Improved Universality Results for Parallel Enzymatic Numerical P Systems

ALBERTO LEPORATI[1]*, ANTONIO E. PORRECA[1], CLAUDIO ZANDRON[1], GIANCARLO MAURI[1]

Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336/14, 20126 Milano, Italy

Received ...; In final form ...

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.

Key words: Enzymatic numerical P systems, universality, all-parallel mode, one-parallel mode, flattening, small universal systems

1 INTRODUCTION

Numerical P systems have been introduced in [9] 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, whose values evolve by means of *programs*. In [9] it is proved that nondeterministic numerical P systems with polynomial production functions char-

* email: `alberto.leporati@unimib.it`

acterize 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 [5] as an extension of numerical P systems in which some variables, named the *enzymes*, control the application of the rules; their most promising application seems to be the simulation of control mechanisms of mobile and autonomous robots, as shown in [2, 6, 7]. Formally, an *enzymatic numerical P system* is a construct of the form:

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

where $m \geq 1$ is the degree of the system (the number of membranes), H is an alphabet of labels, μ 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 [5] 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 change the values of variables during computations. Each program consists 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 [11]. 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 $v_1, \dots, v_{n_i}$ be all these variables; let $x_{1,i}, \dots, x_{k_i,i}$ be some variables from Var_i, let $F_{l,i}(x_{1,i}, \dots, x_{k_i,i})$ be the production function of a given program $P_{l,i} \in Pr_i$, and let $c_{l,1}, \dots, c_{l,n_i}$ be natural numbers. The program $P_{l,i}$ is written in the following form:

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

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 [11], 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 [11], in this paper we assume that this is always the case by construction, so we will deal only with such systems; see [9] for other possible approaches.

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

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

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 $e_{j,i}(t) > \min(x_{1,i}(t), \ldots, x_{k_i,i}(t))$. Variable $e_{j,i}$ operates like an *enzyme*, that enables the execution of the program, but — as it also happens 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, they can receive "contributions" from other programs and regions.

A *configuration* of Π at time $t \in \mathbb{N}$ is given by the values of all the variables of Π 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 Π. The *initial configuration* can thus be described as the sequence $(Var_1(0), \ldots, Var_m(0))$. The system Π evolves from a configuration to another one by means of *computation steps*, in which one or more programs of Π are executed according to a *mode* of computation. In [11], 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 in both settings.

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 interpreted as 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 consider is the fact that 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 present are designed to produce only non-negative numbers in their output variables when the input variables (if present) are assigned with non-negative numbers. Another possibility, mentioned in [11] 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 (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. If one of the parameters m, n, r is not bounded by a constant value, we replace it by $*$.

Some results concerning the computational power of EN P systems are reported in [11, 10]. In particular, the following characterizations of $\mathbb{N}\mathbf{RE}$ (the family of recursively enumerable sets of natural numbers) are proved:

$$\begin{aligned}\mathbb{N}\mathbf{RE} &= \mathbf{ENP}_7(poly^5(5), seq) = \mathbf{ENP}_*(poly^1(2), oneP) \\ &= \mathbf{ENP}_4(poly^1(6), allP)\end{aligned}$$

In section 2 we 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. Such characterizations will be obtained by simulating 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:

- $j : (\textsc{inc}(r), k, l)$, 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.

- $j : (\textsc{dec}(r), k, l)$, 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.

In a *deterministic* n-register machine all INC instructions have the form $j : (\textsc{inc}(r), k, k)$, and can be simply written as $j : (\textsc{inc}(r), k)$.

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 $\mathbb{N}\mathbf{RE}$ [4]. More precisely, register machines can be used either in the computing, generating or accepting mode (see, for example, [1]). Working in the *computing mode*, 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_α in registers 1 to α, 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 β containing r_1 to r_β, 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 output registers are neglected, and a vector of non-negative integers is accepted if and only if the machine halts. So doing, for any recursively enumerable set $L \subseteq \mathbf{Ps}(\alpha)\mathbf{RE}$ of vectors of non-negative integers there exists a deterministic register machine M with $(\alpha + 2)$ registers accepting L. In the *generating* mode we need nondeterministic register machines: input registers are neglected, and vectors of non-negative integers are generated in the output registers if and when the machine, starting with all registers being empty, halts. So doing, for any recursively enumerable set $L \subseteq \mathbf{Ps}(\beta)\mathbf{RE}$ of vectors of non-negative integers there exists a nondeterministic register machine M with $(\beta + 2)$ registers generating L.

2 UNIVERSALITY OF EN P SYSTEMS

Our aim is to improve the universality results, shown in [11, 10], concerning all-parallel and one-parallel EN P systems. We first prove that these P systems can be "flattened".

Theorem 1. *Let* Π *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* Π' *of degree* 1 *that computes (resp., generates, accepts) the same function (resp., family of sets) using the same rule application mode.*

Proof. Consider an EN P system $\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0)))$. 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 Π' of degree 1, by putting all the variables and all the programs of Π — keeping both indexes, also in the variables occurring in programs — in the membrane of Π'. Clearly, this establishes a bijection between the variables (resp., programs) of Π and the corresponding variables (resp., programs) of Π'. So any program $P_{l,i}$ of Π still operates on the correct variables when transformed and put into Π', regardless of whether or not it uses an enzyme. Input and output variables are also preserved. If Π 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 Π' by letting it work in the all-parallel mode. The same applies when Π and Π' work in the one-parallel mode, so the claim of the theorem follows. Note that if Π works in the sequential mode, then at each computation step at most one program is selected in each region; this means that globally Π executes a set of programs which cannot be captured in Π' by any of the sequential, one-parallel and all-parallel modes. $\square$

This result already allows us to improve the universality results shown in [11, 10] 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, so doing one-parallel EN P systems still require an unbounded number of variables, since each "new" variable in Π' is indexed with the region of Π it comes from.

In what follows we improve both these results; let us start with all-parallel EN P systems.

Theorem 2. *Every deterministic register machine can be simulated 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. Let $M = (n, P, m)$ be a deterministic register machine with n registers and m instructions. The initial instruction of P has the label 0 and the machine halts if and when the program counter assumes the value m. We construct the EN P system $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$ of degree 1, that simulates M, as follows:

- $H = \{s\}$ is the label of the only membrane (the skin) of Π_M;
- $\mu = [\,]_s$ is the membrane structure;
- $Var_1 = \{r_1, \ldots, r_n\} \cup \{p_0, \ldots, p_m\}$;

- $Pr_1 = \{$program (2) for all instructions $j : (\textsc{inc}(i), k) \in P\} \cup \{$programs (3)–(6) for all instructions $j : (\textsc{dec}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values for the variables of Var_1, obtained by setting:
  - $r_i =$ the contents of the i-th register of M, for all $1 \leq i \leq n$;
  - $p_0 = 1$;
  - $p_j = 0$ for all $1 \leq j \leq m$.

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.

Each increment instruction $j : (\textsc{inc}(i), k)$ of M is simulated by Π_M in one step by the execution of the program

$$2p_j \to 1|r_i + 1|p_k \tag{2}$$

This program is executed at *every* computation step of Π_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. Thus, no variable is affected 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 $j : (\textsc{dec}(i), k, l)$ is simulated in one step by the parallel execution of the following programs:

$$-p_j \to 1|r_i \tag{3}$$
$$r_i + 2|_{p_j} \to 1|r_i + 1|p_l \tag{4}$$
$$p_j \to 1|p_k \tag{5}$$
$$r_i - 1|_{p_j} \to 1|p_k \tag{6}$$

If $p_j = 0$, programs (4) and (6) are not enabled (since by construction $r_i \geq 0$ and thus $p_j \leq r_i$), while programs (3) and (5) 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 $p_j = 0$ does not cause interference in 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 (3) correctly decrements r_i, and program (5) 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 (4) and (6) 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 (3) sends a contribution of -1 to r_i, while program (5) incorrectly sets p_k to 1; both programs set p_j to zero. This time, however, programs (4) and (6) are also executed. Both set the value of r_i to zero. After that, program (4) adds 1 to r_i, canceling the effect of program (3); as a result, the value assumed by r_i after the execution of the two programs is zero. Program (4) also makes p_l assume the value 1, thus correctly pointing at the next instruction of M to be simulated. Finally, program (6) gives a contribution of -1 to p_k, canceling the effect of program (5); 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, Π_M reaches a final configuration. □

Since deterministic register machines can be used either to compute partial recursive functions or to accept recursively enumerable sets, by Theorem 2 also all-parallel EN P systems acquire the same capabilities. In particular, using EN P systems in the accepting mode, with a single input number provided in variable r_1, we obtain the following characterization:

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

By simulating *small* deterministic register machines, we also obtain small universal all-parallel EN P systems:

Corollary 1. *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 [3], and illustrated in Figure 1. This machine has $n = 8$ registers

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

FIGURE 1
The small universal deterministic register machine defined in [3]

and $m = 22$ instructions. By following the arguments given in the proof of Theorem 2 we construct an all-parallel EN P system Π_{M_u}, of degree 1, that simulates it. It is easily checked that Π_{M_u} has 31 variables: 8 for the registers, and 23 for the program counter. The number of increment and decrement instructions of M_u are 9 and 13, respectively. Each increment instruction is translated to 1 program of Π_{M_u} while each decrement instruction produces 4 programs, for a total of 61 programs. $\square$

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

Theorem 3. *Every (deterministic or nondeterministic) register machine can be simulated 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. Let $M = (n, P, m)$ be a nondeterministic register machine with n registers and m instructions. We construct the one-membrane EN P system $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$ that simulates M, as follows:

- $H = \{s\}$ is the label of the only membrane (the skin) of Π_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 $j : (\textsc{inc}(i), k, l) \in P\} \cup \{z_{j,1}, z_{j,2}, z_{j,3}, z_{j,4}, z_{j,5}$ for all instructions $j : (\textsc{dec}(i), k, l) \in P\}$;

- $Pr_1 = \{$programs (7)–(10) for all instructions $j : (\textsc{inc}(i), k, l) \in P\} \cup$ $\{$programs (11)–(16) for all instructions $j : (\textsc{dec}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values for the variables of Var_1, obtained by setting:
  - $r_i =$ the contents of the i-th register of M, for all $1 \leq i \leq n$;
  - $p_0 = q_0 = 1$;
  - $p_j = q_j = 0$ 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 P contains the instruction $j : (\textsc{inc}(i), k, l)$;
  - $z_{j,1} = z_{j,2} = z_{j,3} = z_{j,4} = z_{j,5} = 0$ for all $0 \leq j < m$ such that $j : (\textsc{dec}(i), k, l) \in P$.

Differently from what we have done in Theorem 2, 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.

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

$$z_{j,1} + 3|_{p_j} \to 1|r_i + 1|p_k + 1|q_k \tag{7}$$

$$z_{j,1} + 3|_{p_j} \to 1|r_i + 1|p_l + 1|q_l \tag{8}$$

$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{9}$$

$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{10}$$

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 (9) and (10) as well as one among programs (7) and (8) are executed, since variable $z_{j,1}$ makes these latter programs compete in the one-parallel mode of application. Assume that program (7) is executed (a similar argument holds if (8) is chosen instead): its effect is incrementing r_i and setting p_k and q_k to 1, thus correctly pointing at the next instruction of M to be simulated. The effect of programs (9) and (10) 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 $j : (\textsc{inc}(i), k)$ is performed by using the same programs without (8). In this case no competition occurs between the programs, and the simulation is also deterministic.

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

$$z_{j,1} - 1|_{p_j} \to 1|r_i \tag{11}$$

$$r_i + 3|_{p_j} \to 1|r_i + 1|p_l + 1|q_l \tag{12}$$

$$z_{j,2} + 2p_j|_{r_i} \to 1|p_j + 1|p_k \tag{13}$$

$$z_{j,3} + 2q_j|_{r_i} \to 1|q_j + 1|q_k \tag{14}$$

$$z_{j,4} - 1|_{p_j} \to 1|q_j \tag{15}$$

$$z_{j,5} - 1|_{q_j} \to 1|p_j \tag{16}$$

If $p_j = q_j = 0$ then programs (11), (12), (15) and (16) are not enabled, while programs (13) and (14) 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 the overall simulation is not affected.

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 (11) correctly decrements r_i, whereas program (12) is not executed since $r_i \geq p_j$. Programs (13) and (14) 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. Simultaneously, programs (15) and (16) 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 (11) sends a contribution of -1 to r_i. This time, however, program (12) 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 (13) and (14) are inactive, and hence are not executed. Finally, programs (15) and (16) send a contribution of -1 to p_j and q_j, so that their final value will be zero.

From the description given above it follows 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 thus reaches a final configuration. $\square$

A direct consequence of Theorem 3 is the characterization of $\mathbb{N}\mathbf{RE}$ as $\mathbf{ENP}_1(poly^1(2), oneP)$, alternative to the one described in [11]. Once again, by simulating the small deterministic register machine reported in Figure 1 we obtain a small EN P system.

Corollary 2. *There exists a universal one-parallel deterministic EN P system of degree* 1*, having* 146 *variables and* 105 *programs.*

Let us note that, since the EN P system mentioned in the statement of Corollary 2 is deterministic, it also works in the all-parallel mode, albeit in this case the system referred to in Corollary 1 is smaller.

By looking at the operation of the EN P system described in the proof of Theorem 3, we can see that the only programs whose production functions depend upon two variables are programs (13) and (14). 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 $p_j = q_j = 1$. This fact suggests that we can modify the proof of Theorem 3 so that each variable r_i contains the *double* of the contents of the i-th register of the simulated register machine M. So doing, we can get rid of variables $z_{j,2}$ and $z_{j,3}$ in programs (13) and (14), thus obtaining a one-parallel EN P system whose linear production functions each depend on just one variable. Each increment instruction $j : (\text{INC}(i), k, l)$ of M would be simulated in one step by the following programs:

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

$$z_{j,1} + 4|_{p_j} \to 2|r_i + 1|p_l + 1|q_l \tag{18}$$

$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{19}$$

$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{20}$$

where 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 (17) and (18) compete in the one-parallel mode. If the machine M to be simulated is deterministic then program (18) disappears, and also the simulation becomes deterministic.

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

$$z_{j,1} - 2|_{p_j} \to 1|r_i \tag{21}$$

$$r_i + 4|_{p_j} \to 2|r_i + 1|p_l + 1|q_l \tag{22}$$

$$2p_j|_{r_i} \to 1|p_j + 1|p_k \tag{23}$$

$$2q_j|_{r_i} \to 1|q_j + 1|q_k \tag{24}$$

$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{25}$$

$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{26}$$

The simulation is analogous to the one described in the proof of Theorem 3, with a few small differences.

The case when $p_j = q_j = 0$ operates just like in the proof of Theorem 3.

Now assume that $p_j = q_j = 1$ and $r_i > 0$. Program (21) correctly decrements r_i (subtracting 2 from its value), whereas program (22) is not executed since $r_i > p_j$. Programs (23) and (24) 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 (25) and (26) 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 (21) sends a contribution of -2 to r_i. This time, however, program (22) 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 (23) and (24) are inactive, and hence are not executed. Finally, programs (25) and (26) send a contribution of -1 to p_j and q_j, so that their final value will be zero.

From the above description it follows that after the simulation of each instruction the value of variable r_i is exactly the double of the contents of register i, for $1 \leq i \leq n$, and that variables $p_0, \ldots, p_m$ and $q_0, \ldots, q_m$ correctly indicate the next instruction of M to be simulated.

If we denote by $2\mathbb{N}\mathbf{RE} = \{\{2x \mid x \in X\} \mid X \in \mathbb{N}\mathbf{RE}\}$ the family of recursively enumerable sets of even natural numbers, then due to the above construction we obtain the following characterization:

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

As a further byproduct we also obtain a small universal deterministic EN P system that computes any partial recursive function $f : 2\mathbb{N} \to 2\mathbb{N}$, by simulating the universal deterministic register machine illustrated in Figure 1. With respect to the small EN P system mentioned in Corollary 2 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 2, 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. Let Π_M be the EN P system described above, working with even values in variables r_i. The idea is to produce a new one-parallel EN P system Π'_M that, given a vector from $\mathbb{N}^\alpha$ as input, prepares a corresponding input vector for Π_M by doubling its components. Then Π_M is used to compute the (double of the) output vector from $\mathbb{N}^\beta$, if it exists. At this point Π'_M should take this vector and halve each component, to produce its output. To avoid this last step and directly produce the output vector, we proceed as follows: while preparing the input for Π_M, Π'_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 Π_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 Π_M and Π'_M are combined together.

Theorem 4. *Every (deterministic or nondeterministic) register machine can be simulated 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. Let $M = (n, P, m)$ be a nondeterministic register machine, having n registers and m instructions in its program P. We build a one-parallel EN P system $\Pi_M = (1, H, \mu, (Var_1, Pr_1, Var_1(0)))$ that simulates M, as follows:

- $H = \{s\}$ is the label of the only membrane (the skin) of Π_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 $0 \leq j < m\}$;

- $Pr_1 = \{$programs (27) for all $1 \leq i \leq n\} \cup \{$program (28)$\} \cup \{$programs (29)–(32) for all instructions $j : (\textsc{inc}(i), k, l) \in P\} \cup \{$programs (33)–(38) for all instructions $j : (\textsc{dec}(i), k, l) \in P\}$;
- $Var_1(0)$ is the vector of initial values for the variables of Var_1, obtained by setting:
  - $t_i =$ the contents of the i-th register of M, for all $1 \leq i \leq n$;
  - $r_i = s_i = 0$ 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 contents of the registers of M are introduced into the P system as the initial values of variables $t_1, \ldots, t_n$. During the simulation, the variables $s_1, \ldots, s_n$ will also contain the values of the registers of M, while variables $r_1, \ldots, r_n$ will contain the doubles of such values. In the first computation step, the following programs

$$3t_i \to 2|r_i + 1|s_i \quad \text{for all } 1 \leq i \leq n \tag{27}$$

$$2p \to 1|p_0 + 1|q_0 \tag{28}$$

initialize variables $s_1, \ldots, s_n$ and $r_1, \ldots, r_n$, zeroing $t_1, \ldots, t_n$; simultaneously, the value of variable p is moved to both p_0 and q_0, in order to start the simulation of M. In subsequent computation steps programs (27) and (28) are always executed; however they produce no effect, since variables $t_1, \ldots, t_n$ and p are zero.

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

$$z_{j,1} + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_k + 1|q_k \tag{29}$$

$$z_{j,1} + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_l + 1|q_l \tag{30}$$

$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{31}$$

$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{32}$$

Note that, when adding 2 to r_i, the system now also increments s_i. Once again, if the machine M to be simulated is deterministic then program (30) disappears and the simulation itself becomes deterministic.

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

$$z_{j,1} - 3|_{p_j} \to 2|r_i + 1|s_i \tag{33}$$
$$r_i + 5|_{p_j} \to 2|r_i + 1|s_i + 1|p_l + 1|q_l \tag{34}$$
$$2p_j|_{r_i} \to 1|p_j + 1|p_k \tag{35}$$
$$2q_j|_{r_i} \to 1|q_j + 1|q_k \tag{36}$$
$$z_{j,2} - 1|_{p_j} \to 1|q_j \tag{37}$$
$$z_{j,3} - 1|_{q_j} \to 1|p_j \tag{38}$$

As expected, when subtracting or adding 2 to r_i by programs (33) and (34), respectively, the system now also accordingly decrements or increments s_i.

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 $1 \le i \le n$. 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 above theorem gives the following sought characterization:

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

Moreover, it can be easily checked that when the register machine M being simulated is deterministic, the simulating EN P system Π_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.

As usual, if we simulate the small universal deterministic register machine reported in Figure 1 we obtain a further small universal one-parallel deterministic EN P system:

Corollary 3. *There exists a universal one-parallel deterministic EN P system of degree* 1*, having* 137 *variables and* 108 *programs.*

Since the universal register machine M_u simulated in Corollary 3 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 Corollary 1, 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 when working in such a mode we must resort to one of the other systems 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 smaller ones instead. Designing sets of programs that simulate consecutive INC and DEC instructions of M_u, as it has already been done in [8] and several other times in the literature, could be a hint for finding smaller systems.

2.1 A Note Concerning Output Values

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, where the set of all values assumed by the output variables are collected during the entire computation. In this subsection 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 5. *The EN P systems used in Theorems 2, 3, and 4 can be modified so that their output is produced into variables whose value is set only once during the computation.*

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. Let $x_1, \ldots, x_n$ denote variables $r_1, \ldots, r_n$ of Π_M for Theorems 2, 3, and variables $s_1, \ldots, s_n$ in Theorem 4. By construction, variables $x_1, \ldots, x_\beta$ contain the output value 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 Π_M by introducing the following new variables:

- $\{y_1, \ldots, y_n\}$, whose values are kept identical to $x_1, \ldots, x_n$ until p_m becomes 1 (if this happens);
- $\{z_1, \ldots, z_\beta\}$, as the write-once variables that will contain the output;
- $\{u_1, \ldots, u_n\}$, as flags;

and programs:

$$np_m \to 1|u_1 + \ldots + 1|u_n \tag{39}$$

$$u_i \to 1|y_i \qquad \text{for all } 1 \le i \le n \tag{40}$$

$$x_i|_{y_i} \to z_i \qquad \text{for all } 1 \le i \le \beta \tag{41}$$

Moreover, each program already present in Π_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 in the proof of Theorem 4. So doing, after simulating each instruction of M the values of variables x_i and y_i will be the same for all $1 \le i \le n$. Since $y_1, \ldots, y_n$ never appear in the production functions of these modified programs, no change is caused to the behavior of Π_M.

All new variables are initialized to zero before the computation starts. During the first computation step variables $y_1, \ldots, y_n$ are initialized to the values of $x_1, \ldots, x_n$, as in the initialization step of Theorem 4. The computation then proceeds as prescribed by the programs of Π_M. If and when the computation reaches a final configuration then program (39) is executed, with the effect of zeroing p_m and setting $u_1, \ldots, u_n$ to 1. When this happens, by programs (40) the values of $y_1, \ldots, y_n$ are incremented, thus becoming larger than the values of $x_1, \ldots, x_n$. This means that programs (41) 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, before and after reaching a final configuration of Π_M the value of variables $z_1, \ldots, z_\beta$ is never affected. In fact, when $p_m = 0$ program (39) has no effect, since it distributes a contribution of 0 to $u_1, \ldots, u_n$, leaving their value unaltered. This happens both before p_m becomes 1, and after executing program (39). Programs (40) increment the values of $y_1, \ldots, y_n$ only once, when $u_1 = \ldots = u_n = 1$, otherwise they produce no effect. Finally, programs (41) 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. □

3 CONCLUSIONS AND DIRECTIONS FOR FURTHER WORK

In this paper we have improved some previously known unversality results about enzymatic numerical P systems working in the all-parallel and one-parallel modes. The improvements concern the number of membranes and the number of variables used in the production functions.

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 proved 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 characterization $\mathbb{N}\mathbf{RE} = \mathbf{ENP}_7(poly^5(5), seq)$ by *sequential* EN P systems contained in [11] can be improved.

Acknowledgements

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

REFERENCES

[1] Artiom Alhazov, Rudolf Freund, and Marion Oswald. (2006). Symbol/membrane complexity of P systems with symport/antiport rules. In Rudolf Freund, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 3850 of *Lecture Notes in Computer Science*, pages 96–113. Springer Berlin Heidelberg.

[2] Cătălin Buiu, Cristian I. Vasile, and Octavian Arsene. (2012). Development of membrane controllers for mobile robots. *Information Sciences*, 187:33–51.

[3] Ivan Korec. (1996). Small universal register machines. *Theoretical Computer Science*, 168(2):267–301.

[4] Marvin L. Minsky. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Upper Saddle River, NJ, USA.

[5] Ana B. Pavel, Octavian Arsene, and Cătălin Buiu. (2010). Enzymatic numerical P systems - a new class of membrane computing systems. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pages 1331–1336.

[6] Ana B. Pavel and Cătălin Buiu. (2012). Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, 11(3):387–393.

[7] Ana B. Pavel, Cristian I. Vasile, and Ioan Dumitrache. (2012). Robot localization implemented with enzymatic numerical P systems. In Tony J. Prescott, Nathan F. Lepora, Anna Mura, and Paul F.M.J. Verschure, editors, *Biomimetic and Biohybrid Systems*, volume 7375 of *Lecture Notes in Computer Science*, pages 204–215. Springer Berlin Heidelberg.

[8] Andrei Păun and Gheorghe Păun. (2007). Small universal spiking neural P systems. *Biosystems*, 90(1):48–60.

[9] Gheorghe Păun and Radu Păun. (July 2006). Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, 73(1–2):213–227.

[10] Cristian I. Vasile, Ana B. Pavel, and Ioan Dumitrache. (2013). Universality of enzymatic numerical P systems. *International Journal of Computer Mathematics*, 90(4):869–879.

[11] Cristian I. Vasile, Ana B. Pavel, Ioan Dumitrache, and Gheorghe Păun. (2012). On the power of enzymatic numerical P systems. *Acta Informatica*, 49(6):395–412.