ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **18**, Number 3, 2015, 273–295

On String Languages Generated by Numerical P Systems

Zhiqiang ZHANG[1], Tingfang WU[1], Linqiang PAN[1*], Gheorghe PĂUN[2]

[1]Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
E-mail: lqpan@mail.hust.edu.cn

[2]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania

Abstract. Numerical P systems are a class of P systems inspired both from the structure of living cells and from economics. Usually, they are used as devices for computing sets of numbers. In this work, we consider numerical P systems as language generators: symbols are associated with numbers (portions of the "production value") sent to the environment. The generative capacity of such systems is investigated, both for the standard and the enzymatic form of numerical P systems, having as a reference the families of languages in the Chomsky hierarchy.

Key-words: bio-inspired computing, membrane computing, numerical P system, Chomsky hierarchy, universality.

1. Introduction

Numerical P systems [11] occupy a somewhat "exotic" position in membrane computing, as they take from the cell biology only the membrane architecture, but they process numerical variables in the compartments defined by membranes. More precisely, these variables are subject of certain *evolution programs* inspired from economics: a *production function* computes a *production value* based on the current

*Corresponding author

values of variables and then a *repartition protocol* distributes this value among variables in the region where the program resides and variables from the directly upper and lower regions. In this way, the values of variables change continuously. The values assumed by a distinguished variable is the set of numbers computed by the system. Characterizations of Turing computable sets of numbers are obtained in this way for rather restricted forms of programs (*e.g.*, for polynomials of low degrees as production functions). A special class of numerical P systems is that of *enzymatic ones*, where a control on the use of programs is considered; this class was proved to be useful in designing robot controllers, *e.g.*, in [7–9, 16].

In this work, we consider another way to use a numerical P system, explored for most classes of devices investigated in membrane computing, but not yet for numerical P systems, namely, as language generators. We follow the style of spiking neural P systems as language generators, see [1, 2]: portions of the production values are also sent out of the system and with value i sent out we associate a symbol b_i. In this way, a string can be associated with a computation, hence a language can be associated with a numerical P system.

Of course, many technical details appear in this framework: we should work with a finite alphabet, hence negative values or "too large" values should be prevented (we choose to abort the computation in such cases); we need to define the end of the string, the step when its last symbol is produced; we have to decide what we do in a step when no value is sent out (we will distinguish this case from the case when value zero is sent out). The technical details will be fixed and explained in Section 3.

We examine here languages generated by non-enzymatic, by enzymatic, and by purely enzymatic (all programs are enzymatic) numerical P systems, working in the so-called *one-parallel* mode (in each compartment, programs are used in parallel, but with the restriction that no variable appears in more than one production function). The obtained families are compared especially with families *FIN, REG, CF, RE*, of finite, regular, context-free, and recursively enumerable languages.

Many research directions remain open – several are formulated along the paper, while further research topics are added in Section 8.

2. Prerequisites

Readers are assumed to be familiar with basic elements of membrane computing, *e.g.*, from [10,12], and formal language theory, as available in many monographs (*e.g.*, in [13]). We mention here only a few notions and notations which are used in what follows.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$.

A Chomsky grammar is given in the form $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the finite set of rules. For regular grammars, the rules are of the forms $A \to aB$, $A \to a$, for some $A, B \in N$, $a \in T$.

We denote by *FIN, REG, CF, CS, REC, RE* the families of finite, regular, context-free, context-sensitive, recursive (with a decidable membership), and recursively enumerable languages. The family of Turing computable sets of numbers is denoted by *NRE* (these sets are length sets of RE languages, hence the notation).

Let $V = \{b_1, b_2, \ldots, b_q\}$ be an alphabet, for some $q \geq 1$. For a string $x \in V^*$, let us denote by $val_q(x)$ the value in (base q)+1 of x (we use (base q)+1 in order to consider the symbols $b_1, \ldots, b_q$ as digits $1, 2, \ldots, q$, thus avoiding the digit 0 in the left hand of the string). We extend this notation in the natural way to sets of strings.

A register machine is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label, and I is the set of instructions; each element of H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

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

A register machine is *deterministic* if the ADD instructions have the form l_i : (ADD$(r), l_k, l_k$); we write these instructions simply as l_i : (ADD$(r), l_k$).

A register machine M generates a number in the following way. The machine starts with all registers empty (i.e., storing the number zero). It applies the instruction with label l_0 and proceeds to apply instructions as indicated by labels (and, in the case of SUB instructions, by the contents of registers). If the register machine reaches the halt instruction, then the number n stored at that time in the first register is said to be computed/generated by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE [5].

A register machine can also be used in the accepting mode: one starts with all registers empty, except of a specified register, the input one, where a number x is introduced; the computation starts (with instruction with label l_0) and, if it halts, then the number x is accepted. In this way, again all sets of numbers from NRE are characterized (even by deterministic register machines).

Convention: when comparing the power of two language generating or accepting devices the empty string λ is ignored.

3. Definition of Numerical P Systems

We introduce now numerical P systems (we also use the abbreviation NP systems) in the form we investigate in this paper, *i.e.*, as string generators.

Such a system is a construct

$$\Pi = (m, H, \mu, (V_1, Pr_1, V_1(0)), \ldots, (V_m, Pr_m, V_m(0)), q),$$

where:

- $m \geq 1$ is the number of membranes;
- H is an alphabet (of labels for membranes in μ);
- μ is a hierarchical (cell-like) membrane structure with m membranes labeled with the elements of H;
- $V_i = \{x_{j,i} \mid 1 \leq j \leq k_i\}$, $1 \leq i \leq m$, is the set of variables in region i;
- $V_i(0)$, $1 \leq i \leq m$, is a vector which indicates the initial values of the variables in region i;
- Pr_i, $1 \leq i \leq m$, is the finite set of programs in region i; each program has the following form:

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

  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;
- $q \geq 1$ is a constant.

The programs allow the system to evolve the values of variables during computations. A 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. Here only polynomial functions with integer coefficients are considered as production functions. Of course, not all variables from V_i need to effectively appear in the functions $F_{l,i}$.

Initially, the variables have the values specified by $V_i(0), 1 \leq i \leq m$. By 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.

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 membranes. 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 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, $v_{l,i,r}(t+1) = v_{l,i,r}(t) + \sum_{s=1}^{k} ad_{l_s,i_s,r}(t)$.

Note that in equation (1), $q_{l,i}(t)$ is an integer only if the value of the production function $F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))$ is divisible by the respective sum $C_{l,i}$. If at any step, all the values of the production functions are divisible by the respective sums, then 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:

1. the remainder is lost (the production which is not immediately distributed is lost),
2. the remainder is added to the production obtained in the next step (the non-distributed production is carried over to the next step),
3. the system simply stops and aborts, no result is associated with that computation [11]. We denote these three cases with *lost, carry, stop*, respectively. In this paper, the systems we construct are of the *div* type.

In [6], a control was introduced in enzymatic numerical P systems: a distinguished variable, called *enzyme*, is associated with a program and the program can be applied only when the current value of the enzyme is greater than the smallest value of the variables involved in the production function of the program. Formally, an enzymatic program has the form

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

where $e_{l,i}$ is a variable from V_i different from variables $x_{1,i}, \ldots, x_{k_i,i}$ which are present in $F_{l,i}$ and from $v_{l,i,1}, \ldots, v_{l,i,l_i}$. Such a program may be applied at time t only if $e_{l,i}(t) > min\{x_{j,i}(t) \mid x_{j,i} \text{ appears in } F_{l,i}\}$. Note that $e_{l,i}(t)$ remains unchanged in the program where it appears as an enzymatic variable; in other programs, $e_{l,i}$ can appear as a usual variable in production functions or in repartition protocols, hence it can be "consumed" or it can receive "contributions".

If every program is enzymatic, then we call the system *purely enzymatic*.

Numerical P systems can evolve in the *all-parallel* mode (at each step, in each membrane, all applicable programs are applied, allowing that more than one program share the same variable), in the *sequential* mode (at each step, only one program is applied in each membrane; if more than one program in a membrane can be used, then one of them is non-deterministically chosen), 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; in the case of multiple choices, the programs to apply are chosen in a non-deterministic way). In this paper, we only consider the one-parallel mode.

Using the programs in the way mentioned above, we obtain transitions among configurations. A sequence of such transitions forms a computation. With a computation we can associate a result in the form of a set of numbers by designating a variable as the output one and collecting the (positive) values taken by this variable along the computation.

In this paper, we use NP systems as string generators. To this aim, we associate a string with a computation in a way somewhat similar to that adopted for spiking neural P systems [3] when used as string generators [1,2]. In spiking neural P systems, with i spikes sent out one associates a symbol b_i. We proceed in a similar way: we just consider a special variable *out* in the environment which can appear in the repartition protocol of programs in the skin region of a numerical P system. At each step its value is first reset to zero, then it receives a new value. If at one step it receives several values from several programs, all these values are added up and the sum is the value it receives at this step. If the value is a number i between 1 to q, then the symbol b_i is added to the generated string. If at any step variable *out* receives a value which is greater than q or smaller than 0, then this computation aborts, no result is associated with it.

In order to define the generated string, we need to define its end. This is clear in the case when the computation halts (no further program can be applied), and this can be taken as a definition of successful computations in purely enzymatic P systems. In non-enzymatic and in (non-purely) enzymatic systems the computations never halt, and then we define the end of the string by means of a *signal*: the step when the system sends out value 0. Because for purely enzymatic systems we have halting at out disposal, in this paper we avoid sending out value 0, that is, this case is simply ignored. (For a general definition, however, a decision should be made also for value 0 sent out. A possibility is to proceed as in spiking neural P systems, where in such a case a special symbol, b_0, is added to the string. This case remains to be investigated for NP systems.)

It still remains a case not covered: the steps when the system sends no value to variable *out*. We have two choices: to forbid such steps, by the definition of correct computations, or to proceed as in the case of spiking neural P systems and to associate the string λ to the generated string (the string is not increased, the system can continue working).

In this way, we define two languages generated by a numerical P system Π. If at each step a positive value is sent to variable *out* (with the exception of the last step, for non-enzymatic and for enzymatic P systems, when value 0 is sent out, marking the end), then we denote the generated language by $L^{res}(\Pi)$ (with res coming from *restricted*). If in the steps when no value is sent out (neither 0) we interpret that the system adds λ to the generated string, then the generated language is denoted by $L^{\lambda}(\Pi)$.

To easily remember, we list the previous conventions/definitions in Table 1.

We denote by $L^{\alpha}\beta NP_m^{\gamma}(poly^n(r), Var_{k_1}, Pro_{k_2})$, $\alpha \in \{res, \lambda\}$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$, the family of languages $L^{\alpha}(\Pi)$, generated by β numerical P systems Π (E = enzymatic, pE = purely enzymatic; if the system is non-enzymatic, then β is omitted) with at most m membranes, at most k_1 variables, and at most k_2 programs,

with production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial; the superscript $\gamma = hal$ is used for purely enzymatic systems, to indicate that the result is obtained when the system reaches a halting configuration; in the case when the end of the computation is defined by means of a signal (sending value 0 out), then we replace *hal* by *fin*. If one of the parameters m, n, r, k_1, k_2 is not bounded, then we replace it with $*$.

Table 1. Conventions and Definitions

Sending out	Non-enzymatic & Enzymatic	Purely enzymatic
$1, 2, \ldots, q$	b_i	b_i
< 0 or $> q$	abort	abort
0	end signal	ignored here
nothing	λ or forbidden (*res*)	λ or forbidden (*res*)

4. Some Examples

In order to illustrate the way in which the numerical P systems generate languages, we discuss a few examples which will also be useful later.

Example 1. Let us consider the numerical P system

$$\Pi = (1, \{0\}, [\,]_1, (\{x\}, \{2x \to 1|x+1|out,\ x \to 1|out\}, (1)), 1).$$

The system works in the one-parallel mode, hence in each step only one of the two programs can be applied. Initially, variable x has value 1. Program $2x \to 1|x+1|out$ keeps the value of x unchanged and sends out 1. After $n \geq 0$ steps when the program $2x \to 1|x+1|out$ is used, once the program $x \to 1|out$ is applied, the value of x becomes zero, hence in the next step value 0 is sent out and the computation ends. Therefore, this system generates the infinite language $L^{res}(\Pi) = \{b_1^n \mid n \geq 1\}$.

We consider now a slightly more complex system, still non-enzymatic.

Example 2. Let us examine the numerical P system Π from Figure 1. It generates the non-regular language $L^{res}(\Pi) = \{b_2\} \cup \{b_1^n b_2^{n+1} \mid n \geq 1\}$.

Initially, if the program $2(x_1 + x_2 + y_2 - y_1) \to 1|out$ is applied, which sends out a copy of 2 and zeros all the variables of the system, hence the computation ends. Then the generated language is $\{b_2\}$.

If at the first step, we start with one of the programs $3x_1 \to 1|x_1+1|y_1+1|out$ and $3x_1 \to 1|x_2+1|y_1+1|out$, then the generated languages is $\{b_1^n b_2^{n+1} \mid n \geq 1\}$. Program $3x_1 \to 1|x_1+1|y_1+1|out$ keeps variable x_1 unchanged, increases variable y_1 by 1, and sends out 1. Suppose this program is applied for $n-1$ steps, for some $n \geq 1$. Then y_1 is increased to $n-1$ and 1 is sent out for $n-1$ times. In the next step program $3x_1 \to 1|x_2+1|y_1+1|out$ is applied, which passes the value of variable x_1 to x_2 and sets variable x_1 to zero, simultaneously increases y_1, and sends out 1 once more. Variable x_2 equal to 1 triggers the process of generating b_2. When $x_2 = 1$, program

$4x_2 \to 1|x_2 + 1|y_2 + 2|out$ increases the value of y_2 and sends out 2, simultaneously keeping the value of x_2 unchanged. After n steps, variable y_2 is increases to n, which equals variable y_1. The next step, program $2(x_1 + x_2 + y_2 - y_1) \to 1|out$ is applied, which sets the variables x_1, x_2, y_1, and y_2 to zero, simultaneously sends out 2 for one more time. All variables are equal to zero, so zero is sent out, hence the string is completed.

Note that program $2(x_1 + x_2 + y_2 - y_1) \to 1|out$ is used to check whether variables y_2 and y_1 are equal. If $y_1 \neq y_2$, then the value sent out is larger than 3 or smaller than 0, which leads to the computation aborting. Only when the value of variable y_1 equals that of y_2 and x_1 or x_2 equals 1, the computation is correctly completed.

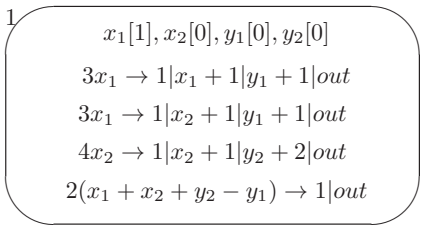

Fig. 1. The numerical P system from Example 2.

We end this section with an example of a purely enzymatic numerical P system.

Example 3. The purely enzymatic NP system Π from Figure 2 generates the non-regular language $L^{res}(\Pi) = \{b_1^n b_2^{n+1} \mid n \geq 1\}$. Initially, variable e_1 equals 1, which activates programs $x_1 + 2|_{e_1} \to 1|y_1 + 1|out$ and $x_1 + 3|_{e_1} \to 1|x_1 + 1|e_2 + 1|out$. As these two programs share the variable x_1, only one of them can be applied at any step (because of the one-parallel mode). If the first program is chosen, then in the next step the two programs can still be applied. Once the second program is applied, the value of variable x_1 will be increased, which disables the two programs application in the next step.

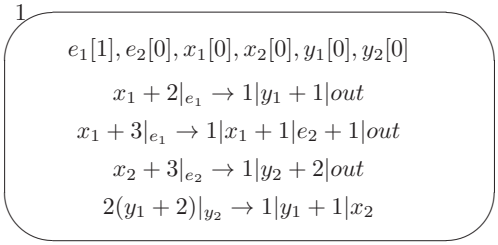

Fig. 2. The numerical P system from Example 3.

Suppose the first program is applied for $n - 1$ steps, for some $n \geq 1$; variable y_1 will increase to $n - 1$ and simultaneously the program sends out 1 for $n - 1$ times. Then the second program is applied, which sends out 1 for one more time and sends a

contribution 1 to variable e_2. In the next step e_2 equals 1, which triggers the process of generating symbol b_2. When $e_2 = 1$, program $x_2 + 3|_{e_2} \to 1|y_2 + 2|out$ is activated, which sends out value 2 and increases y_2 by 1. After n steps, the value of y_2 is increased to n, which activates the program $2(y_1 + 2)|_{y_2} \to 1|y_1 + 1|x_2$. After the application of this program, the values of x_2 and y_1 are equal to $n+1$, which disables the programs $x_2 + 3|_{e_2} \to 1|y_2 + 2|out$ and $2(y_1 + 2)|_{y_2} \to 1|y_1 + 1|x_2$. Thus in the next step no program can be applied, the computation ends. One can see that at each step there is a positive value sent out, hence the *res* condition is observed.

5. Preliminary Observations

Let us start with the observation that deterministic systems generate at most a singleton string: the computation either aborts or never halts, or it ends correctly and then only one string is generated. For instance, systems with only one program in each region, working in any mode, as well as systems working in the all-parallel mode, are deterministic. That is why the all-parallel mode is of no interest from the point of view of generating strings (it might be of interest for generating infinite sequences, but this is out of the scope of this paper).

Let us recall the "flattening lemma" from [4]: for any enzymatic (the assertion holds also for non-enzymatic and for purely enzymatic systems) with several membranes working in the one-parallel mode (the same for the all-parallel mode), there is an equivalent one-membrane system, of the same type, working in the same mode. The lemma can be extended also to systems generating strings. Actually, in all proofs which follows we only use systems with only one membrane.

Directly from definitions, it is obvious that parameters m, n, r, k_1, k_2 in $L^{\alpha}\beta NP_m^{\gamma}(poly^n(r), Var_{k_1}, Pro_{k_2})$ induce hierarchies (higher the values of parameters, larger the families), and that non-enzymatic and purely enzymatic systems are particular cases of enzymatic systems. However, only the results about non-enzymatic systems extend automatically to enzymatic systems, not also those about purely enzymatic systems, because of the halting, which is possible in purely enzymatic systems and not in enzymatic systems. In their turn, non-enzymatic and purely enzymatic systems look "non-comparable" (no direct inclusion holds between the associated families of languages).

For families which characterize RE, some of the hierarchies on m, n, r, k_1, k_2 collapse, but for families which are not universal (several will be identified in Theorem 1 below) the problem remains open whether or not these hierarchies are infinite.

The fact that the strings in the languages of the form $L^{res}(\Pi)$ are obtained in the end of computations of a length equal to the length of the string or at most with one additional step (in the case of marking the end of the computation by sending value 0 out) implies the fact that such languages are recursive: taking a string w, we construct all computations of type *res* with respect to a given system Π (their set is finite) and in this way we can check whether w can be generated by Π. Due to its relevance, we write this observation as a theorem:

Theorem 1. $L^{res}\beta NP_*^{\gamma}(poly^n(*), Var_*, Pro_*) \subseteq REC$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$.

This shows that we cannot have universality for such systems – the problem remains open to compare the respective families with families in Chomsky hierarchy other than *RE*. We will (preliminarily) do it for *FIN, REG, CF*, while the case of CS remains open.

6. The Power of Non-Enzymatic Numerical P Systems

In all considerations below, we work with the alphabet $V = \{b_1, b_2, \ldots, b_q\}$, for some $q \geq 1$. By a simple renaming of symbols, we may assume that any given language L over an alphabet with at most q symbols is a language over V.

6.1. The *res* Case

We start by considering non-enzymatic numerical P systems as language generators, working in the restricted case, when the system sends a positive value out in each computation step.

From Example 1, we get the following result (in view of the first observation from the previous section, it is relevant to note that we have two programs – all other parameters are minimal).

Theorem 2. $L^{res}NP_1^{fin}(poly^1(1), Var_1, Pro_2) - FIN \neq \emptyset$.

If the numbers of variables and of programs are not bounded, then all regular languages can be generated:

Theorem 3. $REG \subseteq L^{res}NP_1^{fin}(poly^1(1), Var_*, Pro_*)$.

Proof. For $L \in REG$, consider a regular grammar $G = (N, V, S, P)$ such that $L = L(G)$, where $N = \{A_1, A_2, \ldots, A_n\}, n \geq 1, S = A_1$, and the rules in P are of the forms $A_i \to b_k A_j, A_i \to b_k, 1 \leq i, j \leq n, 1 \leq k \leq q$. Without any lose of the generality, in rules as above we suppose $i \neq j$.

Then L can be generated by the numerical P system represented in Figure 3.

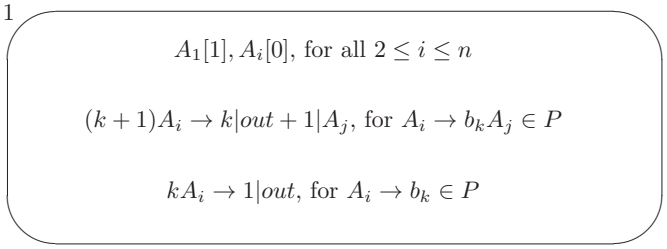

Fig. 3. The numerical P system from the proof of Theorem 3.

In each moment, only one of variables A_i $(1 \leq i \leq n)$ is equal to 1, indicating the rule to be simulated in this step.

For each rule $A_i \to b_k A_j$ in P, there is a program $(k+1)A_i \to k|out+1|A_j$ in the numerical P system. When $A_i = 1$, this program sends out number k, passes the value of variable A_i to A_j, and resets A_i to zero, hence correctly simulates the rule $A_i \to b_k A_j \in P$. For each rule $A_i \to b_k$ in P, there is a program $kA_i \to 1|out$. When A_i equals 1, this program sends out k and sets variable A_i to zero. After that, all variables are zero, hence the system sends out zero and the computation is completed. □

We do not know whether for 1, 2, 3 variables or programs, as well as for at most three variables in each program the previous inclusion is proper, but for 4 variables in the system, at most 4 in each program, and for 4 programs this is true, as a consequence of Example 2, hence we have:

Theorem 4. $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_4) - REG \neq \emptyset$.

Actually, when increasing the number of variables and of programs, more complex languages can be still generated:

Theorem 5. $L^{res}NP_1^{fin}(poly^1(4), Var_7, Pro_6) - CF \neq \emptyset$.

Proof. Following a similar idea as that in Example 2, we can construct the system Π represented in Figure 4 which generates the non-context-free language $L^{res} = \{b_2b_3\} \cup \{b_1^n b_2^{n+1} b_3^{n+1} \mid n \geq 1\}$.

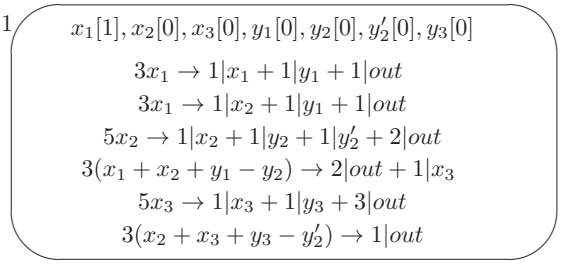

Fig. 4. The numerical P system from the proof of Theorem 5.

Initially, if the program $3(x_1+x_2+y_1-y_2) \to 2|out+1|x_3$ is applied, which sends out 1 copy of 2, zeros the variables x_1, x_2, y_1 and y_2, and increases x_3 by 1. Then program $3(x_2+x_3+y_3-y_2') \to 1|out$ is applied, which sends out 1 copy of 3, zeros variables x_2, x_3, y_3 and y_2'. Hence all the values are zeroed, the computation ends. Then the generated language is $\{b_2b_3\}$.

If at the first step, program $3x_1 \to 1|x_1+1|y_1+1|out$ is applied, then the generated languages are $\{b_1^n b_2^{n+1} b_3^{n+1} \mid n \geq 1\}$. This program sends out 1, keeps x_1 unchanged, and increases y_1 by 1. Suppose it applies for $n-1$ steps, for some $n \geq 1$. Then $n-1$ copies of 1 are sent out, and variable y_1 increases to $n-1$. Then program $3x_1 \to 1|x_2+1|y_1+1|out$ is applied, which passes the value of x_1 to x_2, increases y_1,

and sends out 1 for one more time. After that, program $5x_2 \to 1|x_2+1|y_2+1|y_2'+2|out$ is applied for n steps, which sends out 2 for n times, and saves the number n in variables y_2 and y_2'. Then program $3(x_1 + x_2 + y_1 - y_2) \to 2|out + 1|x_3$ sets all the variables x_1, x_2, y_1, y_2 to zero, simultaneously sends out 2 once again, and increases x_3 to 1. After that, program $5x_3 \to 1|x_3 + 1|y_3 + 3|out$ is applied for n steps, which sends out 3 for n times, and saves the number n in variable y_3. Finally, program $3(x_2+x_3+y_3-y_2') \to 1|out$ sets the variables x_2, x_3, y_3, and y_2' to zero, simultaneously sends out 3 once again. All variables are now equal to zero, so zero is sent out and the computation ends.

Programs $3(x_1 + x_2 + y_1 - y_2) \to 2|out + 1|x_3$ and $3(x_2 + x_3 + y_3 - y_2') \to 1|out$ are trap programs. If the value of variable y_1 is not equal to y_2, or y_2' is not equal to y_3, then the corresponding program sends out a value greater than 3 or smaller than zero, both resulting in aborting the computation. □

Systems with a comparable descriptional complexity can also generate non-semilinear (hence again non-context-free) languages.

Theorem 6. *The family* $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_7)$ *contain non-semilinear languages.*

Proof. The numerical P system Π from Figure 5 generates the language $\{b_2\} \cup \{b_1^m b_2^{n+1} \mid n \geq 1, 1 \leq m \leq 2^{n+1} - 1\}$ which is a non-semilinear one.

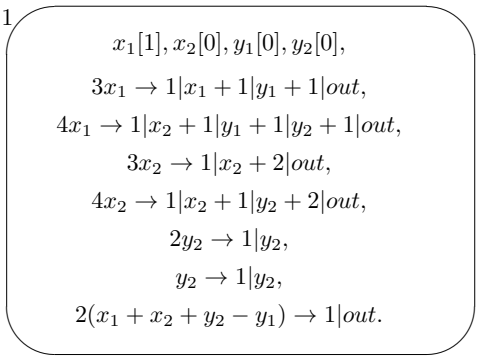

Fig. 5. The numerical P system from the proof of Theorem 6.

Initially, if the program $2(x_1 + x_2 + y_2 - y_1) \to 2|out$ is applied, then all the variables of the system are reset to zero, hence the computation ends. In this case, the system generates the string b_2.

If at the first step the program $3x_1 \to 1|x_1 + 1|y_1 + 1|out$ is applied, then the generated language is $\{b_1^m b_2^{n+1} \mid n \geq 1, 1 \leq m \leq 2^{n+1} - 1\}$.

Program $3x_1 \to 1|x_1 + 1|y_1 + 1|out$ sends 1 out, keeps the value of variable x_1 unchanged, and increases y_1 by 1. Suppose this program is applied for $m-1$ steps, for some $m \geq 1$. In this way, it sends out value 1 for $m - 1$ times and increases variable y_1 to $m-1$. Then the program $4x_1 \to 1|x_2+1|y_1+1|y_2+1|out$ is applied, which sets

x_1 to zero, increases variables x_2, y_1 and y_2 by 1, and sends out 1 once again. In the next step, variable x_1 equal to 0 halts the process sending out 1 and increasing the variable y_1; variable x_2 equal to 1 triggers the start of the process sending out 2. In the above m steps, the number of times that value 1 is sent out is m, which is saved in variable y_1.

The group of two programs $3x_2 \to 1|x_2 + 2|out$ and $4x_2 \to 1|x_2 + 1|y_2 + 2|out$ is used to send out 2. As they share the common variable x_2, at each step only one of them is applied. Similarly, only one of the two programs $2y_2 \to 1|y_2$ and $y_2 \to 1|y_2$ is applied at one step. Each group has two choices to choose the program to be applied – hence in total we have four combinations. At one step, by choosing different programs combinations to apply, the value of y_2 can be preserved unchanged, increases by 1, doubled, or doubled plus one. Hence, by choosing different combinations of the four programs to apply for n steps, the value of y_2 can be any value from 1 to $2^{n+1} - 1$. (This fact can be proved by induction. For $n = 1$, it is easy to check that applying the different combinations of the four programs for one step, y_2 can take any value from 1 to 3. Next, suppose the assertion holds true for $n = t > 1$, we prove it is also true for $n = t + 1$. Let p be any value from 1 to $2^{t+2} - 1$. If p is smaller than $2^{t+1} - 1$, then, according to the assumption, y_2 can be increased to p in t steps, then using the combination of programs $3x_2 \to 1|x_2+2|out$ and $y_2 \to 1|y_2$ for one step, not changing the value of y_2, hence $y_2 = p$ is obtained in $t + 1$ steps. If p is greater than $2^{t+1} - 1$, then p can be expressed as either $2q$ or $2q + 1$ such that $q \leq 2^{t+1} - 1$. Then according to the assumption, $y_2 = q$ can be obtained in t steps. If $p = 2q$, in the next step, applying the combination of the programs $3x_2 \to 1|x_2 + 2|out$ and $2y_2 \to 1|y_2$, $y_2 = 2q = p$ is obtained; if $p = 2q + 1$, in the next step, applying the combination of programs $4x_2 \to 1|x_2 + 1|y_2 + 2|out$ and $2y_2 \to 1|y_2$, $y_2 = 2q + 1 = p$ is obtained. Hence for p greater than $2^{t+1} - 1$, $y_2 = p$ can be obtained in $t + 1$ steps. Hence, the assertion holds true.) At last, we use the program $2(x_1 + x_2 + y_2 - y_1) \to 1|out$. If $y_2 \neq y_1$ the computation aborts. If $y_1 = y_2$, this program sends out 2, zeros all the variables of the system, hence in the next step, 0 is sent out, and the computation ends. When there are m copies of b_1 generated, the value of y_1 is m; when there are $n+1$ copies of b_2 generated, the value of y_2 is at most $2^{n+1} - 1$. Hence $y_1 = y_2$ means that when there are $n + 1$ copies of b_2 generated, the copies of b_1 generated are at most $2^{n+1} - 1$. The theorem holds true. □

6.2. The λ Case

The possibility to work inside the system without increasing the generated string adds power to any computing device, and this is also confirmed in our case for purely enzymatic systems (Theorem 13): we get a characterization of RE languages. This is not surprising, as we know that, used as number generators, numerical P systems are universal, both in the non-enzymatic case [11], and, even for restricted forms of production functions, for enzymatic systems [14, 15].

However, we were not able to also obtain a result of the form $RE = L^{\lambda}NP_1^{fin}(poly^n(r), Var_*, Pro_*)$, although we *conjecture* that such a result is true.

7. The Power of Purely Enzymatic Numerical P Systems

Using enzymes for controlling the programs application adds "programming" possibilities, but having all programs enzymatic is a restriction. With these ingredients having apparently contradictory effects, the results are, however, in a great extent similar to those from the previous sections.

Let us note first that rather restricted systems can generate infinite languages:

Theorem 7. $L^{res}pENP_1^{hal}(poly^1(1), Var_2, Pro_2) - FIN \neq \emptyset$.

Proof. Consider the purely enzymatic numerical P system $\Pi = (1, \{0\}, [\ ]_1, (\{x_1, x_2\}, \{2x_1|_{x_2} \to 1|x_1 + 1|out,\ 2(1 + x_1)|_{x_2} \to 1|x_1 + 1|out\}, (1, 2)), 2)$, which can generate the infinite language $\{b_1^n b_2 \mid n \geq 0\}$. Only one of the two programs can be applied, because the system works in the one-parallel mode. After $n \geq 0$ steps when the program $2x_1|_{x_2} \to 1|x_1 + 1|out$ is used, once the program $2(1 + x_1)|_{x_2} \to 1|x_1 + 1|out$ is applied, in the next step no program can be applied, hence the system halts. □

With sufficiently many variables and programs, all finite languages can be generated (with the simplest production functions):

Theorem 8. $FIN \subset L^{res}pENP_1^{hal}(poly^1(1), Var_*, Pro_*)$.

Proof. Let $L = \{x_1, x_2, \ldots, x_n\} \subset V^*, n \geq 1$, be a finite language, and let $x_i = x_{i,1} \ldots x_{i,r_i}$ for $x_{i,j} \in V$, $1 \leq i \leq n, 1 \leq j \leq r_i = |x_i|$. For $b \in V$, define $ind(b) = i$ if $b = b_i$. Then L can be generated by the purely enzymatic numerical P system shown in Figure 6.

1

$$e_1[1], e_1'[2], e_{i,j}[0], e_{i,j}'[1], g[0], g_{i,j}[0], \text{ for all } 1 \leq i \leq n, 2 \leq j \leq r_i,$$

$$g + 3ind(x_{i,1})|_{e_1} \to 1|out + 1|e_{i,2} + 1|e_{i,2}', \text{ for all } 1 \leq i \leq n,$$
$$4e_1|_{e_1'} \to 1|e_1 + 1|g,$$

$$8e_{i,j}|_{e_{i,j}'} \to 1|g_{i,j} + 1|e_{i,j} + 1|e_{i,j+1} + 1|e_{i,j+1}', \text{ for all } 1 \leq i \leq n, 2 \leq j \leq r_i - 1,$$
$$g_{i,j} + ind(x_{i,j})|_{e_{i,j}} \to 1|out, \text{ for all } 1 \leq i \leq n, 2 \leq j \leq r_i,$$
$$4e_{i,r_i}|_{e_{i,r_i}'} \to 1|g_{i,r_i} + 1|e_{i,r_i}, \text{ for all } 1 \leq i \leq n.$$

Fig. 6. The purely enzymatic numerical P system from the proof of Theorem 8.

Initially, the enzymatic variable e_1 equals 1, which activates programs $g + 3ind(x_{i,1})|_{e_1} \to 1|out + 1|e_{i,2} + 1|e_{i,2}'$, $1 \leq i \leq n$. Because they share a common variable g, only one of them can be applied. We non-deterministically choose one to apply. Program $g+3ind(x_{i,1})|_{e_1} \to 1|out+1|e_{i,2}+1|e_{i,2}'$ sends out the first symbol $x_{i,1}$ of the string x_i and increases the values of the enzymatic variables $e_{i,2}$ and $e_{i,2}'$, which will activate the programs generating the second symbol of the string x_i. Simultaneously, program $4e_1|_{e_1'} \to 1|e_1 + 1|g$ is applied, which increases the value of variable e_1 and g to 2, hence disables all the programs $g + 3ind(x_{i,1})|_{e_1} \to 1|out + 1|e_{i,2} + 1|e_{i,2}'$, $1 \leq i \leq n$, and itself in the next step.

The j-th step of the computation generates the j-th symbol of the string x_i by applying the program $g_{i,j} + ind(x_{i,j})|_{e_{i,j}} \to 1|out$. Simultaneously, program $8e_{i,j}|_{e'_{i,j}} \to 1|g_{i,j} + 1|e_{i,j} + 1|e_{i,j+1} + 1|e'_{i,j+1}$ increases all the variables $g_{i,j}$, $e_{i,j}$, $e_{i,j+1}$, and $e'_{i,j+1}$ by the double value of variable $e_{i,j}$. Thus, for the next step, variable $e_{i,j}$ equal to $g_{i,j}$ and greater than variable $e'_{i,j}$ disables the two programs above. The system proceeds in this way to generate the symbols one by one. In the last step programs $g_{i,r_i} + ind(x_{i,r_i})|_{e_{i,r_i}} \to 1|out$ and $4e_{i,r_i}|_{e'_{i,r_i}} \to 1|g_{i,r_i} + 1|e_{i,r_i}$ are applied. The former program generates the last symbol of string x_i and the latter program increases the values of variables g_{i,r_i} and e_{i,r_i}, thus disabling both the programs above. After that, no program can be applied, hence the system halts. Non-deterministically choosing the first program to apply, each string of L can be generated. Thus, the theorem holds true. □

With slightly more complex production functions (involving two variables) we can cover all regular languages:

Theorem 9. $REG \subseteq L^{res}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.

Proof. For $L \in REG$, consider a regular grammar $G = (N, V, S, P)$ such that $L = L(G)$, where $N = \{A_1, A_2, \ldots, A_n\}, n \geq 1, S = A_1$, and the rules in P are of the forms $A_i \to b_k A_j, A_i \to b_k, 1 \leq i, j \leq n, 1 \leq k \leq q$. Without any lose of the generality, we suppose $i \neq j$.

Then L can be generated by the purely enzymatic numerical P system shown in Figure 7.

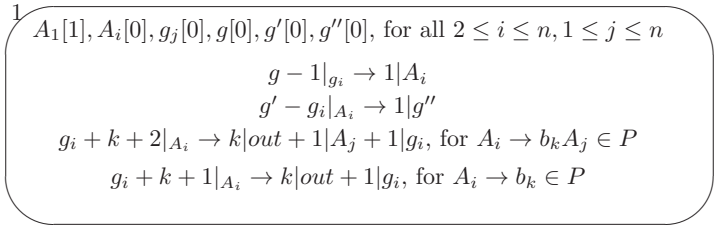

Fig. 7. The purely enzymatic numerical P system from the proof of Theorem 9.

In each step, only one of the variables A_i is equal to 1, which indicates one of the rules starting with A_i in P is simulated. For one-parallel mode, in each step, only one of the programs, which are associated with enzymatic variable A_i sharing the common variable g_i, can be applied. One of these programs is non-deterministically chosen to be applied. If the program $g_i + k + 2|_{A_i} \to k|out + 1|A_j + 1|g_i$ is chosen, then A_j gets a contribution 1, which correctly points to the next rule to be simulated, and g_i also gets a contribution 1, which disables this program in the next step; simultaneously, the program sends outside a number k indicating that b_k is generated. The simulation of the rule of G is correct. In the next step, programs $g-1|_{g_i} \to 1|A_i, g'-g_i|_{A_i} \to 1|g''$ are applied, which reset the variables A_i and g_i to zero. Simultaneously, $A_j = 1$ activates the programs simulating the rules which start with A_j, and this is done in the same

way as above. The process continues until the program $g_i + k + 1|_{A_i} \to k|out + 1|g_i$ is applied, which simulates the rule $A_i \to b_k$. There is no new A_j equal to 1, hence no program can be used, the system halts. Clearly, at each step a positive value is sent out, hence the theorem holds true. □

The previous inclusion is proper. Example 3 proves this assertion, for small numbers of variables and programs (still, larger than those in Theorem 7).

Theorem 10. $L^{res}pENP_1^{hal}(poly^1(1), Var_6, Pro_4) - REG \neq \emptyset$.

Actually, also non-context-free languages can be generated (at the price of further increasing the number of variables and programs):

Theorem 11. $L^{res}pENP_1^{hal}(poly^1(1), Var_9, Pro_6) - CF \neq \emptyset$.

Proof. Extending the idea in Example 3, we can construct the system from Figure 8 which generates the non-context-free language $\{b_1^n b_2^{n+1} b_3^{n+3} \mid n \geq 1\}$. We leave the task of examining the way this system works as an exercise to the reader. □

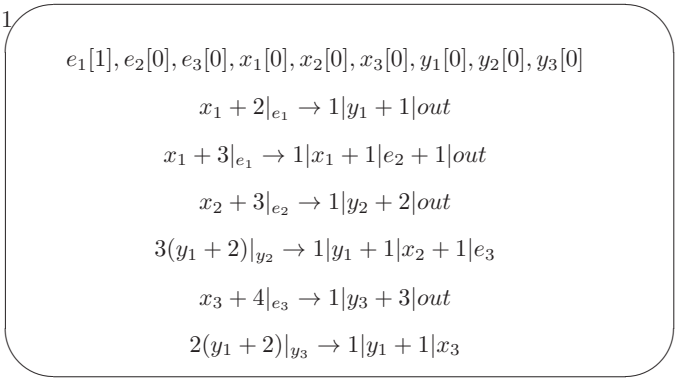

Fig. 8. The purely enzymatic numerical P system from the proof of Theorem 11.

Non-semilinear languages (hence again non-context-free) can also be generated by purely enzymatic NP systems (with a number of variables and programs comparable to the previous result).

Theorem 12. *The family* $L^{res}pENP_1^{hal}(poly^1(2), Var_7, Pro_6)$ *contains non-semilinear languages.*

Proof. The purely enzymatic NP system Π from Figure 9 generates the non-semilinear language $\{b_1^n b_2^m \mid n \geq 1, 1 \leq m \leq 2^n + 1\}$.

Initially, variable e_1 equals 1, which activates the following three programs

$$x_1 + 1|_{e_1} \to 1|out, \tag{2}$$

$$x_1 + 4|_{e_1} \to 1|x_1 + 1|x_2 + 1|e_2 + 1|out \tag{3}$$

$$x_2 + 2y_1|_{e_1} \to 1|y_1. \tag{4}$$

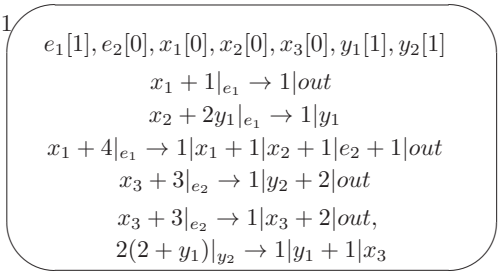

Fig. 9. The purely enzymatic numerical P system from the proof of Theorem 12.

Because the system works in the one-parallel mode, at each step one of the programs (2) and (3) (they share the same variable x_1) together with program (4), can be applied. If program (2) is chosen, then in the next step all the three programs can still be applied. Once program (3) is applied, the values of variables x_1 and x_2 will be increased, which disables these three programs application. Suppose program (2) is applied for $n-1$ steps, for some $n \geq 1$. It sends out 1 for $n-1$ times. Then program (3) is applied, which sends out 1 once again and sends a contribution 1 to variables x_1, x_2, and e_2. Simultaneously, in the above n steps, program (4) increases variable y_1 to 2^n. The next step, variables x_1 and x_2 equal 1, which disables the application of programs (2) – (4); at the same time e_2 equals 1, which activates the following two programs

$$x_3 + 3|_{e_2} \to 1|y_2 + 2|out, \tag{5}$$

$$x_3 + 3|_{e_2} \to 1|x_3 + 2|out. \tag{6}$$

Programs (5) and (6) share the same variable x_3, hence only one of them can be applied. Program (5) increases the variable y_2 and sends 2 out. The role of program (6) is to increase the value of x_3 to 1, thus disabling these two programs. If program (6) does not apply, then program (5) can be applied at most $2^n + 1$ times. We can see that after 2^n steps, the value of y_2 is increased to $2^n + 1$, which activates the program $2(2+y_1)|_{y_2} \to 1|y_1+1|x_3$. In the next step, this program increase the values of variables y_1 and x_3 to $2^n + 2$, thus disables the programs (5), (6), and itself. The system halts. □

As expected, passing from languages $L^{res}(\Pi)$ to languages $L^{\lambda}(\Pi)$, the power of our systems increases to universality:

Theorem 13. $RE = L^{\lambda}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.

Proof. We prove that for any recursively enumerable language $L \subseteq V^*$, there exists a purely enzymatic numerical P system Π, such that $L^{\lambda}(\Pi) = L$.

For any such L, there exists a deterministic register machine M which halts after processing the input u_0 placed initially in its input register if and only if $u_0 = val_q(w)$ for some $w \in L$. Therefore, it is sufficient to show how to output a string w, generate

the encoding $val_q(w)$, and simulate the instructions of a register machine with a purely enzymatic numerical P system.

We construct the purely enzymatic numerical P system Π performing the following operations:

1. Output value i, for some $1 \leq i \leq q$, and at the same time save value i in variable x_2;

2. Multiply the number stored in variable x_1, which represents the value of register 1 of register machine M (initially, its value is 0) by $q+1$, then add the value of variable x_2;

3. Repeat from step 1, or, non-deterministically, stop the increase of the value of variable x_1 (and hence of the string) and pass to the next step;

4. After the last increase of the value of x_1, we have here $val_q(x)$ for a string $x \in V^+$. Start now to simulate the work of the register machine M in recognizing the number $val_q(x)$. The computation halts only if this number is accepted by M, hence the string x produced by the system is introduced in the generated language only if $val_q(x) \in N(M)$.

In constructing the system Π we use the fact that a register machine can be simulated by an enzymatic numerical P system [4]; actually, the system constructed in [4] is purely enzymatic.

The rest of the construction (sending out the value corresponding to a symbol and introducing the same value in variable x_2; computing the value of the produced string; choosing non-deterministically to end the string to generate and switching to the checking phase, etc.) is explicitly presented below.

The overall appearance of Π is given in Figure 10. Suppose $M = (m, H, l_0, l_h, I)$.

Initially, all variables are equal to 0, with the exception of e_0, which is equal to 1 and which activates the programs $a + 2i + 3|_{e_0} \to 1|a + i|x_2 + 1|e + 1|e' + i|out$, $1 \leq i \leq m$. Because these programs share the variable a, only one of them can be applied, non-deterministically choosing the one to apply. The i-th program sends a contribution i out, hence the first letter b_i of the generated string. Simultaneously, value i is saved in variable x_2. Variable a is increased by 1, which disables the application of this program in the next step. Variables e and e' are also increased by 1, which will activate the programs of computing $val_q(x)$.

Programs $d + 2(q+1)x_1|_e \to 1|x_1 + 1|x_1'$ and $d' + 2x_2|_e \to 1|x_1 + 1|x_1'$ increase the value of variable x_1 to $val_q(x)$ and that of x_1' to a value greater than 1. Simultaneously, programs $d' - 1|_{e'} \to 1|e$ and $d'' - 1|_e \to 1|e'$ set the variables e and e' to zero, respectively, thus disabling the programs $d+2(q+1)x_1|_e \to 1|x_1+1|x_1'$ and $d'+2x_2|_e \to 1|x_1 + 1|x_1'$ in the next step. Variable x_1' is greater than 1, hence it activates the programs $a' + a|_{x_1'} \to 1|a''$ and $a' + 2e_0|_{x_1'} \to 1|p_0 + 1|p_0'$. However only one of them is applied. The former one sets variable a to zero, which activates the program $a + 2i + 3|_{e_0} \to 1|a + i|x_2 + 1|e + 1|e' + i|out$, $1 \leq i \leq m$, thus restarts the process of generating a symbol of the string. The latter program passes the value of variable e_0 to variables p_0 and p_0', which activates the programs of simulating the register machine M. We non-deterministically choose one program to apply, thus the system either continues to prolong the string or passes to the checking phase.

The programs used in [4] are taken here without any change, excepting some

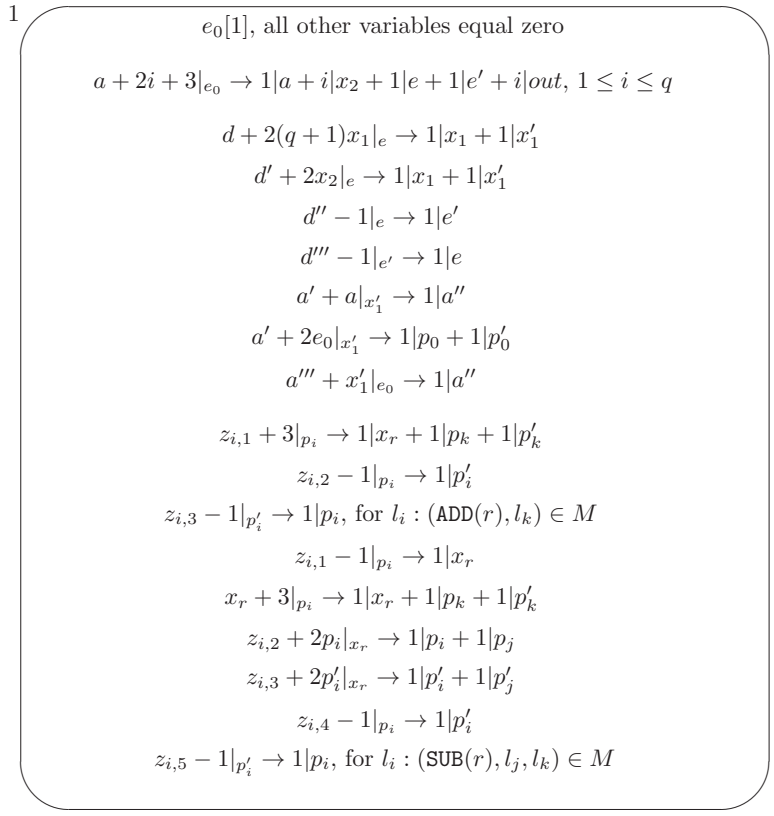

Fig. 10. The numerical P system from the proof of Theorem 13.

notations of variables, hence we do not describe their functioning.

When the value $val_q(x)$ is accepted, register machine M reaches the halt instruction l_h, thus the corresponding variables p_h and p_h' become equal to 1. As there is no program which contains these two variables in the production functions or as enzymatic variables, no program can be applied. Consequently, the system halts. □

8. The Power of Enzymatic Numerical P Systems

As we have pointed out in Section 5, all results about non-enzymatic systems directly extend to enzymatic systems. For the reader's convenience, we recall them here:

Theorem 14. *1.* $L^{res}ENP_1^{fin}(poly^1(1), Var_1, Pro_2) - FIN \neq \emptyset$.

2. $REG \subseteq L^{res}ENP_1^{fin}(poly^1(1), Var_*, Pro_*)$.

3. $L^{res}ENP_1^{fin}(poly^1(4), Var_4, Pro_4) - REG \neq \emptyset$.

4. $L^{res}ENP_1^{fin}(poly^1(4), Var_7, Pro_6) - CF \neq \emptyset$.

5. *The family* $L^{res}ENP_1^{fin}(poly^1(4), Var_4, Pro_7)$ *contains non-semilinear languages.*

Note that point 1 above is stronger than the enzymatic counterpart from Theorem 7, point 2 is stronger than the counterpart from both Theorem 8 and Theorem 9, but Theorems 10, 11, and 12 do not have stronger counterparts in the non-enzymatic case. Actually, the results for purely enzymatic systems have minimal production functions (one variable, of degree one), but slightly more variables.

We leave as an exercise the task of finding non-regular or non-context-free languages generated by simple enzymatic numerical P systems, and we prove here only a counterpart of Theorem 12, stronger than result 5 in Theorem 14:

Theorem 15. *The family* $L^{res}ENP_1^{hal}(poly^1(2), Var_4, Pro_6)$ *contains non-semilinear languages.*

Proof. The enzymatic numerical P system Π from Figure 11 generates the non-semilinear language $\{b_1^n b_2^m \mid n \geq 1, 1 \leq m \leq 2^{n+1} + 2\}$.

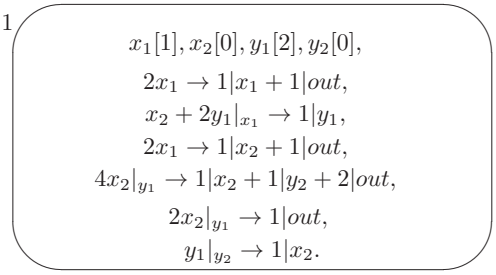

Fig. 11. The enzymatic numerical P system from the proof of Theorem 15.

Initially, non-deterministically choose one of the two programs $2x_1 \to 1|x_1+1|out$ and $2x_1 \to 1|x_2 + 1|out$ to apply. If the program $2x_1 \to 1|x_1 + 1|out$ is chosen, it sends out 1, and keeps x_1 unchanged. Suppose this program is applied for $n-1$ steps, for some $n \geq 1$. Then the number of copies of 1 sent out is $n - 1$. Then program $2x_1 \to 1|x_2 + 1|out$ is applied, which sends out 1 once more, zeros variable x_1, and increases variables x_2 by 1. In the next step, variable x_1 equal to 0 halts the process of sending out 1, while variable x_2 equal to 1 triggers the start of the process sending out 2 and disables the program increasing the variable y_2. Simultaneously, in the above n steps, program $x_2 + 2y_1|_{x_1} \to 1|y_1$ increases the value of variable y_2 to 2^t. The value t is between 0 to n, that is the application steps it wins in the competition with the other two programs $4x_2|_{y_1} \to 1|x_2 + 1|y_2 + 2|out$ and $2x_2|_{y_1} \to 1|out$ among the above n steps. In the above n steps, the copies of 1 sent out are n, and the value of y_1 equals 2^t $(t \leq n + 1)$.

When x_2 equals 1 and y_1 equals 2^t ($t \leq n+1$), both programs $4x_2|_{y_1} \to 1|x_2 + 1|y_2 + 2|out$ and $2x_2|_{y_1} \to 1|out$ can be applied. However, in each step only one of them is non-deterministically chosen to be applied. If the former program $4x_2|_{y_1} \to 1|x_2+1|y_2+2|out$ is applied, it sends out 2, increases y_2 by 1, and keeps x_2 unchanged. Hence in the next step, these two programs can still be applied. If the latter program $2x_2|_{y_1} \to 1|out$ is applied, x_2 is zeroed. Then in the next step, 0 is sent out, thus ending the computation. On the other hand, even if the latter program $2x_2|_{y_1} \to 1|out$ is not applied, the computation will be ended after the former program $4x_2|_{y_1} \to 1|x_2 + 1|y_2 + 2|out$ is applied for $2^t + 2$ steps, hence at most $2^t + 2$ copies of 2 is sent out. In fact, after $2^t + 1$ steps, variable y_2 increases to $2^t + 1$, greater than the value of y_1, thus activates the program $y_1|_{y_2} \to 1|x_2$. This program zeros the variable y_1, which disables programs $4x_2|_{y_1} \to 1|x_2 + 1|y_2 + 2|out$ and $2x_2|_{y_1} \to 1|out$ in the next step. Simultaneously, program $4x_2|_{y_1} \to 1|x_2 + 1|y_2 + 2|out$ is applied and sends out 2 for one more time. Hence, in the next step, only one of the programs $2x_1 \to 1|x_1 + 1|out$ or $2x_1 \to 1|y_2 + 1|out$ sends out zero. The computation ends. The total copies of 2 sent out is $2^t + 2$. Therefor, for $t \leq n + 1$, the copies of 2 sent out are at most $2^{n+1} + 2$. The theorem holds true. □

9. Concluding Remarks

In this work, we have investigated numerical P systems as string generators. We have examined the generative capacity of such systems (both numerical P systems and enzymatic systems are considered) comparing the generated families of languages with the families of finite, regular, context-free, and recursively enumerable languages of the Chomskian hierarchy. As this paper just opens this research vista, many questions remain to be investigated.

For instance, the systems considered in this paper work in the one-parallel mode. We noticed that the all-parallel mode is of no interest (the system behave then deterministically), but the generative capacity of numerical P systems working in the sequential mode remains to be examined.

For numerical P systems working in the one-parallel mode, there are 6 families obtained by combining two classification criteria: restrictive or λ; enzymatic, purely enzymatic or non-enzymatic. Similarly, 6 families are obtained for the sequential mode. The precise relationships between these families remain to be found, as well as the relations between them and the families in the Chomsky hierarchy.

One further possibility is to proceed as in the case of spiking neural P systems and also consider the language generated by producing symbols b_i as above, for $i = 1, 2, \ldots, q$, but also associating the symbol b_0 to a step when value 0 is sent out. Which of the results reported here remain valid, what new results can be obtained?

For all families which are not known to be equal to families in the Chomsky hierarchy, it might be of interest to examine their closure properties and decidability properties. (For families which are not universal we only know – Theorem 1 – that their membership problem is decidable.) What restriction should be imposed in order to get characterizations of REG or CF?

It is of interest to design optimization algorithms based on numerical P systems, as in the case of optimization spiking neural P systems in [17].

Acknowledgements. This paper is written during a three months stay of Zhiqiang Zhang and Tingfang Wu in Curtea de Argeş, Romania, in the fall of 2015. This work is supported by National Natural Science Foundation of China (61033003, 61320106005 and 61472154), Ph.D. Programs Foundation of Ministry of Education of China (2012014213008), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

[1] CHEN H., FREUND R., IONESCU M., PĂUN Gh., PÉREZ-JIMÉNEZ M.J., *On string languages generated by spiking neural P systems*, Fundamenta Informaticae, 2007, **75**(1), pp. 141–162.

[2] CHEN H., ISHDORJ T.-O., PĂUN Gh., PÉREZ-JIMÉNEZ M.J., *Spiking neural P systems with extended rules*, Proc. Fourth Brainstorming Week on Membrane Computing, Sevilla, 2006, RGNC Report 02/2006, pp. 241–265.

[3] IONESCU M., PĂUN Gh., YOKOMORI T., *Spiking neural P systems*, Fundamenta Informaticae, 2006, **71**(2-3), pp. 279–308.

[4] LEPORATI A., PORRECA A.E., ZANDRON C., MAURI G., *Improving universality results on parallel enzymatic numerical P systems*, Proceedins of 11th Brainstorming Week on Membrane Computing, Sevilla, February 2013, Fenix Editora, Sevilla, 2013, pp.177–200.

[5] MINSKY M.L., *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, N.J., 1967.

[6] PAVEL A.B., ARSENE O., BUIU C., *Enzymatic numerical P systems – A new class of membrane computing systems*, Proc. IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010, pp. 1331–1336.

[7] PAVEL A.B., BUIU C., *Using enzymatic numerical P systems for modeling mobile robot controllers*, Natural Computing, 2012, **11**(3), pp. 387–393.

[8] PAVEL A.B., VASILE C.I., Dumitrache I., *Robot localization implemented with enzymatic numerical P systems*, Biomimetic and Biohybrid Systems, Springer, Berlin, 2012, pp. 204–215.

[9] PAVEL A.B., VASILE C.I., DUMITRACHE I., *Membrane computing in robotics*, Beyond Artificial Intelligence, Springer, Berlin, 2013, pp. 125–135.

[10] PĂUN Gh., *Membrane Computing. An Introduction*, Springer, Berlin, 2002.

[11] PĂUN Gh., PĂUN R., *Membrane computing and economics: numerical P systems*, Fundamenta Informaticae, 2006, **73**(1), pp. 213–227.

[12] PĂUN Gh., ROZENBERG G., SALOMAA A., (Eds.), *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford, UK, 2010.

[13] ROZENBERG G., SALOMAA A., (eds.), *Handbook of Formal Languages*, Springer, Berlin, 1997.

[14] VASILE C.I., PAVEL A.B., DUMITRACHE I., *Universality of enzymatic numerical P systems*, International Journal of Computer Mathematics, 2013, **90**(4), pp. 869–879.

[15] VASILE C.I., PAVEL A.B., DUMITRACHE I., PĂUN Gh., *On the power of enzymatic numerical P systems*, Acta Informatica, 2012, **49**(6), pp. 395–412.

[16] WANG X., ZHANG G., NERI F., JIANG T., ZHAO J., GHEORGHE M., IPATE F., LEFTICARU R., *Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots*, Integrated Computer-Aided Engineering, 2015, **23**(1), pp. 15–30.

[17] ZHANG G., RONG H., NERI F., PÉREZ-JIMÉNEZ M.J., *An optimization spiking neural P system for approximately solving combinatorial optimization problems*, International Journal of Neural Systems, 2014, **24**(5), pp. 1–16.