to variables $v_1, \ldots, v_{n_i}$ proportionally with coefficients $c_{l,1}, \ldots, c_{l,n_i}$. So each of the variables $v_s$, for $1 \leq s \leq n_i$, will receive the amount $q \cdot c_{l,s}$. An important observation is that variables $x_{1,i}, \ldots, x_{k_i,i}$ involved in the production function are reset to zero after computing the production value, while the other variables from $Var_i$ retain their value. 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. As pointed out in [17], a delicate problem concerns the issue whether the production value is divisible by the total sum of coefficients $C_{l,i}$. As it is done in [17], in this paper we assume that this is the case, and we deal only with such systems; see [10] for other possible approaches.

Besides programs (1), EN P systems may also have programs of the form

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

where $e_{j,i}$ is a variable from $Var_i$ different from $x_{1,i}, \ldots, x_{k_i,i}$ and from $v_1, \ldots, v_{n_i}$. Such a program can be applied at time $t$ only if $e_{j,i}(t) > \min(x_{1,i}(t), \ldots, x_{k_i,i}(t))$. Stated otherwise, variable $e_{j,i}$ operates like an *enzyme*, that enables the execution of the program, but — like it happens also with catalysts — it is neither consumed nor modified by the execution of the program. However, in EN P systems enzymes can evolve by means of other programs, that is, enzymes can receive "contributions" from other programs and regions.

A *configuration* of $\Pi$ at time $t \in \mathbb{N}$ is given by the values of all the variables of $\Pi$ at that time; in a compact notation, we can write it as the sequence $(Var_1(t), \ldots, Var_m(t))$, where $m$ is the degree of $\Pi$. The *initial configuration* can thus be described as the sequence $(Var_1(0), \ldots, Var_m(0))$. The system $\Pi$ evolves from an initial configuration to other configurations by means of *computation steps*, in which one or more programs of $\Pi$ (depending upon the *mode* of computation) are executed. In [17], at each computation step the programs to be executed are chosen in the so called *sequential* mode: one program is nondeterministically chosen in each region, among the programs that can be executed at that time. Another possibility is to select the programs in the so called *all-parallel* mode: in each region, all the programs that can be executed are selected, with each variable participating in all programs where it appears. Note that in this case EN P systems become *deterministic*, since nondeterministic choices between programs never occur. A variant of parallelism, analogous to the maximal one which is often used in membrane computing, is the so called *one-parallel* mode: in each region, all the programs which can be executed can be selected, but the actual selection is made in such a way that each variable participates in only one of the chosen programs. We say that the system reaches a *final configuration* if and when it happens that no applicable set of programs produces a change in the current configuration. In such a case, a specified set of variables contains the output of the computation. Of course, a computation may never reach a final configuration. Note that in the usual definition of EN P systems the output of a computation is instead defined as the collection of values taken by a specified set of variables during the whole computation. In what follows we prove our results both by considering outputs in the final configurations, and by the latter notion of producing the output.

EN P systems can be used to compute functions, in the so called *computing mode*, by considering some *input variables* and *output variables*. The initial values of the input variables are considered the actual arguments of the function, while the value of the output variables in the final configuration (provided that the system reaches it) is viewed as the output of the computed function. If the system never reaches a final configuration, then the computed function is undefined for the specified input values. By neglecting input variables, (nondeterministic) EN P systems can also be used in the *generating mode*, whereas by neglecting output variables we can use (deterministic or nondeterministic) EN P systems in the *accepting mode*, where the input is accepted if the system reaches a final configuration.

A technical detail to take care of is the fact that normally we would like to characterize families of sets of *natural numbers* (sometimes including and sometimes excluding zero), while the input and output variables of EN P systems may also assume negative values. The systems we will propose are designed to produce only non-negative numbers in the output variables when the input variables (if present) are assigned with non-negative numbers. So if the systems are used in the intended way, they always produce meaningful (and correct) results. Another possibility, mentioned in [17] but not considered here, is to filter the output values so that only the positive ones are considered as output.

When using EN P systems in the generating or accepting modes, we denote by $\mathbf{ENP}_m(poly^n(r), app\_mode)$ the family of sets of (possibly vectors of) non-negative integer numbers which are computed by EN P systems of degree $m \geq 1$, using polynomials of degree at most $n \geq 0$ with at most $r \geq 0$ arguments as production functions; the fact that the programs are applied in the sequential, one-parallel or all-parallel mode is denoted by assigning the value *seq*, *oneP* or *allP* to the *app_mode* parameter, respectively. When $app\_mode \in \{seq, oneP\}$ and the P system is deterministic, we write *det* after the *app_mode* parameter; this specification is not needed for all-parallel EN P systems, since they are always deterministic. If one of the parameters $m$, $n$, $r$ is not bounded by a constant value, we replace it by $*$.

With this notation, we can summarize the characterizations of $\mathbb{N}\mathbf{RE}$ proved in [17] as follows:

$$\begin{aligned}\mathbb{N}\mathbf{RE} &= \mathbf{ENP}_7(poly^5(5), seq) = \mathbf{ENP}_*(poly^1(2), oneP) \\ &= \mathbf{ENP}_{254}(poly^2(253), allP)\end{aligned}$$

whereas the improvement of the last equality given in [16] can be written as $\mathbb{N}\mathbf{RE} = \mathbf{ENP}_4(poly^1(6), allP)$.

In section 3 we further improve the results concerning EN P systems working in the all-parallel and in the one-parallel modes: in both cases, we will obtain characterizations of $\mathbb{N}\mathbf{RE}$ by using just one membrane, and linear production functions that use each at most one variable.

### 2.2 Register Machines

In what follows we will simulate register machines, so we briefly recall their definition and some of their computational properties.

An *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 labelled 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:

- $j : (\text{INC}(r), k, l)$, with $0 \leq j < m$, $0 \leq k, l \leq m$ and $1 \leq r \leq n$.
  This instruction, labelled with $j$, increments the value contained in register $r$, then nondeterministically jumps either to instruction $k$ or to instruction $l$.
- $j : (\text{DEC}(r), k, l)$, with $0 \leq j < m$, $0 \leq k, l \leq m$ and $1 \leq r \leq n$.
  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).

A *deterministic* $n$-register machine is an $n$-register machine in which all INC instructions have the form $j : (\text{INC}(r), k, k)$; in what follows, we will write these instructions simply as $j : (\text{INC}(r), k)$.

A *configuration* of an $n$-register machine $M$ is described by the contents of each of its registers and by the program counter, that indicates the next instruction to be executed. Computations start by executing the first instruction of $P$ (labelled with 0), and possibly terminate when the instruction currently executed jumps to label $m$ (we may equivalently assume that $P$ includes the instruction $m$ : HALT, explicitly stating that the computation must halt).

It is well known that register machines provide a simple universal computational model, and that machines with three registers suffice to characterize $\mathbb{N}$**RE** [8]. More precisely, we can use register machines in the computing, generating or accepting mode, obtaining the following results [3, 4, 5]. For the computing mode, we have:

**Proposition 1.** *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ $(\alpha, \beta > 0)$, there exists a deterministic register machine $M$ with $(\max\{\alpha, \beta\} + 2)$ registers computing $f$ in such a way that, when starting with $n_1$ to $n_\alpha$ in registers 1 to $\alpha$, $M$ has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label $m$ with registers 1 to $\beta$ containing $r_1$ to $r_\beta$, and all other registers being empty; if $f(n_1, \ldots, n_\alpha)$ is undefined then the final label of $M$ is never reached.*

In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts:

**Proposition 2.** *For any recursively enumerable set $L \subseteq$ **Ps**$(\alpha)$**RE** of vectors of non-negative integers there exists a deterministic register machine $M$ with $(\alpha + 2)$ registers accepting $L$ in such a way that, when starting with $n_1$ to $n_\alpha$ in registers 1 to $\alpha$, $M$ has accepted $(n_1, \ldots, n_\alpha) \in L$ if and only if it halts in the final label $m$ with all registers being empty.*

To generate vectors of non-negative integers, we need nondeterministic register machines:

**Proposition 3.** *For any recursively enumerable set $L \subseteq$ **Ps**$(\beta)$**RE** of vectors of non-negative integers there exists a non-deterministic register machine $M$ with $(\beta + 2)$ registers generating $L$, i.e., when starting with all registers being empty, $M$ generates $(r_1, \ldots, r_\beta) \in L$ if it halts in the final label $m$ with registers 1 to $\beta$ containing $r_1$ to $r_\beta$, and all other registers being empty.*

## 3 Universality of EN P Systems

As stated above, our aim is to improve the universality results shown in [17, 16], concerning all-parallel and one-parallel EN P systems. We first prove that these P systems can be "flattened".

**Theorem 1.** *Let $\Pi$ be any computing (or generating, or accepting) EN P system of degree $m \geq 1$, working in the all-parallel or in the one-parallel mode. Then there exists an EN P system $\Pi'$ of degree 1 that computes (resp., generates, accepts) the same function (resp., family of sets) using the same rule application mode.*

*Proof.* Let $\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0)))$ be an EN P system, computing a function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ $(\alpha, \beta \geq 0)$ and working in the all-parallel mode. All the other cases (one-parallel, generating and accepting modes) can be simply deduced from the following argumentation.

Note that each variable $x_{j,i} \in Var_i$ and each program $P_{l,i} \in Pr_i$ already indicates in one of its indexes the region that contains it. We build a new EN P system $\Pi'$ of degree 1, by putting all the variables and all the programs of $\Pi$ — keeping both indexes, also in the variables occurring in programs — in the membrane of $\Pi'$. Clearly, this establishes a bijection between the variables (resp., programs) of $\Pi$ and the corresponding variables (resp., programs) of $\Pi'$, since the presence of both indexes in $\Pi'$ allows one to keep track of the region of $\Pi$ from which each variable and each program comes from. So any program $P_{l,i}$ of $\Pi$ still operates on the correct variables when transformed and put into $\Pi'$, regardless of whether or not it uses an enzyme. Also input and output variables are preserved, and so the only issue is related with the mode used to select the programs to be applied. If $\Pi$ works in the sequential mode, then at each computation step only (at most) one program is selected in each region; this means that globally $\Pi$ executes a set of programs which cannot be captured in $\Pi'$ by any of the sequential, one-parallel and all-parallel modes. Instead, if $\Pi$ works in the all-parallel mode then at each computation step all the programs that can be executed are selected, and the same happens in $\Pi'$ by letting it work in the all-parallel mode. The same applies when $\Pi$ and $\Pi'$ work in the one-parallel mode, and so the claim of the theorem follows. □

This result already allows to improve the universality results shown in [17, 16] for all-parallel and one-parallel EN P systems, obtaining the following characterizations of $\mathbb{N}\mathbf{RE}$:

$$\mathbb{N}\mathbf{RE} = \mathbf{ENP}_1(poly^1(6), allP) = \mathbf{ENP}_1(poly^1(2), oneP)$$

However — as stated in the Introduction — despite this simplification, one-parallel EN P systems still require an unbounded number of variables, since each "new" variable in $\Pi'$ is indexed with the region of $\Pi$ it comes from.

Anyhow, we can improve both results. We start with the first equality, concerning all-parallel EN P systems.

**Theorem 2.** *Each partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ $(\alpha > 0,\ \beta \geq 0)$ can be computed by a one-membrane EN P system working in the all-parallel mode, having linear production functions that use each at most one variable.*

*Proof.* Since all-parallel EN P systems are deterministic, we prove the statement by simulating deterministic register machines. Let $M = (n, P, m)$ be such a machine with $n$ registers, computing $f$ by means of program $P$. The initial instruction of $P$ has the label 0 and the machine halts if and when the program counter assumes the value $m$. Observe that according to the result stated in Proposition 1, $n = \max\{\alpha, \beta\} + 2$ is enough. The input values $x_1, \ldots, x_\alpha$ are expected to be in the

first $\alpha$ registers before the computation starts, and the values of $f(x_1, \ldots, x_\alpha)$ — if any — are expected to be in registers 1 to $\beta$ at the end of a halting computation. Moreover, without loss of generality, we may assume that at the beginning of a computation all the registers except possibly the registers 1 to $\alpha$ contain zero.

We construct the EN P system $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$ of degree 1, where:

- $H = \{s\}$ is the label of the only membrane (the skin) of $\Pi_M$;
- $\mu = [\,]_s$ is the membrane structure;
- $Var_1 = \{r_1, \ldots, r_n\} \cup \{p_0, \ldots, p_m\}$;
- $Pr_1 = \{2p_j \to 1|r_i + 1|p_k$ for all instructions $j : (\text{INC}(i), k) \in P\} \cup \{-p_j \to 1|r_i,\ r_i + 2|_{p_j} \to 1|r_i + 1|p_l,\ p_j \to 1|p_k,\ r_i - 1|_{p_j} \to 1|p_k$ for all instructions $j : (\text{DEC}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values of the variables of $Var_1$, obtained by putting:
  - $r_i = x_i$ for all $1 \leq i \leq \alpha$;
  - $r_i = 0$ for all $\alpha + 1 \leq i \leq n$;
  - $p_0 = 1$;
  - $p_j = 0$ for all $1 \leq j \leq m$.

The value of register $i$, for $1 \leq i \leq m$, is contained in variable $r_i$. The input values $x_1, \ldots, x_\alpha$ are introduced into the P system as the initial values of variables $r_1, \ldots, r_\alpha$. Variables $p_0, \ldots, p_m$ are used to indicate the value of the program counter; at the beginning of each computation step, the variable corresponding to the value of the program counter of $M$ will assume value 1, while all the others will be equal to zero.

The simulation of $M$ by $\Pi_M$ works as follows. Each increment instruction $j : (\text{INC}(i), k)$ is simulated in one step by the execution of the program

$$2p_j \to 1|r_i + 1|p_k$$

This program is executed at *every* computation step of $\Pi_M$; however, when $p_j = 0$ it has no effect: $p_j$ is once again set to zero, and a contribution of zero is distributed among variables $r_i$ and $p_k$. All variables are thus unaffected in this case. When $p_j = 1$, the production value $2p_j = 2$ is distributed among $r_i$ and $p_k$, giving a contribution of 1 to each of them. Hence the value of $r_i$ is incremented, the value of $p_k$ passes from 0 to 1, while the value of $p_j$ is zeroed. All the other variables are unaffected, and the system is now ready to simulate the next instruction of $M$.

Each decrement instruction $j : (\text{DEC}(i), k, l)$ is simulated in one step by the parallel execution of the following programs:

$$-p_j \to 1|r_i \tag{2}$$

$$r_i + 2|_{p_j} \to 1|r_i + 1|p_l \tag{3}$$

$$p_j \to 1|p_k \tag{4}$$

$$r_i - 1|_{p_j} \to 1|p_k \tag{5}$$

If $p_j = 0$, programs (3) and (5) are not enabled (since by construction $r_i \geq 0$ and thus $p_j \leq r_i$), while programs (2) and (4) distribute a contribution of zero to $r_i$ and $p_k$; before doing so, variable $p_j$ is set to zero, thus leaving its value unchanged. Hence, the case in which $p_j = 0$ causes no problems to the overall simulation.

Now assume that $p_j = 1$ and $r_i > 0$. In this case, the value of $r_i$ should be decremented and the computation should continue with instruction $k$. Program (2) correctly decrements $r_i$, and program (4) passes the value of $p_j = 1$ to $p_k$, thus correctly pointing at the next istruction of $M$ to be simulated. The execution of both programs sets the value of $p_j$ to zero, which is also correct. Programs (3) and (5) have no effect since to be executed it should be $p_j > r_i$, that is, $r_i < 1$ (which means $r_i = 0$, since $r_i \geq 0$ by construction).

Now assume that $p_j = 1$ and $r_i = 0$. In this case, the value of $r_i$ should be kept equal to zero, and the computation should continue with instruction $l$. Program (2) sends a contribution of $-1$ to $r_i$, while program (4) sets — incorrectly — $p_k$ to 1; both programs set $p_j$ to zero. This time, however, programs (3) and (5) are also executed. Both set the value of $r_i$ to zero. After that, program (3) adds 1 to $r_i$, thus canceling the effect of program (2); as a result, the value assumed by $r_i$ after the execution of the two programs is zero. Program (3) also makes $p_l$ assume the value 1, thus correctly pointing to the next instruction of $M$ to be simulated. Finally, program (5) gives a contribution of $-1$ to $p_k$, canceling the effect of program (4); the resulting value of $p_k$ will thus be 0.

It follows from the description given above that after the simulation of each instruction of $M$ the value of every variable $r_i$ equals the contents of register $i$, for $1 \leq i \leq n$, while the only variable among $p_0, \ldots, p_m$ equal to 1 indicates the next instruction of $M$ to be simulated. When the program counter of $M$ reaches the value $m$, the corresponding variable $p_m$ assumes value 1. Since no program contains the variable $p_m$ either in the production function or among the enzymes that enable or disable the execution of the program, $\Pi_M$ reaches a final configuration; the result of the computation is contained in variables $r_1, \ldots, r_\beta$. □

By taking $\beta = 0$ in the previous proof, we get the following result concerning the accepting variant of EN P systems working in the all-parallel mode.

**Corollary 1.** *For any $L \in \mathbf{Ps}(\alpha)\mathbf{RE}$ there exists a one-membrane EN P system, having linear production functions each depending upon at most one variable, that accepts $L$ by working in the all-parallel mode.*

*Proof.* We consider a register machine $M$ with $(\alpha + 2)$ registers accepting $L$ according to Proposition 2, and we construct the one-membrane EN P system $\Pi_M$ that accepts $L$ following the construction given in the proof of Theorem 2. The input values $x_1, \ldots, x_\alpha$ expected to be in the first $\alpha$ registers in $M$ are assigned as initial values to variables $r_1$ to $r_\alpha$ in $\Pi_M$, whereas the initial values of variables $r_{\alpha+1}$ to $r_n$ are 0. The P system $\Pi_M$ accepts this input if and only if it reaches a final configuration. □

By putting $\alpha = 1$ in Corollary 1, we obtain the following characterization:

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

**Fig. 1.** The small universal deterministic register machine defined in [6]

$$\mathbb{N}\mathbf{RE} = \mathbf{ENP}_1(poly^1(1), allP)$$

A direct consequence of Theorem 2 is that there exists a *small* universal all-parallel EN P system that computes every possible partial recursive function.

**Theorem 3.** *There exists a universal all-parallel EN P system of degree* 1*, having* 31 *variables and* 61 *programs.*

*Proof.* We consider the small universal deterministic register machine $M_u$ described in [6], and illustrated in Figure 1. This machine has $n = 8$ registers and $m = 22$ instructions, and can be used to compute any unary partial recursive function $f : \mathbb{N} \to \mathbb{N}$ as follows. Let $(\varphi_0, \varphi_1, \ldots)$ be a fixed admissible enumeration of the unary partial recursive functions. Since $M_u$ is universal, there exists a recursive function $g$ such that for all natural numbers $y, z$ it holds $\varphi_y(z) = M_u(g(y), z)$. Hence, to compute $f(x)$ we first consider the index $y$ of $f$ in the above enumeration of unary recursive functions. Then we put $g(y)$ and $x$ in registers 2 and 3 of $M_u$, respectively, and we start the computation; the value of $f(x)$ will be found in register 1 if and when $M_u$ halts.

By following the arguments given in the proof of Theorem 2 we construct the all-parallel EN P system $\Pi_{M_u} = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$ of degree 1, where:

- $H = \{s\}$ is the label of the only membrane (the skin) of $\Pi$;
- $\mu = [\,]_s$ is the membrane structure;
- $Var_1 = \{r_1, \ldots, r_8\} \cup \{p_0, \ldots, p_{22}\}$;
- $Pr_1 = \{2p_j \to 1|r_i + 1|p_k$ for all instructions $j : (\text{INC}(i), k)$ listed in Figure 1$\} \cup \{-p_j \to 1|r_i,\ r_i + 2|_{p_j} \to 1|r_i + 1|p_l,\ p_j \to 1|p_k,\ r_i - 1|_{p_j} \to 1|p_k$ for all instructions $j : (\text{DEC}(i), k, l)$ listed in Figure 1$\}$;

- $Var_1(0)$ is the vector of initial values of the variables of $Var_1$, obtained by putting:
  - $r_2 = g(y)$, the "code" associated to function $f$;
  - $r_3 = x$, the input of $f$;
  - $r_1 = r_4 = r_5 = r_6 = r_7 = r_8 = 0$;
  - $p_0 = 1$;
  - $p_i = 0$ for all $1 \leq i \leq 22$.

This system simulates the operation of $M_u$, as described in the proof of Theorem 2. Hence, if and when the computation reaches a final configuration, variable $r_1$ contains the value of $f(x)$.

The number of increment and decrement instructions of $M_u$ are 9 and 13, respectively. Each increment instruction is translated to 1 program of $\Pi_{M_u}$ while each decrement instruction produces 4 programs, for a total of 61 programs. The variables are $n + m + 1 = 31$. □

We now turn to EN P systems working in the one-parallel mode. We start proving the following theorem.

**Theorem 4.** *Each partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ $(\alpha, \beta \geq 0)$ can be computed by a one-membrane EN P system working in the one-parallel mode, having linear production functions that use each at most two variables.*

*Proof.* We proceed like in the proof of Theorem 2, with the difference that here we simulate both deterministic and nondeterministic register machines. Let $M = (n, P, m)$ be a nondeterministic register machine with $n = \max\{\alpha, \beta\} + 2$ registers, that computes $f$ by means of program $P$. As usual, the input values $x_1, \ldots, x_\alpha$ are expected to be in the first $\alpha$ registers before the computation starts, all the other registers being empty. If and when the computation of $M$ halts, the values of $f(x_1, \ldots, x_\alpha)$ will be found in registers 1 to $\beta$.

We construct the one-membrane EN P system $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$, where:

- $H = \{s\}$ is the label of the only membrane (the skin) of $\Pi_M$;
- $\mu = [\,]_s$ is the membrane structure;
- $Var_1 = \{r_1, \ldots, r_n\} \cup \{p_0, \ldots, p_m\} \cup \{q_0, \ldots, q_m\} \cup \{z_{j,1}, z_{j,2}, z_{j,3}$ for all instructions $j : (\text{INC}(i), k, l) \in P\} \cup \{z_{j,1}, z_{j,2}, z_{j,3}, z_{j,4}, z_{j,5}$ for all instructions $j : (\text{DEC}(i), k, l) \in P\}$;
- $Pr_1 = \{z_{j,1}+3|_{p_j} \to 1|r_i+1|p_k+1|q_k,\ z_{j,1}+3|_{p_j} \to 1|r_i+1|p_l+1|q_l,\ z_{j,2}-1|_{p_j} \to 1|q_j,\ z_{j,3}-1|_{q_j} \to 1|p_j$ for all instructions $j : (\text{INC}(i), k, l) \in P\} \cup \{z_{j,1}-1|_{p_j} \to 1|r_i,\ r_i + 3|_{p_j} \to 1|r_i + 1|p_l + 1|q_l,\ z_{j,2} + 2p_j|_{r_i} \to 1|p_j + 1|p_k,\ z_{j,3} + 2q_j|_{r_i} \to 1|q_j + 1|q_k,\ z_{j,4} - 1|_{p_j} \to 1|q_j,\ z_{j,5} - 1|_{q_j} \to 1|p_j\}$ for all instructions $j : (\text{DEC}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values of the variables of $Var_1$, obtained by putting:
  - $r_i = x_i$ for all $1 \leq i \leq \alpha$;

- $r_i = 0$ for all $\alpha + 1 \leq i \leq n$;
- $p_0 = q_0 = 1$;
- $p_j = q_j = 0$ for all $1 \leq j \leq m$;
- $z_{j,1} = z_{j,2} = z_{j,3} = 0$ for all $0 \leq j < m$ such that $j : (\text{INC}(i), k, l) \in P$;
- $z_{j,1} = z_{j,2} = z_{j,3} = z_{j,4} = z_{j,5} = 0$ for all $0 \leq j < m$ such that $j : (\text{DEC}(i), k, l) \in P$.

Just like in the proof of Theorem 2, the value of register $i$, for $1 \leq i \leq n$, is contained in variable $r_i$, and the input values $x_1, \ldots, x_\alpha$ are introduced into the P system as the initial values of variables $r_1, \ldots, r_\alpha$. This time, however, the system uses both variables $p_0, \ldots, p_m$ and $q_0, \ldots, q_m$ to indicate the value of the program counter of $M$, so that when simulating the $j$-th instruction of $P$ variables $p_j$ and $q_j$ are both set to 1, while all the others are zero. This double representation of the program counter will allow us to set its value while also using it as an enzyme: precisely, variable $p_j$ will be used as an enzyme to update the value of $q_j$, and vice versa. The auxiliary variables $z_{j,1}, \ldots, z_{j,5}$, when defined, are used during the simulation of INC and DEC instructions, and are always set to zero.

The simulation of $M$ by $\Pi_M$ works as follows. Each increment instruction $j : (\text{INC}(i), k, l)$ is simulated in one step by the execution of the following programs:

$$z_{j,1} + 3|_{p_j} \rightarrow 1|r_i + 1|p_k + 1|q_k \tag{6}$$

$$z_{j,1} + 3|_{p_j} \rightarrow 1|r_i + 1|p_l + 1|q_l \tag{7}$$

$$z_{j,2} - 1|_{p_j} \rightarrow 1|q_j \tag{8}$$

$$z_{j,3} - 1|_{q_j} \rightarrow 1|p_j \tag{9}$$

These programs are not executed when $p_j = q_j = 0$, since variables $z_{j,1}$, $z_{j,2}$ and $z_{j,3}$ are zero, hence in this case they have no effect. When $p_j = q_j = 1$, instead, programs (8) and (9) as well as one among programs (6) and (7) are executed, since variable $z_{j,1}$ makes these latter programs compete in the one-parallel mode of application. Assume that program (6) wins the competition (a similar argument holds if (7) wins instead): its effect is incrementing $r_i$ and setting $p_k$ and $q_k$ to 1, thus correctly pointing to the next instruction of $M$ to be simulated. The effect of programs (8) and (9) is giving a contribution of $-1$ to both $p_j$ and $q_j$, whose final value will thus be zero. All the other variables are unaffected. If $M$ is deterministic, then the simulation of the instruction $j : (\text{INC}(i), k)$ is performed by using the same programs without (7). In this case no competition occurs between the programs, and so the simulation is deterministic.

Each decrement instruction $j : (\text{DEC}(i), k, l)$ is simulated in one step by the execution of the following programs:

$$z_{j,1} - 1|_{p_j} \to 1|r_i \tag{10}$$
$$r_i + 3|_{p_j} \to 1|r_i + 1|p_l + 1|q_l \tag{11}$$
$$z_{j,2} + 2p_j|_{r_i} \to 1|p_j + 1|p_k \tag{12}$$
$$z_{j,3} + 2q_j|_{r_i} \to 1|q_j + 1|q_k \tag{13}$$
$$z_{j,4} - 1|_{p_j} \to 1|q_j \tag{14}$$
$$z_{j,5} - 1|_{q_j} \to 1|p_j \tag{15}$$

If $p_j = q_j = 0$ then programs (10), (11), (14) and (15) are not enabled, while programs (12) and (13) are enabled only if $r_i > 0$. However, in this case they set to 0 variables $p_j$ and $q_j$ (thus leaving their value unaltered), and distribute a contribution of zero to $p_j$, $q_j$, $p_k$ and $q_k$, thus producing no effect. All the other variables are left unchanged, so no problems occur to the overall simulation.

Now assume that $p_j = q_j = 1$ and $r_i > 0$. In this case, the value of $r_i$ should be decremented and the computation should continue with instruction $k$. Program (10) correctly decrements $r_i$, whereas program (11) is not executed since $r_i \geq p_j$. Programs (12) and (13) set to 1 variables $p_k$ and $q_k$ (thus pointing at the next instruction of $M$ to be simulated), and send a contribution of 1 to variables $p_j$ and $q_j$, after setting their value to zero. On the other hand, programs (14) and (15) send a contribution of $-1$ to $p_j$ and $q_j$, so that their final value will be zero.

Now assume that $p_j = q_j = 1$ and $r_i = 0$. In this case, the value of $r_i$ should be kept equal to zero, and the computation should continue with instruction $l$. Program (10) sends a contribution of $-1$ to $r_i$. This time, however, program (11) is also executed; its effect is sending a contribution of 1 to $r_i$, after setting it to zero (so that its final value will be zero), and setting to 1 the value of variables $p_l$ and $q_l$. Programs (12) and (13) are inactive, and hence are not executed. Finally, programs (14) and (15) send a contribution of $-1$ to $p_j$ and $q_j$, so that their final value will be zero.

It follows from the description given above that after the simulation of each instruction of $M$ the value of every variable $r_i$ equals the contents of register $i$, for $1 \leq i \leq n$, while variables $p_0, \ldots, p_m$ and $q_0, \ldots, q_m$ correctly indicate the next instruction of $M$ to be simulated. When the program counter of $M$ reaches the value $m$, the corresponding variables $p_m$ and $q_m$ assume value 1. Since no program contains these variables either in the production function or among the enzymes, the simulation reaches a final configuration; the result of the computation is contained in variables $r_1, \ldots, r_\beta$. □

By taking $\beta = 0$ and $\alpha \geq 1$ in the previous proof, we obtain the following result concerning the accepting variant of EN P systems working in the one-parallel mode.

**Corollary 2.** *For any $L \in \mathbf{Ps}(\alpha)\mathbf{RE}$ there exists a one-membrane EN P system, having linear production functions each depending upon at most two variables, that accepts $L$ by working in the one-parallel mode.*

On the other hand, by taking $\alpha = 0$ and $\beta \geq 1$ we get the following characterization of $\mathbf{Ps}(\beta)\mathbf{RE}$ by the generating variant of EN P systems working in the one-parallel mode.

**Corollary 3.** *For any $L \in \mathbf{Ps}(\beta)\mathbf{RE}$ there exists a one-membrane (nondeterministic) EN P system, having linear production functions each depending upon at most two variables, that generates $L$ by working in the one-parallel mode.*

By putting $\alpha = 1$ and $\beta = 0$ in Corollary 2, and $\alpha = 0$ and $\beta = 1$ in Corollary 3, we obtain the following characterization:

$$\mathbb{N}\mathbf{RE} = \mathbf{ENP}_1(poly^1(2), oneP)$$

Another consequence of Theorem 4 is that there exists the small universal deterministic one-parallel EN P system mentioned in the following theorem.

**Theorem 5.** *There exists a universal one-parallel deterministic EN P system of degree* 1*, having* 146 *variables and* 105 *programs.*

*Proof.* The system mentioned in the statement simulates the small universal deterministic register machine $M_u$ reported in Figure 1, and is built according to the description given in the proof of Theorem 4, as we have done in the proof of Theorem 3. The number of increment and decrement instructions of $M_u$ are 9 and 13, respectively. Each increment and each decrement instruction is translated to 3 and 6 programs of the small universal EN P system, respectively, for a total of 105 programs. As for variables, 8 are used to simulate the registers of $M_u$, and 46 are used to denote the value of its program counter; moreover, there are 3 and 5 auxiliary variables for each increment and each decrement instruction, respectively, for a total of 146 variables. □

Let us note that, since the EN P system mentioned in the statement of Theorem 5 is deterministic, it also works in the all-parallel mode, albeit in this case the system described in Theorem 3 is smaller.

By looking at the operation of the EN P system described in the proof of Theorem 4, we can see that the only programs whose production functions depend upon two variables are programs (12) and (13). Further, if we remove variables $z_{j,2}$ and $z_{j,3}$ from these programs the simulation of register machine $M$ continues to work correctly, except in the case when $r_i = 1$ and $p_j = q_j = 1$. Hence if $r_i$ could only assume even values (so that the value $2v$ denotes the fact that the contents of the $i$-th register of $M$ is $v$) we could get rid of variables $z_{j,2}$ and $z_{j,3}$ in programs (12) and (13), thus obtaining a one-parallel EN P system whose linear production functions each depend on just one variable. This is exactly what we do in the next theorem, where $2\mathbb{N}$ denotes the set of even natural numbers.

**Theorem 6.** *Each partial recursive function $f : (2\mathbb{N})^\alpha \to (2\mathbb{N})^\beta$ $(\alpha, \beta \geq 0)$ can be computed by a one-membrane EN P system working in the one-parallel mode, having linear production functions that use each at most one variable.*

*Proof.* The proof is similar to the one given for Theorem 4. The one-parallel EN P system $\Pi_M$ that simulates the nondeterministic register machine $M = (n, P, m)$ is now defined as follows:

$$\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$$

where:

- $H = \{s\}$ is the label of the only membrane (the skin) of $\Pi_M$;
- $\mu = [\,]_s$ is the membrane structure;
- $Var_1 = \{r_1, \ldots, r_n\} \cup \{p_0, \ldots, p_m\} \cup \{q_0, \ldots, q_m\} \cup \{z_{j,1}, z_{j,2}, z_{j,3}$ for all $0 \leq j < m\}$;
- $Pr_1 = \{z_{j,1}+4|_{p_j} \to 2|r_i+1|p_k+1|q_k,\ z_{j,1}+4|_{p_j} \to 2|r_i+1|p_l+1|q_l,\ z_{j,2}-1|_{p_j} \to 1|q_j,\ z_{j,3}-1|_{q_j} \to 1|p_j$ for all instructions $j : (\text{INC}(i), k, l) \in P\} \cup \{z_{j,1}-2|_{p_j} \to 1|r_i,\ r_i+4|_{p_j} \to 2|r_i+1|p_l+1|q_l,\ 2p_j|_{r_i} \to 1|p_j+1|p_k,\ 2q_j|_{r_i} \to 1|q_j+1|q_k,\ z_{j,2}-1|_{p_j} \to 1|q_j,\ z_{j,3}-1|_{q_j} \to 1|p_j\}$ for all instructions $j : (\text{DEC}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values of the variables of $Var_1$, obtained by putting:
  - $r_i = 2x_i$ for all $1 \leq i \leq \alpha$;
  - $r_i = 0$ for all $\alpha + 1 \leq i \leq n$;
  - $p_0 = q_0 = 1$;
  - $p_j = q_j = 0$ for all $1 \leq j \leq m$;
  - $z_{j,1} = z_{j,2} = z_{j,3} = 0$ for all $0 \leq j < m$.

As stated above, now the value of $r_i$ is the double of the value of register $i$, for $1 \leq i \leq n$. So, in particular, the double of the input values $x_1, \ldots, x_\alpha$ are introduced into the P system as the initial values of variables $r_1, \ldots, r_\alpha$. Once again, like in the proof of Theorem 4, the system uses both variables $p_0, \ldots, p_m$ and $q_0, \ldots, q_m$ to indicate the value of the program counter of $M$, so that when simulating the $j$-th instruction of $P$ variables $p_j$ and $q_j$ are both equal to 1, while all the others are zero. The value of variables $z_{j,1}, z_{j,2}, z_{j,3}$ is always zero during the entire computation.

Each increment instruction $j : (\text{INC}(i), k, l)$ of $M$ is simulated in one step by the execution of the following programs:

$$z_{j,1} + 4|_{p_j} \to 2|r_i + 1|p_k + 1|q_k \tag{16}$$

$$z_{j,1} + 4|_{p_j} \to 2|r_i + 1|p_l + 1|q_l \tag{17}$$

$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{18}$$

$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{19}$$

The simulation is analogous to the one described in the proof of Theorem 4, with the difference that instead of incrementing $r_i$ the system now adds 2 to it; to do so, the production value computed by the first two programs must be 4 instead of 3. Nondeterminism is given by the fact that, when $p_j = q_j = 1$, variable $z_{j,1}$ makes programs (16) and (17) compete in the one-parallel mode. If the machine $M$ to be simulated is deterministic, then program (17) disappears, and so the simulation becomes deterministic.

Each decrement instruction $j : (\text{DEC}(i), k, l)$ is simulated in one step by the execution of the following programs:

$$z_{j,1} - 2|_{p_j} \rightarrow 1|r_i \tag{20}$$
$$r_i + 4|_{p_j} \rightarrow 2|r_i + 1|p_l + 1|q_l \tag{21}$$
$$2p_j|_{r_i} \rightarrow 1|p_j + 1|p_k \tag{22}$$
$$2q_j|_{r_i} \rightarrow 1|q_j + 1|q_k \tag{23}$$
$$z_{j,2} - 1|_{p_j} \rightarrow 1|q_j \tag{24}$$
$$z_{j,3} - 1|_{q_j} \rightarrow 1|p_j \tag{25}$$

The simulation is analogous to the one described in the proof of Theorem 4, with small differences.

The case when $p_j = q_j = 0$ operates just like in the proof of Theorem 4: programs (20), (21), (24) and (25) are not active, while programs (22) and (23) are executed only if $r_i > 0$; however, in such a case, a contribution of 0 is distributed to variables $p_j$, $q_j$, $p_k$, $q_k$ after setting $p_j$ and $q_j$ to zero.

Now assume that $p_j = q_j = 1$ and $r_i > 0$. Program (20) correctly decrements $r_i$ (subtracting 2 from its value), whereas program (21) is not executed since $r_i > p_j$. Programs (22) and (23) set to 1 variables $p_k$ and $q_k$ (thus pointing at the next instruction of $M$ to be simulated), and send a contribution of 1 to variables $p_j$ and $q_j$, after setting their value to zero. On the other hand, programs (24) and (25) send a contribution of $-1$ to $p_j$ and $q_j$, so that their final value will be zero.

Now assume that $p_j = q_j = 1$ and $r_i = 0$. In this case, the value of $r_i$ should be kept equal to zero, and the computation should continue with instruction $l$. Program (20) sends a contribution of $-2$ to $r_i$. This time, however, program (21) is also executed; its effect is sending a contribution of 2 to $r_i$, after setting it to zero (so that its final value will be zero), and setting to 1 the value of variables $p_l$ and $q_l$. Programs (22) and (23) are inactive, and hence are not executed. Finally, programs (24) and (25) send a contribution of $-1$ to $p_j$ and $q_j$, so that their final value will be zero.

It follows from the description given above that the simulation is correct, and that after the simulation of each instruction the value of variable $r_i$ is exactly the double of the contents of register $i$, for $1 \leq i \leq n$. If and when the program counter of $M$ reaches the value $m$, the corresponding variables $p_m$ and $q_m$ assume value 1 and the computation reaches a final configuration; the result of the computation is then contained in variables $r_1, \ldots, r_\beta$. □

Let $2\mathbb{N}\mathbf{RE}$ denote the family of recursively enumerable sets of even natural numbers: $2\mathbb{N}\mathbf{RE} = \{\{2x \mid x \in X\} \mid X \in \mathbb{N}\mathbf{RE}\}$. By taking $\beta = 0$ and $\alpha \geq 1$ (resp., $\alpha = 0$ and $\beta \geq 1$) in the previous proof one obtains a characterization of the recursively enumerable sets of vectors of even natural numbers by accepting (resp., generating) one-parallel EN P systems. In particular, by putting $\beta = 0$ and $\alpha = 1$ or $\alpha = 0$ and $\beta = 1$, we obtain:

$$2\mathbb{N}\mathbf{RE} = \mathbf{ENP}_1(poly^1(1), oneP)$$

As a byproduct of Theorem 6 we also obtain a small universal deterministic EN P system that computes any partial recursive function $f : 2\mathbb{N} \rightarrow 2\mathbb{N}$, by simulating

the universal deterministic register machine illustrated in Figure 1. With respect to the small EN P system described in the proof of Theorem 5 we have removed two auxiliary variables from the programs that simulate each decrement instruction, hence the new system consists of 105 programs and 120 variables. As discussed after the proof of Theorem 5, this small EN P system is deterministic too and hence it also works in the all-parallel mode; however, it works only with even natural numbers as inputs and outputs.

Of course one would desire a characterization of $\mathbb{N}\mathbf{RE}$ (instead of $2\mathbb{N}\mathbf{RE}$) by one-parallel EN P systems having linear production functions, each depending upon just one variable. We can actually obtain such a characterization by using the EN P system $\Pi_M$ described in the proof of the previous theorem as a subroutine. The idea is to produce a new one-parallel EN P system $\Pi'_M$ that, given a vector from $\mathbb{N}^\alpha$ as input, prepares a corresponding input vector for $\Pi_M$ by doubling its components. Then $\Pi_M$ is used to compute the output vector from $\mathbb{N}^\beta$, if it exists. At this point $\Pi'_M$ should take this output and halve each component, to produce its output. To avoid this further step, we proceed as follows: while preparing the input for $\Pi_M$, $\Pi'_M$ also makes a copy of its input into additional variables $s_i$, for $1 \leq i \leq n$. Then we modify the programs of $\Pi_M$ in such a way that, while simulating a (possibly nondeterministic) register machine $M$, it keeps in $s_i$ the contents of the registers, and in $r_i$ the doubles of such contents. So the programs use variables $r_i$ to correctly perform the simulation, while at the end of the computation the result will be immediately available in variables $s_i$. The details are given in the proof of the following theorem, where the systems $\Pi_M$ and $\Pi'_M$ are combined together.

**Theorem 7.** *Each partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ ($\alpha \geq 0,\ \beta \geq 0$) can be computed by a one-membrane EN P system working in the one-parallel mode, having linear production functions that use each at most one variable.*

*Proof.* Like in the proofs of Theorems 4 and 6, we build a one-parallel EN P system $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$ that simulates a nondeterministic register machine $M = (n, P, m)$ that computes $f$, as follows:

- $H = \{s\}$ is the label of the only membrane (the skin) of $\Pi_M$;
- $\mu = [\,]_s$ is the membrane structure;
- $Var_1 = \{r_1, \ldots, r_n\} \cup \{s_1, \ldots, s_n\} \cup \{t_1, \ldots, t_n\} \cup \{p\} \cup \{p_0, \ldots, p_m\} \cup \{q_0, \ldots, q_m\} \cup \{z_{j,1}, z_{j,2}, z_{j,3}$ for all $0 \leq j < m\}$;
- $Pr_1 = \{3t_i \to 2|r_i + 1|s_i$ for all $1 \leq i \leq \alpha\} \cup \{2p \to 1|p_0 + 1|q_0\} \cup \{z_{j,1} + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_k + 1|q_k,\ z_{j,1} + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_l + 1|q_l,\ z_{j,2} - 1|_{p_j} \to 1|q_j,\ z_{j,3} - 1|_{q_j} \to 1|p_j$ for all instructions $j : (\text{INC}(i), k, l) \in P\} \cup \{z_{j,1} - 3|_{p_j} \to 2|r_i + 1|s_i,\ r_i + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_l + 1|q_l,\ 2p_j|_{r_i} \to 1|p_j + 1|p_k,\ 2q_j|_{r_i} \to 1|q_j + 1|q_k,\ z_{j,2} - 1|_{p_j} \to 1|q_j,\ z_{j,3} - 1|_{q_j} \to 1|p_j\}$ for all instructions $j : (\text{DEC}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values of the variables of $Var_1$, obtained by putting:
  - $t_i = x_i$ (the input values of $f$) for all $1 \leq i \leq \alpha$;
  - $t_i = 0$ for all $\alpha + 1 \leq i \leq n$;

- $r_i = s_i = 0$ for all $1 \leq i \leq n$;
- $p = 1$;
- $p_j = q_j = 0$ for all $0 \leq j \leq m$;
- $z_{j,1} = z_{j,2} = z_{j,3} = 0$ for all $0 \leq j < m$.

The input values $x_1, \ldots, x_\alpha$ of $f$ are introduced into the P system as the initial values of variables $t_1, \ldots, t_\alpha$. Moreover, the value of variable $p$ is set to 1. In the first step of its computation, the P system will copy the values of $t_1, \ldots, t_\alpha$ to $s_1, \ldots, s_\alpha$, and the double of these values to variables $r_1, \ldots, r_\alpha$. So doing, after the simulation of each instruction of $M$ variables $s_1, \ldots, s_n$ will contain the values of the registers of $M$, while $r_1, \ldots, r_n$ will contain their doubles. While making these copies, the value of variable $p$ is copied to both $p_0$ and $q_0$, in order to start the simulation of $M$. The simulation proceeds much like in the way described in the proof of Theorem 6; the programs there illustrated are here modified in order to deal with the new variables. If and when the simulation reaches a final configuration, variables $s_1, \ldots, s_\beta$ contain the result of the computation.

The initialization step is performed by executing the following programs:

$$3t_i \to 2|r_i + 1|s_i \quad \text{for all } 1 \leq i \leq \alpha$$
$$2p \to 1|p_0 + 1|q_0$$

Each increment instruction $j : (\text{INC}(i), k, l)$ of $M$ is simulated in one step by the execution of the following programs:

$$z_{j,1} + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_k + 1|q_k \tag{26}$$
$$z_{j,1} + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_l + 1|q_l \tag{27}$$
$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{28}$$
$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{29}$$

The simulation is analogous to the one described in the proof of Theorem 6, with the difference that when adding 2 to $r_i$ the system now also increments $s_i$; to do so, the production value computed by the first two programs must be 5 instead of 4. Once again, if the machine $M$ to be simulated is deterministic then program (27) disappears and the simulation itself becomes deterministic.

Each decrement instruction $j : (\text{DEC}(i), k, l)$ is simulated in one step by the execution of the following programs:

$$z_{j,1} - 3|_{p_j} \to 2|r_i + 1|s_i \tag{30}$$
$$r_i + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_l + 1|q_l \tag{31}$$
$$2p_j|_{r_i} \to 1|p_j + 1|p_k \tag{32}$$
$$2q_j|_{r_i} \to 1|q_j + 1|q_k \tag{33}$$
$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{34}$$
$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{35}$$

The simulation is analogous to the one described in the proof of Theorem 6, with the only difference that when subtracting or adding 2 to $r_i$ by programs (30) and (31), respectively, the system now also decrements or increments $s_i$, respectively.

It can be easily checked that the simulation is correct, and that after simulating each instruction of $M$ the values of variable $s_i$ (resp., $r_i$) is equal to the contents (resp., the double of the contents) of register $i$, for $1 \leq i \leq n$. If and when the program counter of $M$ reaches the value $m$, the corresponding variables $p_m$ and $q_m$ assume value 1 and the computation reaches a final configuration; the result of the computation can then be recovered from variables $s_1, \ldots, s_\beta$. □

By taking $\beta = 0$ and $\alpha \geq 1$ in the previous proof, we obtain the following result concerning the accepting variant of EN P systems working in the one-parallel mode.

**Corollary 4.** *For any $L \in \mathbf{Ps}(\alpha)\mathbf{RE}$ there exists a one-membrane EN P system, having linear production functions each depending upon at most one variable, that accepts $L$ by working in the one-parallel mode.*

On the other hand, by taking $\alpha = 0$ and $\beta \geq 1$ we get the following characterization of $\mathbf{Ps}(\beta)\mathbf{RE}$ by the generating variant of EN P systems working in the one-parallel mode.

**Corollary 5.** *For any $L \in \mathbf{Ps}(\beta)\mathbf{RE}$ there exists a one-membrane (nondeterministic) EN P system, having linear production functions each depending upon at most one variable, that generates $L$ by working in the one-parallel mode.*

By putting $\alpha = 1$ and $\beta = 0$ in Corollary 4, and $\alpha = 0$ and $\beta = 1$ in Corollary 5, we obtain the following characterization:

$$\mathbb{N}\mathbf{RE} = \mathbf{ENP}_1(poly^1(1), oneP)$$

Moreover, it can be easily checked that when the register machine $M$ simulated in Theorem 7 and in Corollary 4 is deterministic, the simulating EN P system $\Pi_M$ works in the all-parallel mode. This means that the above construction leads to a further characterization of $\mathbb{N}\mathbf{RE}$ by all-parallel recognizing EN P systems having linear production functions of one variable, alternative to the one obtained by Theorem 2.

Another consequence of Theorem 7 is that there exists a further small universal one-parallel deterministic EN P system, as stated in the following theorem.

**Theorem 8.** *There exists a universal one-parallel deterministic EN P system of degree* 1*, having* 137 *variables and* 108 *programs.*

*Proof.* The system mentioned in the statement simulates the small universal deterministic register machine $M_u$ reported in Figure 1, and is built according to the description given in the proof of Theorem 7, as we have done in the proofs of Theorems 3 and 5. The number of increment and decrement instructions of $M_u$ are 9 and 13, respectively, and each of them is translated to 3 and 6 programs

of the small universal EN P system, respectively. The initialization step requires further $\alpha+1=3$ programs, since $M_u$ is fed with two input values: the "code" of $f$ and its input. We thus obtain a total of 108 programs. As for variables, $8 \cdot 3 = 24$ are used to simulate the registers of $M_u$, and 46 are used to denote the value of its program counter; moreover, there are 3 auxiliary variables for each instruction of $M$, and one variable ($p$) which used to trigger the start of the simulation, for a total of 137 variables. □

Since the universal register machine $M_u$ simulated in Theorem 8 is deterministic, the simulating small EN P system is deterministic too, and works both in the all-parallel as well as in the one-parallel mode. By comparing the number of variables and programs in all "small" EN P systems described in this paper, we see that the smallest is the one described in Theorem 3, containing only 31 variables and 61 programs. However such a small EN P system is not able to work in the one-parallel mode, hence in case we are forced to do so we must resort to one of the others described in this paper; the choice will depend upon the parameter (number of variables or number of programs) we want to minimize, as well as whether we are willing to work with even inputs and outputs. It is left as an open problem to prove that these are the smallest possible universal EN P systems, or finding instead smaller ones. Designing sets of programs that simulate consecutive INC and DEC instructions of $M_u$, as it has already been done in [9] and several other times in the literature, could be a hint for finding smaller systems.

## 4 Producing Output in Separate Variables

In all EN P systems described above, the output is considered to be the value of some specified variables in the final configuration, if and when this is reached. This is different from how EN P systems produce their output in most existing papers: usually, some separate output variables are considered, and the output of the system is defined as the set of all values assumed by these variables during the entire computation. In this section we prove that each of our EN P systems can be easily modified in order to produce its output according to this latter way.

**Theorem 9.** *The EN P systems used in Theorems 2, 4, 6 and 7 can be modified so that their output is produced into separate variables.*

*Proof.* Let $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0))$ be one of the EN P systems mentioned in the statement, simulating a register machine $M$ computing the partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$. Let $x_1, \ldots, x_\beta$ denote the output variables of $\Pi_M$, that is, variables $r_1, \ldots, r_\beta$ for Theorems 2, 4, 6 and variables $s_1, \ldots, s_\beta$ in Theorem 7. Note that, by construction, these variables contain the value of $f$ if and when a final configuration is reached, and this happens if and only if $p_m$ (the variable indicating label $m$ of the program of $M$) assumes value 1.

We modify $\Pi_M$ by introducing the following new variables:

- $\{y_1, \ldots, y_\beta\}$, whose values are kept identical to $x_1, \ldots, x_\beta$ until $p_m$ becomes 1 (if this happens);
- $\{z_1, \ldots, z_\beta\}$, as the new output variables;
- $\{u_1, \ldots, u_\beta\}$, as flags;

and programs:

$$\beta p_m \to 1|u_1 + \ldots + 1|u_\beta \tag{36}$$

$$u_i \to 1|y_i \qquad \text{for all } 1 \leq i \leq \beta \tag{37}$$

$$x_i|_{y_i} \to z_i \qquad \text{for all } 1 \leq i \leq \beta \tag{38}$$

Moreover, each program already present in $\Pi_M$ that changes the value of an output variable $x_i$ is modified in order to also apply the same change to the new variable $y_i$, as done in the proof of Theorem 7. So doing, after simulating each instruction of $M$ the values of variables $x_i$ and $y_i$ will be the same for all $1 \leq i \leq \beta$. Since $y_1, \ldots, y_\beta$ never appear in the production functions of these modified programs, no change is caused to the behavior of $\Pi_M$.

All new variables are initialized to zero before the computation starts. During the first computation step variables $y_1, \ldots, y_\alpha$ are initialized to the values of $x_1, \ldots, x_\alpha$, as in the initialization step of Theorem 7. The computation then proceeds as prescribed by the programs of $\Pi_M$. If and when the computation reaches a final configuration then program (36) is executed, with the effect of zeroing $p_m$ and setting $u_1, \ldots, u_\beta$ to 1. When this happens, by programs (37) the values of $y_1, \ldots, y_\beta$ are incremented, thus becoming larger than the values of $x_1, \ldots, x_\beta$. This means that programs (38) can now be applied, with the effect of copying the values of the original output variables $x_1, \ldots, x_\beta$ to the new output variables $z_1, \ldots, z_\beta$.

On the other hand, note that before and after reaching a final configuration of $\Pi_M$ the value of variables $z_1, \ldots, z_\beta$ is never affected. In fact, when $p_m = 0$ program (36) has no effect, since it distributes a contribution of 0 to $u_1, \ldots, u_\beta$, leaving their value unaltered. This happens both before $p_m$ becomes 1, and after executing program (36). Programs (37) increment the values of $y_1, \ldots, y_\beta$ only once, when $u_1 = \ldots = u_\beta = 1$, otherwise they produce no effect. Finally, programs (38) are first executed as soon as the values of $y_1, \ldots, y_\beta$ become larger than that of $x_1, \ldots, x_\beta$, after which they distribute a contribution of zero to $z_1, \ldots, z_\beta$.

So the only value assumed by the new output variables $z_1, \ldots, z_\beta$, besides zero, is the output value of $M$. □

## 5 Conclusions and Directions for Further Work

In this paper we have studied the computational power of enzymatic numerical P systems working in the all-parallel and one-parallel modes.

We have improved some previously known universality results, in terms of number of membranes and number of variables used in the production functions.

So, by using a flattening technique, we have first shown that every EN P system working either in the all-parallel or in the one-parallel mode can be simulated by an equivalent one-membrane EN P system working in the same mode. Then we have shown that linear production functions, each depending upon at most one variable, suffice to reach universality for both computing modes. As a byproduct we have obtained several small universal deterministic EN P systems, the smallest one having only 31 variables and 61 programs.

It is left open whether smaller universal EN P systems exist. It is also left open whether the known universality result on *sequential* EN P systems contained in [17] — a characterization of $\mathbb{N}\mathbf{RE}$ by sequential EN P systems of degree 7, whose production functions are polynomials of degree at most 5, each depending upon at most 5 variables — can be improved.

**Acknowledgements**

The ideas exposed in this paper emerged during the *Eleventh Brainstorming Week on Membrane Computing (BWMC 2013)*, held in Seville from February 4$^{\text{th}}$ to February 8$^{\text{th}}$, 2013.

This research was partially funded by Lombardy Region under project NEDD.

## References

1. P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg: Membrane systems with promoters/inhibitors. *Acta Informatica* 38(10):695–720, 2002.
2. C. Buiu, C.I. Vasile, O. Arsene: Development of membrane controllers for mobile robots. *Information Sciences* 187:33–51, 2012.
3. R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae* 49(1-3):81–102, 2002.
4. R. Freund, Gh. Păun: On the number of non-terminals in graph-controlled, programmed, and matrix grammars. In: M. Margenstern, Y. Rogozhin (Eds.), *Universal Machines and Computations*, Chişinău, 2001, LNCS 2055, Springer-Verlag, 2001, pp. 214–225.
5. R. Freund, Gh. Păun: From regulated rewriting to computing with membranes: collapsing hierarchies. *Theoretical Computer Science* 312:143–189, 2004.
6. I. Korec: Small universal register machines. *Theoretical Computer Science* 168:267–301, 1996.
7. Y. Matijasevitch: *Hilbert's tenth problem*. MIT Press, Cambridge, London, 1993.
8. M.L. Minsky: *Computation. finite and infinite machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
9. A. Păun, Gh. Păun: Small universal spiking neural P systems. *Biosystems* 90(1):48–60, 2007.
10. Gh. Păun, R. Păun: Membrane computing and economics: numerical P systems. *Fundamenta Informaticae* 73:213–227, 2006.
11. Gh. Păun, G. Rozenberg, A. Salomaa (eds.): *The Oxford handbook of membrane computing*. Oxford University Press, 2010.

12. A.B. Pavel: *Membrane controllers for cognitive robots*. Master's thesis, Department of Automatic Control and System Engineering, Politechnica University of Bucharest, Romania, 2011.
13. A.B. Pavel, O. Arsene, C. Buiu: Enzymatic numerical P systems – A new class of membrane computing systems. *IEEE Fifth International Conference on Bio-Inspired Computing: Theory and Applications (BIC-TA)*, IEEE, 2010, pp. 1331–1336.
14. A.B. Pavel, C. Buiu: Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing* 11(3):387–393, 2012.
15. A.B Pavel, C.I. Vasile, I. Dumitrache: Robot localization implemented with enzymatic numerical P systems. In: T.J. Prescott et al. (Eds.), Proceedings of *Living Machines 2012*, Barcelona, Spain, July 9–12, 2012, LNAI 7375, Springer-Verlag, 2012, pp. 204–215.
16. C.I. Vasile, A.B. Pavel, I. Dumitrache: Universality of enzymatic numerical P systems. *International Journal of Computer Mathematics*, 2013. DOI: 10.1080/00207160.2012.748897
17. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: On the power of enzymatic numerical P systems. *Acta Informatica* 49(6):395–412, 2012.