Some Open Problems about Numerical P Systems

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and

Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, ghpaun@gmail.com

Summary. Some open problems and research topics related to numerical P systems are formulated – also recalling the problems from the corresponding section of the "mega-paper" [2] produced for the previous BWMC.

1 Introduction

Although introduced already in 2006, [8] (see also Section 8.1 of Chapter 23 of [10]) – and being an alternative to multiset or string processing P systems, meant to compute using numerical variables – the numerical P systems have received more attention only in the last years, especially in the framework of devising and implementing controllers for mobile robots.

In short, numerical P systems are a class of computing models inspired by both the cell structure and economics: numerical variables evolve in the compartments of a cell-like structure by means of so-called *production–repartition programs*. The variables have a given initial value and the production function is usually a polynomial whose values for the current values of variables is distributed among variables in the neighboring compartments according to the "repartition protocol". In this way, the values of variables evolve; all positive values taken by a specified variable are said to be computed by the P system.

In a natural way, these systems can also be used for computing mappings: specified variables of the system are considered as being the function variables and specified variables provide the results (hence functions from vectors to vectors of numbers can be computed – this is the case also in the robot control; of course, a suitable way to define the end of the computation should be found – halting, for instance, although in the basic computing model the computation is not supposed to halt).

In what follows, in order to help the reader (however, (s)he is supposed to be familiar with basic elements of membrane computing), I will first recall the definition of numerical P systems, as given in [10] (Section 23.8.1), with two simple examples, then I will briefly discuss, following [15], the enzymatic numerical P systems as used in robot control; finally, further research suggestions are given.

2 Definitions

We consider usual cell-like membrane structures (with the standard $1, 2, \ldots, m$ labeling of membranes). The regions delimited by these membranes contain numerical variables. The variables in region i are written in the form $x_{j,i}, j \geq 1$. The value of $x_{j,i}$ at time $t \in \mathbf{N}$ is denoted by $x_{j,i}(t)$. These values can be of various types – in what follows we consider integers as values of variables (although in many applications one would most probably use real numbers – this is the case for robot control).

In order to evolve the values of variables, we use *programs*, composed of two components, a *production function* and a *repartition protocol*. The former can be any function with variables from a given region – here we are interested in computability issues, hence we consider only polynomials with integer coefficients. Using such a function (chosen non-deterministically if there are several programs in a given region), we compute a *production value* of the region at a given step. 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 associated with the used production function. For a given region i, let $v_1, \ldots, v_{n_i}$ be all these variables. Following [8], here we consider as repartition protocols expressions of the form

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

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

This is precisely defined as follows. Consider a program

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

and let

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

At a time instant $t \geq 0$ we compute $F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))$. The value $q = F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))/C_{l,i}$ represents the "unitary portion" to be distributed according to the repartition expression to variables $v_1, \ldots, v_{n_i}$. Thus, $v_{l,s}$ will receive $q \cdot c_{l,s}, 1 \leq s \leq n_i$.

A program as above is also written in the form

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

A production function may use only part of the variables from a region. Those variables "consume" their values when the production function is used (they become zero) – the other variables retain their values. To these values – zero in the case of variables contributing to the region production – one adds all "contributions" received from the neighboring regions.

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

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

where μ is a membrane structure with m membranes labeled injectively by $1, 2, \ldots, m$, Var_i is the set of variables from region i, Pr_i is the set of programs from region i (all sets Var_i, Pr_i are finite), $Var_i(0)$ is the vector of initial values for the variables in region i, and x_{j_0,i_0} is a distinguished variable (from a distinguished region i_0), which provides the result of a computation.

Each program is of the form specified above: $pr_{l,i} = (F_{l,i}(x_{1,i}, \ldots, x_{k_i,i}), c_{l,1}|v_1 + c_{l,2}|v_2 + \ldots + c_{l,n_i}|v_{n_i})$ denotes the lth program from region i, where $\{x_{1,i}, \ldots, x_{k_i,i}\} \subseteq Var_i$ (not all variables from region i should appear in $F_{l,i}$).

Such a system evolves in the way informally described before. Initially, the variables have the values specified by $Var_i(0), 1 \leq i \leq m$. A transition from a configuration at time instant t to a configuration at time instant $t+1$ is made by (i) choosing non-deterministically one program from each region, (ii) computing the value of the respective production function for the values of local variables at time t, and then (iii) computing the values of variables at time $t+1$ as directed by repartition protocols. A sequence of such transitions forms a computation, with which we associate a set of numbers, namely, the numbers which occur as positive values of the variable x_{j_0,i_0}; this set of numbers is denoted by $N^+(\Pi)$. If all numbers, positive or negative, are taken into consideration, then we write $N(\Pi)$.

3 Examples

I illustrate the previous definition with the numerical system Π_1 given in Figure 1 with the distinguished variable $x_{1,1}$. One can easily see that variable $x_{1,3}$ increases by 1 at each step, also transmitting its value to $x_{1,2}$. In turn, region 2 transmits the value $2x_{1,2} + 1$ to $x_{1,1}$, which is never consumed, hence its value increases continuously. In the initial configuration all variables are set to 0. Thus, $x_{1,1}$ starts from 0 and continuously receives $2i + 1$, for $i = 0, 1, 2, 3, \ldots$, which implies that in $n \geq 1$ steps the value of $x_{1,1}$ becomes $\sum_{i=0}^{n-1}(2i+1) = n^2$, and consequently $N^+(\Pi_1) = \{n^2 \mid n \geq 0\}$.

The system Π_1 was deterministic; let us consider also a non-deterministic system:

$$\Pi_2 = ([\ \ ]_1, (\{x_{1,1}\}, \{(2x_{1,1}, 1|x_{1,1}),\ (3x_{1,1}, 1|x_{1,1})\}, 1), x_{1,1}).$$

The production is assigned to the unique variable, but in each step we can choose either the first program or the second one; in the former case $x_{1,1}$ is multiplied by

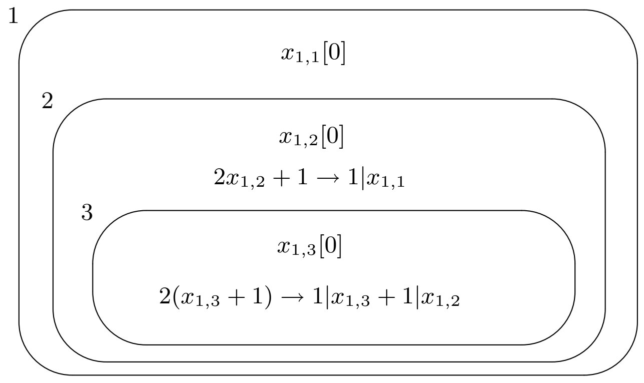
1
$x_{1,1}[0]$
2
$x_{1,2}[0]$
$2x_{1,2} + 1 \to 1|x_{1,1}$
3
$x_{1,3}[0]$
$2(x_{1,3} + 1) \to 1|x_{1,3} + 1|x_{1,2}$

Fig. 1. The system Π_1

2, and in the latter case it is multiplied by 3. Thus, the values of $x_{1,1}$ will be of the form $2^i 3^j$, with $i \geq 0, j \geq 0$. Actually, *all numbers of this form are values of* $x_{1,1}$, where the value $2^i 3^j$ is obtained in step $i + j$.

In these two examples we have chosen the programs in such a way that the production value is divisible by the total sum of coefficients c_j from each region (let us denote this case with *div*). When a current production is not divisible by the given total value of coefficients, then we can take the following decisions: (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 by *lost, carry, stop*, respectively.

4 Families of Numbers Computed

Thus, we can distinguish many types of systems, depending on the programs and their use. The family of sets of numbers $N(\Pi)$ computed by numerical P systems with at most m membranes, production functions which are polynomials of degree at most n, with at most r variables in each polynomial, with nonnegative coefficients, and the distribution of type α is denoted by $NNP_m(poly^n(r), nneg, \alpha),\ m \geq 1, n \geq 0, r \geq 0, \alpha \in \{div, lost, carry, stop\}$. The restriction to deterministic systems is indicated by adding the letter D in front of NNP. If arbitrary coefficients are allowed, then the indication "nneg" is removed. If one of the parameters m, n, r is not bounded, then it is replaced by $*$. The set of positive numbers occurring as values of the output variable is denoted by $N^+(\Pi)$, and NN gets the superscript $+$ when considering the family of such sets. Let NRE be the family of Turing computable sets of natural numbers.

Here are some results concerning these families – mode details can be found in [8].

Theorem 1. (i) $DNN^+P_1(poly^1(1), nneg, div) - SLIN_1^+ \neq \emptyset$.
(ii) $SLIN_1^+ \subset DNN^+P_*(poly^1(1), nneg, div)$.

The main result of [8] shows that, surprisingly enough, numerical P systems of a rather restricted type are Turing complete, even when using small numbers of membranes and polynomials of low degrees with a small number of variables:

Theorem 2. $NRE = NN^+P_8(pol^5(5), div) = NN^+P_7(poly^5(6), div)$.

The proof is based on the characterization of recursively enumerable sets of numbers as positive values of polynomials with integer values, [3]. Latter (see below) the register machines were used in universality proofs, and similar results were obtained, in certain cases, also for deterministic numerical P systems.

Many research topics are open for numerical P systems, among others: a throughout investigation of all classes of systems mentioned above, considering also vectors of numbers, looking for non-universal classes (and decidability results for those classes), hierarchies and normal forms.

5 Enzymatic Numerical P Systems

The numerical P systems were recently used in a series of papers (see references in [1], [14]) for implementing controllers for mobile robots; in this framework the P systems work in the computing mode: an input is introduced in the form of the values of some variables and an output is produced, as the values of other variables. Furthermore, in the robot control context, the so-called *enzymatic* numerical P systems were introduced and used, [4], [5], [6]. Such systems correspond to *catalytic P systems* in the "general" membrane computing: a program is applied only if the value of the associated enzyme is strictly greater than the smallest value of any variable involved in the production polynomial. Enzyme variables are not consumed or produced by the rules which they catalyze, but can be changed by the rules for which they do not act as catalysts. Therefore, their values can evolve during the computational process.

More formally (we recall the definition from [12]), enzymatic numerical P systems (in short, EN P systems) use both evolution programs as introduced above and programs of the form

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

where $e_{j,i}$ is a variable from Var_i different from $x_{1,i}, \dots, x_{k_i,i}$, and from $v_1, \dots, v_{n_i}$. Such a program can be applied at a time t only if $e_{j,i}(t) > \min(x_{1,i}(t), \dots, x_{k_i,i}(t))$. (A slightly more complex definition is considered in [5] and [6] where: $e_{j,i}(t) > \min(|x_{1,i}(t)|, \dots, |x_{k_i,i}(t)|)$. Considering the absolute value of the variables, instead of their real values, simplifies the design of the membrane structures used to compute *cos* and *sin* functions as power series, but here we consider only the simpler case defined above. We also use here a notation different from that in [5], writing the enzyme in the same way as the promoters are written in multiset rewriting rules.) Note that in order to apply the program it is sufficient that *one variable* has the current value strictly smaller than the value of the enzyme variable. The enzyme cannot evolve by means of the associated program, but it can evolve by means of other programs, and can receive "contributions" from other programs and regions.

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

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

Tissue numerical P systems are also considered in [12], with a parallel use of programs. If in each membrane, at each step, we use a maximal set of programs (programs are selected non-deterministically, and a set of programs is applied only if it is maximal, i.e., no further program can be added to it in such a way that the new set is still applicable). Clearly, two cases are possible: (i) a variable can appear only in *one* production function, and this is the only restriction in choosing (non-deterministically) the programs to apply in a step (we denote this variant with *oneP*), and (ii) if two or more programs which are enabled at a computation step (i.e., they satisfy the condition imposed by the associated enzymes), share variables in their production functions, then they will all use the current values of those variables (we denote this with *allP*).

A large variety of classes of numerical P systems is created in this way: (1) enzymatic or non-enzymatic, (2) deterministic or non-deterministic, (3) sequential, all-parallel, one-parallel, (4) used in the generating, computing, accepting mode, etc. By combining these variants – also considering the cases *div, lost, carry, stop* from the previous sections – a large variety of classes of numerical P systems can be investigated.

In the notations of the families $NN^{\alpha}P_m(poly^n(r), \ldots)$ considered in the previous sections we add now the indication *enz* when enzymes are used, and one of *seq, oneP, allP*, depending on the way (sequential or parallel) the rules are used. When tissue systems are used, we write $NNtP_m(poly^n(r), \alpha, \beta, \gamma)$. However, in what follows we do not mention *div* and *nneg*, as they are always present.

Here are the main results from [8] (written in the new notation) and [12].

Theorem 3. $NRE = NN^+P_8(poly^5(5), seq) = NN^+P_7(poly^5(6), seq) = NNP_7(poly^5(5), enz, seq) = NNtP_*(poly^1(11), enz, oneP) = NNP_{254}(poly^2(253), enz, allP, det)$.

A considerable improvement of the last equality was proved in [13]:

Theorem 4. $NRE = NNP_4(poly^1(6), enz, allP, det)$.

Whether or not the parameters appearing in these results are optimal or not is an open problem.

6 Open Problems

Only a few of the many cases mentioned above were so far investigated, the other ones wait for further research efforts.

In particular, we have seen that enzymes improve the universality results in terms of the complexity of used polynomials, both in the cell-like case and the tissue-like case, provided that the evolution programs are used in a parallel manner. However, two different types of parallelism were used in the two cases – hence the question: can the one-parallel mode (used for tissue P systems) be used also in the cell-like case?

Various extensions of "general" notions in membrane computing to numerical P systems remain to be examined – this is a rich research topic. For instance, other ways of using the programs can be considered: minimally parallel, with bounded parallelism, asynchronously. Then, we can also consider rules for handling membranes, such as membrane division and membrane creation. These operations are the basic tools by which polynomial solutions to computationally hard problems, typically, **NP**-complete problems, are obtained in the framework of P systems with symbol objects. Is this possible also for numerical P systems? This is a particularly interesting issue, both from the point of view of applications and because in this way we can achieve "fypercomputations", [7], in terms of numerical P systems.

Of course, a natural research topic is to further explore the use of numerical P systems in controlling robots, and to look for further applications where functions from $\mathbf{R}^{k_1}$ to $\mathbf{R}^{k_2}$ should be computed. In this framework an important question is to develop a complexity theory based on numerical P systems: define specific complexity classes, compare them with existing classes, look for ways to speed-up computations (see also the previous suggestion: to bring to numerical P systems further ideas investigated for symbol object P systems, in particular, tools to create an exponential working space in polynomial time).

I end with another natural question: defining dP systems, as in [9], with the components being numerical P systems. Can this be useful from the computation efficiency ("parallelization") point of view?

References

1. C. Buiu, C.I. Vasile, O. Arsene: Development of membrane controllers for mobile robots. *Information Sciences*, 187 (2012), 33–51.

2. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Research frontiers of membrane computing: Open problems and research topics. *Intern. J. Found. Computer Sci.*, in press (a preliminary version was published in the proceedings of *Tenth Brainstorming Week on Membrane Computing*, Sevilla, 2012, vol. II, 171–250).
3. Y. Matijasevitch: *Hilbert's Tenth Problem.* MIT Press, Cambridge, London, 1993.
4. A.B. Pavel, C. Buiu: Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, in press, DOI: 10.1007/s11047-011-9286-5, 2011
5. A.B. Pavel, O. Arsene, C. Buiu: Enzymatic numerical P systems – a new class of membrane computing systems. *The IEEE Fifth Intern. Conf. on Bio-Inspired Computing. Theory and applications. BIC-TA 2010*, Liverpool, Sept. 2010, 1331–1336.
6. A.B. Pavel, C.I. Vasile, I. Dumitrache: Robot localization implemented with enzymatic numerical P systems. *Proc. Living Machines 2012*, LNCS 7375, Springer, 2012, 204–215.
7. Gh. Păun: Towards "fypercomputations" (in membrane computing). *Languages Alive. Essays Dedicated to Jurgen Dassow on the Occasion of His 65 Birthday* (H. Bordihn, M. Kutrib, B. Truthe, etcs.), LNCS 7300, Springer, Berlin, 2012, 207–221.
8. Gh. Păun, R. Păun: Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, 73 (2006), 213–227.
9. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.
10. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing.* Oxford Univ. Press, 2010.
11. The P Systems Web Page: `http://ppage.psystems.eu`.
12. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: On the power of enzymatic numerical P systems. *Acta Informatica*, 49, 6 (2012), 395–412.
13. C.I. Vasile, A.B. Pavel, I. Dumitrache: Universality of enzymatic numerical P systems. *Intern. J. Computer Math.*, in press.
14. C.I. Vasile, A.B. Pavel, J. Kelemen: Implementing obstacle avoidance and follower behaviors on Koala robots using numerical P systems. *Tenth Brainstorming Week on Membrane Computing*, Sevilla, 2012, vol. II, 215–227.
15. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: Numerical P systems. Section 13 in [2].