INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 11(2):292-304, April 2016.

Numerical P Systems with Thresholds

Z. Zhang, L. Pan

Zhiqiang Zhang
Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China
zhiqiangzhang@hust.edu.cn

Linqiang Pan*
Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China
*Corresponding author: lqpan@mail.hust.edu.cn

Abstract: Numerical P systems are a class of P systems inspired both from the structure of living cells and from economics. In this work, a control of using evolution programs is introduced into numerical P systems: a threshold is considered and a program can be applied only when the values of the variables involved in the production function of the program are greater than/equal to (lower-threshold) or smaller than/equal to (upper-threshold) the threshold. The computational power of numerical P systems with lower-threshold or upper-threshold is investigated. It is proved that numerical P systems with a lower-threshold, with one membrane and linear production functions, working both in the all-parallel mode and in the one-parallel mode are universal. The result is also extended to numerical P systems with an upper-threshold, by proving the equivalence of the numerical P systems with lower- and upper-thresholds.
Keywords: membrane computing, numerical P system, computation power, universality, register machine.

1 Introduction

Membrane computing is a branch of natural computing, which is inspired from the structure and functioning of living cells. The computing devices considered in membrane computing are called *P systems*. They are parallel, distributed and non-deterministic computational models. According to their membrane structure, there are two main classes of P systems: *cell-like P systems*, with a hierarchical arrangement of membranes [7], and *tissue-like P systems* or *neural-like P systems*, with a net of processor units placed in the nodes of a directed graph [3, 5]. The present work deals with a class of cell-like P systems, called *numerical P systems* [10].

Numerical P systems are motivated by the cell structure and the economic reality. Numerical variables are placed in the regions of a membrane structure. These variables can evolve by means of programs, which are composed of two components, a *production function* and a *repartition protocol*. A production value of the region at a given step is computed by means of the production function. This value is distributed to variables from the region where the program resides, and to variables in its upper and lower neighbors according to the repartition protocol. By a synchronized use of production functions, followed by the repartition of the obtained values, a transition is defined between system configurations. The values assumed by a distinguished variable during a computation form the set of numbers computed by the system.

Many computational properties of numerical P systems have been investigated at both the theoretical level and at the application level [4,9,10,12–17]. Several strategies of using production-repartition programs were considered: sequential (at each step, in each region, only one program can be applied), all-parallel (all programs in a region of the membrane structure are used simultaneously, with each variable participating in all programs where it appears), one-parallel (the programs are chosen to be used in parallel in such a way that each variable participates in only one of the chosen programs).

Using a threshold is an interesting strategy of controlling the use of production-repartition programs. The idea was introduced in numerical P systems in [12], under the name of *enzymatic control*: a distinguished variable, called enzyme, is associated with each program and the program is applied only if the current value of the enzyme is not smaller than the smallest value of the variables involved in the production function of the program. The "enzymatic control" is useful in designing robot controllers based on numerical P systems [13–15].

We here introduce a related but different strategy, similar to the threshold control used in [18] for spiking neural P systems: rules can be used according to the result of the comparison of the number of spikes in the neuron with the constant, which corresponds to the fact that a neuron can fire when its potential is greater than or equal to its threshold. In our case, a constant is associated with the numerical P system and it is used as a control threshold in two natural ways: a program can be applied only when the values of the variables involved in the production function are not smaller than (the lower-threshold case), respectively not greater than (the upper-threshold case) the constant. The computational power of such P systems is investigated. Specifically, it is proved that universality results can be obtained for such P systems with one membrane and linear production functions, working both in the all-parallel mode and in the one-parallel mode. The proof is done (by simulating register machines) only for lower-thresholds, then the result is extended to the case of upper-threshold by proving that numerical P systems with upper-thresholds can simulate systems with lower-thresholds.

The possible usefulness of the threshold control remains to be examined for applications (in robot control). For the sake of applications, it could be useful to consider its stronger versions, such as taking different thresholds for different membranes or even for different programs in the system (maybe also mixing the way to use the thresholds, in the lower or upper ways).

2 Preliminaries

Readers are assumed to be familiar with basic elements of membrane computing, e.g., from [7, 8, 11]. Here we only mention some notions and notations which are used in this paper.

We denote by $\mathbb{N}$ the set of natural numbers, and the set of real numbers is denoted by $\mathbb{R}$. The family of all recursively enumerable sets of k-dimensional vectors of non-negative integers is denoted by $Ps(k)RE$. Since numbers can be seen as one-dimensional vectors, we can replace $Ps(1)$ by N in the notation, thus obtaining NRE.

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 labeled with the elements of the set $\{0, 1, \ldots, m\}$, 0 is the label of the first instruction to be executed, and m is the label of the halt instruction of P. Registers contain non-negative integer values. The instructions of P have the following forms:

- $j : (INC(r), k, l)$, with $0 \leq j < m, 0 \leq k, l \leq m$, and $1 \leq r \leq n$.
  This instruction, labeled with j, increments the value contained in register r, then non-deterministically jumps either to instruction k or to instruction l.

- $j : (DEC(r), k, l)$, 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 by 1 and jump to instruction k. If the value of r is zero, then jump to instruction l (without altering the content of the register).

- m: *Halt.*

A *deterministic* register machine is a register machine in which all INC instructions have the form $j : (INC(r), k, k)$; we write these instructions simply as $j : (INC(r), k)$.

A register machine M generates a set $N(M)$ of numbers in the following way: the machine starts with all registers being empty (i.e., storing the number zero); the machine applies the instruction with label 0 and continues to apply instructions as indicated by the labels (and made possible by the contents of registers); if it reaches the halt instruction, then the number present in register 1 at that time is said to be generated by M. If the computation does not halt, then no number is generated. It is known that register machines generate all sets of numbers which are Turing computable, hence they characterize NRE [6].

A register machine can also be used to compute functions. A function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ is computed by a register machine M if, when starting with n_1 to n_α in registers 1 to α, if $f(n_1, \dots, n_\alpha) = (r_1, \dots, r_\beta)$, then M halts in the final label m with registers 1 to β containing r_1 to r_β, and all other registers being empty; if $f(n_1, \dots, n_\alpha)$ is undefined, then the final label of M is never reached.

A register machine can also be used as an accepting device. A set N of numbers is accepted by a deterministic register machine M if, when starting with $x \in N$ in register 1, M halts in the final label m with all registers being empty.

The following propositions concerning the computational power of register machines are essential for the main results established in this work [1, 2, 6].

Proposition 1. *For any partial recursive function* $f\colon \mathbb{N}^\alpha \to \mathbb{N}^\beta (\alpha, \beta > 0)$*, there exists a deterministic register machine* M *with* $(max\{\alpha, \beta\} + 2)$ *registers computing f.*

Proposition 2. *For any recursively enumerable set* $N \subseteq Ps(\alpha)RE$ *of vectors of non-negative integers there exists a deterministic register machine M with* $(\alpha + 2)$ *registers accepting N.*

Proposition 3. *For any recursively enumerable set* $N \subseteq Ps(\beta)RE$ *of vectors of non-negative integers there exists a non-deterministic register machine M with* $(\beta + 2)$ *registers generating N.*

3 Numerical P Systems with Thresholds

We introduce the class of numerical P systems to be investigated in this work. The definition is general, for the computing case.

A numerical P system with a threshold is a construct

$$\Pi = (m, H, \mu, T, (Var_1, Pr_1, Var_1(0)), \dots, (Var_m, Pr_m, Var_m(0)), Var_{in}, Var_{out}),$$

where

- $m \geq 1$ is the number of membranes;
- H is an alphabet of labels for membranes in μ;
- μ is a rooted tree with q nodes labeled with the elements of H;
- T is a constant, called threshold;
- Var_i, $1 \leq i \leq m$, is the set of variables in region i;
- $Var_i(0)$, $1 \leq i \leq m$, is the set of initial values of the variables in region i;

- Pr_i, $1 \leq i \leq m$, is the set of programs in region i; each program has the form

$$F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})|_T \rightarrow c_{l,i,1}|v_{l,i,1} + \cdots + c_{l,i,l_i}|v_{l,i,l_i},$$

  where $F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})$ is the production function, and $c_{l,i,1}|v_{l,i,1} + \cdots + c_{l,i,l_i}|v_{l,i,l_i}$ is the repartition protocol of the program;

- Var_{in} and Var_{out} are the sets of input and of output variables, respectively.

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. Only polynomial functions are considered here. By using the production functions in each region, 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.

The programs are applied under the control of the threshold T, according to two strategies: bounding the values of variables from below (lower-threshold) and bounding them from above (upper-threshold).

More precisely, in the first case a program can be applied only when the current value of each variable from its production function is greater than or equal to the threshold T. Dually, in the upper-threshold case, a program can be applied only when the current value of each variable from its production function is smaller than or equal to the threshold T.

The repartition of the "production" takes place as follows. For a repartition protocol $RP_{l,i}$, variables $v_{l,i,1}, \ldots, v_{l,i,l_i}$ come from the membrane i where the program resides, the parent membrane and the children membrane. Formally, $\{v_{l,i,1}, \ldots, v_{l,i,l_i}\} \subseteq Var_i \cup Var_{par(i)} \cup (\bigcup_{ch \in Ch(i)} Var_{ch})$, where $par(i)$ is the parent of membrane i and $Ch(i)$ is the set of children of membrane i. The coefficients $c_{l,i,1}, \ldots, c_{l,i,l_i}$ are natural numbers (they may be also 0, in which case the terms "$+0|x$" are omitted), which specify the proportion of the current production value distributed to each variable $v_{l,i,1}, \ldots, v_{l,i,l_i}$. At time t, if we denote with $C_{l,i} = \sum_{s=1}^{l_i} c_{l,i,s}$ the sum of all coefficients of the repartition protocol, and denote with

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

the "unitary portion", then the value $ad_{l,i,r}(t) = q_{l,i}(t) \cdot c_{l,i,r}$ represents the value added to variable $v_{l,i,r}$. If variable $v_{l,i,r}$ appears in several repartition protocols, for example, $RP_{l_1,i_1}, \ldots, RP_{l_k,i_k}$, all these values $ad_{l_1,i_1,r}, \ldots, ad_{l_k,i_k,r}$ are added to variable $v_{l,i,r}$. After computing the production function value, the variables involved in the production function are reset to zero. So, if at time t variable $v_{l,i,r}$ is involved in at least one production function, its value at time $t+1$ is $v_{l,i,r}(t+1) = \sum_{s=1}^{k} ad_{l_s,i_s,r}(t)$; otherwise, $v_{l,i,r}(t+1) = v_{l,i,r}(t) + \sum_{s=1}^{k} ad_{l_s,i_s,r}(t)$.

Such a system evolves in the all-parallel mode (at each step, in each membrane, all programs which can be applied are applied, allowing that more than one program share the same variable) or in the one-parallel mode (apply programs in the all-parallel mode with the restriction that one variable can appear in only one of the applied programs). A configuration represents the values of all system's variables at a given computation step. Initially, the variables have the values specified by $Var_i(0), 1 \leq i \leq m$. Using the programs in the way mentioned above, a transition of the system from a configuration to the next one is defined. A sequence of such transitions forms a computation. If no program in each region can be applied, we say that the system reaches a *halting configuration*.

In this way, a numerical P system can compute a function f: $\mathbb{N}^{\alpha} \to \mathbb{N}^{\beta}(\alpha, \beta \geq 0)$: the α values of the arguments are introduced in the system as the initial values of variables in Var_{in} and the β-vector of the function value is obtained in the variables from Var_{out} in the halting configuration of the system. If the system never reaches a halting configuration, then no result is obtained.

By ignoring the input variables, (non-deterministic) numerical P systems with thresholds can also be used in the *generating mode*, whereas by ignoring the output variables we can use (deterministic or non-deterministic) numerical P systems with thresholds in the *accepting mode.*

Note that $q_{j,i}(t)$ are integers only if the value of the production functions $F_{j,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))$ is divisible by the respective sums $C_{j,i}(t)$. If at any step, all the values of the production functions are divisible by the respective sums, we associate this kind of systems with the notation *div*. If a current production is not divisible by the associated coefficients total, then we can take the following decisions [10]: (i) the remainder is lost (the production which is not immediately distributed is lost), (ii) the remainder is added to the production obtained in the next step (the non-distributed production is carried over to the next step), (iii) the system simply stops and aborts, no result is associated with that computation. We denote these three cases with *lost, carry, stop*, respectively. In this paper, the numerical P systems with thresholds that we construct are of the *div* type.

The set of natural numbers generated or accepted in the way mentioned above by a system Π is denoted by $N_{\alpha}(\Pi), \alpha \in \{gen, acc\}$. We use $N_{\alpha}T_{\gamma}P_{m}^{D}(poly^{n}(r), \beta)$ to denote the family of all sets $N_{\alpha}(\Pi)$ of numbers computed by systems Π working in α mode, with at most m membranes, production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial, using the rules in the mode $\beta \in \{all, one\}$, where *all* stands for all-parallel, and *one* stands for one-parallel, and with the threshold used in the $\gamma \in \{l, u\}$ way, with l indicating the lower-threshold case and u indicating the upper-threshold case; the letter D indicates the use of deterministic systems (we remove D when the systems may also be non-deterministic). If one of the parameters m, n, r is not bounded, then we replace it with $*$.

4 The Universality of Numerical P Systems with Lower-thresholds

In this section, we investigate the computational power of numerical P systems with lower-thresholds working in the all-parallel mode and in the one-parallel mode.

Theorem 4. *Each partial recursive function $f : \mathbb{N}^{\alpha} \to \mathbb{N}^{\beta}$ $(\alpha > 0, \beta > 0)$ can be computed by a deterministic numerical P system with a lower-threshold, with only one membrane, using linear production functions that use each at most three variables, and working in the all-parallel mode.*

Proof: Let $M = (n, P, m)$ be a deterministic register machine with n registers, computing function f. The initial instruction of M has the label 0 and the machine halts only if the instruction with label m is reached. According to Proposition 1, $n = max\{\alpha, \beta\} + 2$ is enough. Before the computation starts, let us assume that the values of the first α registers are equal to $r_1, \ldots, r_{\alpha}$. When the computation halts, the values stored in the registers $1, \ldots, \beta$ are the values computed by $f(r_1, \ldots, r_{\alpha})$.

We construct the following numerical P system with a lower-threshold:

$$\Pi_M = (1, \{0\}, [_0\ ]_0, 1, (Var_0, Pr_0, Var_0(0)), Var_{in}, Var_{out}),$$

where

- $Var_0 = \{x_{i,1}, x_{i,2}, p_j \mid 1 \leq i \leq n, 0 \leq j \leq m\}$;

- $Var_0(0)$ is the vector of initial values of the variables, with:
  - $x_{i,1} = x_{i,2} = r_i$, for all $1 \leq i \leq \alpha$;
  - $x_{i,1} = x_{i,2} = 0$, for all $1 + \alpha \leq i \leq n$;
  - $p_j = 0$, for all $0 \leq j \leq m$ with the exception of $p_0 = 1$;
- $Pr_0 = \{3p_j|_1 \to 1|x_{i,1} + 1|x_{i,2} + 1|p_k$, for all instructions $j : (INC(i), k) \in P\}$
  $\cup \{p_j|_1 \to 1|p_l,$
  $x_{i,1} - x_{i,2} - p_j|_1 \to 1|p_l,$
  $x_{i,1} - x_{i,2} + p_j|_1 \to 1|p_k,$
  $2(x_{i,1} - p_j)|_1 \to 1|x_{i,1} + 1|x_{i,2}$, for all instructions $j : (DEC(i), k, l) \in P\}$;
- $Var_{in} = \{x_{1,i}, \ldots, x_{\alpha,i} \mid 1 \leq i \leq 2\}$;
- $Var_{out} = \{x_{1,1}, \ldots, x_{\beta,1}\}$.

Note that the threshold is equal to 1. The value of register i $(1 \leq i \leq n)$ is encoded by variables $x_{i,1}$ and $x_{i,2}$. The values of $x_{i,1}$ and $x_{i,2}$ are always equal. The input values $r_1, \ldots, r_\alpha$ are set as the initial values of variables $x_{1,i}, \ldots, x_{\alpha,i}$ $1 \leq i \leq 2$, respectively. Variables $p_0, \ldots, p_m$ are used to indicate the instruction to be simulated. During the computation, the values of $p_0, \ldots, p_m$ are equal to 1 or 0 (at most one of them is equal to 1 in each step, and this indicates that the system starts to simulate the corresponding instruction of M).

The increment instruction $j : (INC(i), k)$ is simulated by the program $3p_j|_1 \to 1|x_{i,1} + 1|x_{i,2} + 1|p_k$ in one step. When $p_j = 0$, the program cannot be applied because the value of p_j is smaller than the threshold 1. When $p_j = 1$, the program can be applied since the value of p_j is equal to the threshold. After the application of this program, each of the variables $x_{i,1}, x_{i,2}, p_k$ obtains a portion 1, and variable p_j is reset to zero. Variable $p_k = 1$ indicates that the instruction labeled k will be simulated at the next step, the increment of variables $x_{i,1}, x_{i,2}$ corresponds to the increase of the number stored in register i by 1. So, the increment instruction $j : (INC(i), k)$ has been correctly simulated.

The decrement instruction $j : (DEC(i), k, l)$ is simulated in one step by the following four programs:

$$p_j|_1 \to 1|p_l, \tag{2}$$

$$x_{i,1} - x_{i,2} - p_j|_1 \to 1|p_l, \tag{3}$$

$$x_{i,1} - x_{i,2} + p_j|_1 \to 1|p_k, \tag{4}$$

$$2(x_{i,1} - p_j)|_1 \to 1|x_{i,1} + 1|x_{i,2}. \tag{5}$$

When a decrement instruction $j : (DEC(i), k, l)$ starts to be simulated, which means that $p_j = 1$, there are the following two cases.

- $p_j = 1$, $x_{i,1} = x_{i,2} = 0$. In this case, only program (2) satisfies the threshold condition. After applying program (2), variable p_j is set to zero, and variable p_l receives a contribution 1, which indicates the next instruction to be simulated. Programs (3)–(5) cannot be applied since variables $x_{i,1}$ and $x_{i,2}$ are zero (smaller than the threshold 1). Hence the values of $x_{i,1}$ and $x_{i,2}$ are not changed, and the computation continues with the simulation of instruction l of register machine M.
- $p_j = 1$, $x_{i,1} = x_{i,2} \geq 1$. In this case, all the four programs satisfy the threshold condition, thus all of them can be applied. Program (4) transfers the production value 1 to variable p_k, which indicates the next instruction k to be simulated. By using program (5), variables $x_{i,1}$

and $x_{i,2}$ are decreased; their values are first zeroed and each of them receives a contribution of their former value minus one. The role of program (3) is to cancel the effect of program (2). Program (2) transfers the value of p_j to p_l, thus p_l gets a contribution of 1, which is canceled by program (3) by sending a contribution of -1 to p_l. Hence the values of variables $x_{i,1}$ and $x_{i,2}$ are decremented by one and the next instruction to be simulated is the one labeled with k.

After the simulation of any instruction of M, the values of both variables $x_{i,1}$ and $x_{i,2}$ are equal to the contents of register i $(1 \leq i \leq n)$, while only one of variables $p_0, \ldots, p_m$ is equal to 1, indicating the next instruction of M to be simulated. When M reaches the halt instruction, the corresponding value of variable p_m is equal to 1. Since no program contains the variable p_m in the production function, Π_M reaches a final configuration; the result of the computation is the values of variables $x_{1,1}, \ldots, x_{\beta,1}$. □

According to Proposition 2, for any recursively enumerable set $N \subseteq Ps(\alpha)RE$ of vectors of non-negative integers there exists a deterministic register machine M with $(\alpha + 2)$ registers accepting N. For this register machine M, following the proof in Theorem 4, we can construct a numerical P system with a lower-threshold that accepts N. So, the following corollary holds.

Corollary 5. $Ps(\alpha)RE = N_{acc}T_lP_1^D(poly^1(3), all)$.

For numerical P systems with lower-thresholds working in the one-parallel mode, the following similar results hold.

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

Proof: We proceed like in the proof of Theorem 4, with the difference that here we simulate both deterministic and non-deterministic register machines. Let $M = (n, P, m)$ be a non-deterministic register machine with $n = \max\{\alpha, \beta\}+2$ registers, computing the function f. As usual, the input values $r_1, \ldots, r_\alpha$ are stored in the first α registers before the computation starts, with all the other registers being empty. When the computation halts, the values $f(r_1, \ldots, r_\alpha)$ will be found in registers $1, \ldots, \beta$.

The numerical P system with a lower-threshold to simulate register machine M is constructed as follows.

$$\Pi_M = (1, \{0\}, [_0\ ]_0, 1, (Var_0, Pr_0, Var_0(0)), Var_{in}, Var_{out}),$$

where

- $Var_0 = \{p_{j,g}, x_{i,g} \mid 1 \leq g \leq 5,\ 1 \leq i \leq m, 0 \leq j \leq n\}$;
- $Var_0(0)$ is the vector of initial values of the variables, with:
  - $x_{i,g} = r_i$, for all $1 \leq i \leq \alpha, 1 \leq g \leq 5$;
  - $x_{i,g} = 0$, for all $1 + \alpha \leq i \leq n, 1 \leq g \leq 5$;
  - $p_{j,g} = 0$, for all $0 \leq j \leq m, 1 \leq g \leq 5$;
  - $p_{0,g} = 1$, for all $1 \leq g \leq 5$;

- $Pr_0 = \{2\sum_{g=1}^{5} p_{j,g}|_1 \to \sum_{g=1}^{5} 1|x_{i,g} + \sum_{g=1}^{5} 1|p_{k,g},$
  $2\sum_{g=1}^{5} p_{j,g}|_1 \to \sum_{g=1}^{5} 1|x_{i,g} + \sum_{g=1}^{5} 1|p_{l,g};$
  for all instructions $j : (INC(i), k, l) \in P\}$
  $\cup \{5(x_{i,1} - p_{j,1})|_1 \to \sum_{g=1}^{5} 1|x_{i,g},$
  $8(x_{i,2} - x_{i,3} + p_{j,2})|_1 \to \sum_{g=1}^{5} 1|p_{k,g} + \sum_{g=1}^{3} 1|p_{j,g},$
  $-5(x_{i,4} - x_{i,5} + p_{j,3})|_1 \to \sum_{g=1}^{5} 1|p_{l,g},$
  $5p_{j,4}|_1 \to \sum_{g=1}^{5} 1|p_{l,g},$
  $-3p_{j,5}|_1 \to \sum_{g=1}^{3} 1|p_{j,g};$ for all instructions $j : (DEC(i), k, l) \in P\};$

- $Var_{in} = \{x_{i,g} \mid 1 \leq g \leq 5, 1 \leq i \leq \alpha\};$

- $Var_{out} = \{x_{1,1}, \ldots, x_{\beta,1}\}.$

In order to ensure that at each step only one variable can appear in the production functions of the applied programs, the value of register i ($1 \leq i \leq n$) is contained in five variables $x_{i,g} 1 \leq g \leq 5$, and the system uses five variables $p_{i,g}, 1 \leq g \leq 5$, to control the simulation of the instruction with label i of register machine M (in the following, for brevity, we use $x_{i,g}$ and $p_{i,g}$ to represent all the five variables for $1 \leq g \leq 5$, respectively). During the computation, variables $x_{i,g}$ are always equal to each other, and the same holds for variables $p_{i,g}$. The input values r_i ($1 \leq i \leq \alpha$) are introduced into the system as the initial values of variables $x_{i,g}$ ($1 \leq i \leq \alpha$), respectively. When the instruction i is simulated, all the five variables $p_{i,g}$ are equal to 1, while all the others are zero.

The simulation of an increment instruction $j : (INC(i), k, l)$ is done in one step by the following two programs:

$$2\sum_{g=1}^{5} p_{j,g}|_1 \to \sum_{g=1}^{5} 1|x_{i,g} + \sum_{g=1}^{5} 1|p_{k,g}, \tag{6}$$

$$2\sum_{g=1}^{5} p_{j,g}|_1 \to \sum_{g=1}^{5} 1|x_{i,g} + \sum_{g=1}^{5} 1|p_{l,g}. \tag{7}$$

If $p_{j,g} = 0$, then programs (6) and (7) cannot be executed since the values of variables $p_{j,g} = 0$ are smaller than the thresholds 1. If $p_{j,g} = 1$, then only one of programs (6) and (7) can be applied because their production functions share the same variables (the system works in the one-parallel mode). If program (6) (resp., program (7)) is applied, then variable $x_{i,g}$ is increased by one, setting $p_{k,g}$ (resp., $p_{l,g}$) to 1, thus the system starts to simulate instruction k (resp., instruction l), and resetting variables $p_{j,g}$ to zero. If M is deterministic, then the simulation of the instruction $j : (INC(i), k)$ is performed by using the program (6). In this case, no competition occurs between the programs, and so the simulation is deterministic.

The simulation of a decrement instruction $j : (DEC(i), k, l)$ is done in one step by the following five programs:

$$5(x_{i,1} - p_{j,1})|_1 \to \sum_{g=1}^{5} 1|x_{i,g}, \tag{8}$$

$$8(x_{i,2} - x_{i,3} + p_{j,2})|_1 \to \sum_{g=1}^{5} 1|p_{k,g} + \sum_{g=1}^{3} 1|p_{j,g}, \tag{9}$$

$$-5(x_{i,4} - x_{i,5} + p_{j,3})|_1 \to \sum_{g=1}^{5} 1|p_{l,g} \tag{10}$$

$$5p_{j,4}|_1 \to \sum_{g=1}^{5} 1|p_{l,g}, \tag{11}$$

$$-3p_{j,5}|_1 \to \sum_{g=1}^{3} 1|p_{j,g}. \tag{12}$$

If $p_{j,g} = 0$, then programs (8)–(12) cannot be applied, because $p_{j,g} = 0$ are smaller than the threshold. So, when $p_{j,g} = 0$, no undesirable simulation steps can appear.

If $p_{j,g} = 1, x_{i,g} = 0$, then the values of $x_{i,g}$ should remain unchanged, and the computation should jump to the simulation of instruction l, which is realized by programs (11) and (12) in one step. (Note that in this case programs (8) – (10) cannot be applied, for the values of $x_{i,g}$ are smaller than the thresholds.) The effect of program (11) is to reset variables $p_{j,4}$ to zero and to give a contribution 1 to each of variables $p_{l,g}$, whose values are 1 after the application of this program, thus correctly simulating the passing to instruction l. At the same time, program (12) is applied, the role of which is to set all the variables $p_{j,g}$ ($g \neq 4$) to zero. Variable $p_{j,5}$ appears in the production function, so its initial value is canceled, and it receives no contribution, hence its final value is zero. For variables $p_{j,1}$, $p_{j,2}$ and $p_{j,3}$, their initial values are 1 and receive contribution -1, hence their final values are zero, which is also correct.

If $p_{j,g} = 1, x_{i,g} \geq 1$, then the values of $x_{i,g}$ should be decremented and the computation should proceed to the simulation of instruction k. In this case, all the five programs (8)–(12) can be applied. Programs (8) and (9) decrement the values of $x_{i,g}$ and increment the values of $p_{k,g}$ by 1. The other programs have auxiliary roles. Note that all the variables $p_{j,g}$ and $x_{i,g}$ appear in the production functions of programs (8)–(12), so their values are first reset to zero, and their final values will be the sum of all the contributions they receive. Variables $x_{i,g}$ only appear in the repartition protocol of program (8), which gives a contribution of their initial value minus 1, thus correctly decrementing their values by one. Variables $p_{j,1}$, $p_{j,2}$ and $p_{j,3}$ receive a contribution 1 from program (9) and a contribution -1 from program (12), thus their values will be equal to 0. Variables $p_{j,4}$ and $p_{j,5}$ do not appear in any repartition protocol of programs (8) – (12), thus their final values are zero. The role of program (10) is to cancel the effect of program (11). Program (11) sends a contribution 1, and simultaneously program (10) sends a contribution -1, to each of variables $p_{l,g}$, whose final values are hence equal to 0. Each of variables $p_{j,g}$ receives a contribution 1, thus their final values are 1, which is also correct.

After the simulation of each instruction of M, all the variables $x_{i,g}$ are equal to the contents of register i ($1 \leq i \leq n$), while the variables $p_{j,g}$ ($0 \leq j \leq m$) correctly indicate the next instruction of M to be simulated. When the program counter of M reaches the value k, the corresponding values of variables $p_{k,g}$ is equal to 1. When the program counter of M reaches the value m, the corresponding values of variables $p_{m,g}$ are equal to 1. Since no program contains variables $p_{m,g}$ in the production function, Π_M reaches a halting configuration; the result of the computation is values of variables $x_{1,1}, \ldots, x_{\beta,1}$. $\square$

According to Propositions 2 and 3, for any recursively enumerable set $N \subseteq Ps(\alpha)RE$ of vectors of non-negative integers there exists a deterministic (or non-deterministic) register machine M with $(\alpha+2)$ registers accepting (generating, respectively) N. For this register machine M, following the proof in Theorem 6, we can construct a deterministic (or non-deterministic) numerical P system with a lower-threshold that accepts (or generates, respectively) N.

Corollary 7. $Ps(\alpha)RE = N_{gen}T_lP_1(poly^1(5), one) = N_{acc}T_lP_1^D(poly^1(5), one)$.

In conclusion, we obtain the following characterizations of *NRE*.

Theorem 8. $NRE = N_{acc}T_lP_1^D(poly^1(3), all) = N_{gen}T_lP_1(poly^1(5), one) =$
$= N_{acc}T_lP_1^D(poly^1(5), one)$.

Proof: The first equation can be obtained according to Corollary 5, where $\alpha = 1$. Similarly, letting $\alpha = 1$ in Corollary 7, we can obtain the last two equations. □

5 The Universality of Numerical P Systems with Upper-thresholds

In this section we prove that the computational power of numerical P systems with upper-thresholds (for short, UTNP systems) is equivalent with that of numerical P systems with lower-thresholds (for short, LTNP systems).

Lemma 9. *For any numerical P system with a lower-threshold* Π_l*, there is a P system* Π_u *with an upper-threshold, with the same variables, such that the corresponding variables of* Π_l *and* Π_u *have equal values but of opposite sign.*

Proof: Let Π_l be a numerical P system with a lower-threshold of the form considered in the previous sections. We construct a numerical P system with an upper-threshold Π_u in the following way. Π_u has the same membrane structure as Π_l. In the same membrane, the two systems have the same variables. The initial values of variables in Π_u is the same as in Π_l multiplied with -1. Similarly, for the thresholds of the two systems (they are equal, but of opposite signs). For a program

$$f_l(x_1, \dots, x_i)|_T \to c_1|v_1 + \cdots + c_l|v_l$$

in Π_l, we introduce in Π_u the program

$$f_u(x_1, \dots, x_i)|_{-T} \to c_1|v_1 + \cdots + c_l|v_l,$$

where $f_u(x_1, \dots, x_i)$ is constructed as follows:

- If the production function $f_l(x_1, x_2, \dots, x_n)$ is an odd function, that is,

$$f_l(-x_1, -x_2, \dots, -x_n) = -f_l(x_1, x_2, \dots, x_n),$$

  then $f_u(x_1, x_2, \dots, x_n) = f_l(x_1, x_2, \dots, x_n)$;

- If the production function $f_l(x_1, x_2, \dots, x_n)$ is an even function, that is,

$$f_l(-x_1, -x_2, \dots, -x_n) = f_l(x_1, x_2, \dots, x_n),$$

  then $f_u(x_1, x_2, \dots, x_n) = -f_l(x_1, x_2, \dots, x_n)$;

- If the production function $f_l(x_1, x_2, \ldots, x_n)$ is neither an even function nor an odd function, then it can be expressed as the addition of an even function with an odd function, that is,

$$\begin{aligned} f_l(x_1, x_2, \ldots, x_n) = & \frac{f_l(x_1, x_2, \ldots, x_n) + f_l(-x_1, -x_2, \ldots, -x_n)}{2} \\ & + \frac{f_l(x_1, x_2, \ldots, x_n) - f_l(-x_1, -x_2, \ldots, -x_n)}{2}, \end{aligned}$$

  and then

$$\begin{aligned} f_u(x_1, x_2, \ldots, x_n) = & - \frac{f_l(x_1, x_2, \ldots, x_n) + f_l(-x_1, -x_2, \ldots, -x_n)}{2} \\ & + \frac{f_l(x_1, x_2, \ldots, x_n) - f_l(-x_1, -x_2, \ldots, -x_n)}{2} \\ & = -f_l(-x_1, -x_2, \ldots, -x_n). \end{aligned}$$

Based on the previous construction of the UTNP system Π_u, we can get that, if the two systems are deterministic, working in the all-parallel mode, then at any step, the program in Π_l and its corresponding program in Π_u can simultaneously be applied or cannot be applied, and the two production functions have equal values but of opposite signs. Thus at any step the variable in the two systems get equal contributions but of opposite signs; this is true also for the initial values, hence always the values of variables are equal but of opposite signs. The systems halt simultaneously.

In conclusion, no matter whether Π_l, Π_u work in computing mode or in generating mode, this lemma holds true. □

Corollary 10. *$Ps(\alpha)RE = N_{acc}T_uP_1^D(poly^1(3), all)$.*

Proof: According to Proposition 2, for any recursively enumerable set $N \subseteq Ps(\alpha)RE$ of vectors of non-negative integers there exists a deterministic register machine M with $(\alpha + 2)$ registers accepting N. For this register machine M, following the proof in Theorem 4, we can construct a numerical P system with a lower-threshold Π_M that accepts N.

For Π_M, according to Lemma 9, we can construct an UTNP system Π_u with "contrary" configurations (equal values of variables, but of opposite signs). Now we add the programs

$$1 + p_m - x_{i,1}|_{-1} \to 1|x_{i,1},\ 1 \leq i \leq \beta. \tag{13}$$

to Π_u thus obtaining a new UTNP system Π'_u. The initial value of p_m is 0, hence programs (13) cannot be applied. As long as $p_m = 0$, there is no difference between the functioning of Π'_u and Π_u. When p_m is equal to -1, Π_u reaches a halt configuration, while Π'_u continues executing program (13). The effect of programs (13) is transforming the variables $x_{i,1} \leq -1$ to their contrary. Thus the system Π'_u has the same output as Π_M. □

In a similar way to the proof of Corollary 7, we can prove the following corollary.

Corollary 11. *$Ps(\alpha)RE = N_{gen}T_uP_1(poly^1(5), one) = N_{acc}T_uP_1^D(poly^1(5), one)$.*

If we set $\alpha = 1$ in Corollary 10 and Corollary 11, then we can get the following characterizations.

Theorem 12. *$NRE = N_{acc}T_uP_1^D(poly^1(3), all) = N_{gen}T_uP_1(poly^1(5), one) =$
$= N_{acc}T_uP_1^D(poly^1(5), one)$.*

6 Conclusions and Discussions

In this work, we have introduced thresholds into numerical P systems, and the computational power of such P systems has been investigated. Specifically, we proved that universality can be obtained for such P systems with one membrane and linear production functions working both in the all-parallel mode and in the one-parallel mode.

The rules of numerical P systems with thresholds constructed in Section 4 are applied in the all-parallel mode and in the one-parallel mode, respectively. It remains open what the computational power of numerical P systems with thresholds working in the sequential mode is.

In this work, the polynomial functions used in numerical P systems with the two kinds of thresholds have at most 3 variables for all-parallel systems and 5 variables for one-parallel systems. It is a natural question whether the number of variables can be decremented.

The thresholds are used in the sense of lower-bounds and upper-bounds. Other ways to use the thresholds could be of interest, for example, applying a program only if the values of all (or part of) the variables are strictly greater (or smaller) than the threshold.

Numerical P systems and enzymatic numerical P systems have already been used in robot control [4, 10, 16, 17]. It remains to check whether numerical P systems with thresholds are also useful for such applications.

Acknowledgements

This work was supported by National Natural Science Foundation of China (61320106005 and 61472154), Ph.D. Programs Foundation of Ministry of Education of China (2012014213008), Natural Science Foundation of Hubei Province (2011CDA027), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

Bibliography

[1] Freund, R.; Oswald, M. (2002); GP Systems with Forbidding Context. *Fundamenta Informaticae* 49(1-3), 81–102.

[2] Freund, R.; Păun, G. (2001); On the Number of Non-Terminal Symbols in Graph-Controlled, Programmed and Matrix Grammars. In: *Machines, Computations, and Universality*, 3rd Internat. Conf., MCU, Lecture Notes in Computer Science, vol. 2055, Springer, Berlin, 214–225.

[3] Ionescu, M.; Păun, G., Yokomori; T. (2006); Spiking Neural P Systems. *Fundamenta Informaticae* 71(2-3), 279–308.

[4] Leporati, A.; Porreca, A.E.; Zandron, C.; Mauri, G. (2013); Improving Universality Results on Parallel Enzymatic Numerical P Systems. *Proc. 11th Brainstorming Week on Membrane Computing*, Sevilla, 4–8.

[5] Martín-Vide, C.; Pazos, J.; Păun, Gh.; Rodriguez-Paton, A. (2003); Tissue P Systems. *Theoretical Computer Science* 296(2), 295–326.

[6] Minsky, M.L. (1967); *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[7] Păun, G. (2000); Computing with Membranes. *Journal of Computer and System Sciences* 61(1), 108–143.

[8] Păun, G. (2002); *Membrane Computing–An Introduction.* Springer-Verlag, Berlin.

[9] Păun, G. (2013); Some Open Problems about Numerical P Systems. *Proc. 11th Brainstorming Week on Membrane Computing*, Sevilla, 245–252.

[10] Păun, G.; Păun, R. (2006); Membrane Computing and Economics: Numerical P Systems. *Fundamenta Informaticae*, 73(1), 213–227.

[11] Păun, G.;Rozenberg, G.; Salomaa A.(eds.)(2010); *The Oxford Handbook of Membrane Computing.* Oxford University Press, New York.

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

[13] Pavel, A.B.; Buiu, C. (2012); Using Enzymatic Numerical P Systems for Modeling Mobile Robot Controllers. *Natural Computing* 11(3), 387–393.

[14] Pavel, A.B.; Vasile, C.I.; Dumitrache, I. (2012); Robot Localization Implemented with Enzymatic Numerical P Systems. In: *Biomimetic and Biohybrid Systems*, Springer, 204–215.

[15] Pavel, A.B.; Vasile, C.I.; Dumitrache, I. (2013); Membrane Computing in Robotics. In: *Beyond Artificial Intelligence*, Springer, Berlin, 125–135.

[16] Vasile, C.I.; Pavel, A.B.; Dumitrache, I. (2013); Universality of Enzymatic Numerical P Systems. *International Journal of Computer Mathematics* 90(4), 869–879.

[17] Vasile, C.I.; Pavel, A.B.; Dumitrache, I.; Păun, G. (2012); On the Power of Enzymatic Numerical P Systems. *Acta Informatica* 49(6), 395–412.

[18] Wang, J.; Hoogeboom, H.J.; Pan, L.; Păun, G.; Pérez-Jiménez, M.J. (2010); Spiking Neural P Systems with Weights. *Neural Computation* 22(10), 2615–2646.