Theoretical Computer Science 641 (2016) 85–108

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Numerical P systems with migrating variables

Zhiqiang Zhang [a], Tingfang Wu [a], Andrei Păun [b,c], Linqiang Pan [a,*]

[a] Key Laboratory of Image Information Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
[b] Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei nr. 14, Sector 1, C.P. Bucuresti, 010014, Romania
[c] Escuela Técnica Superior de Ingenieros Informáticos, Universitad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte 28660, Madrid, Spain

A R T I C L E I N F O

Article history:
Received 1 February 2016
Received in revised form 31 May 2016
Accepted 7 June 2016
Available online 9 June 2016
Communicated by T. Yokomori

Keywords:
Bio-inspired computing
Membrane computing
Numerical P system
Universality
Chomsky hierarchy

A B S T R A C T

Numerical P systems are a class of P systems inspired both from the structure of living cells and from economics, where variables are associated with the membranes, and these associations are not changed during the computation. However, in the standard P systems, a crucial character for objects is that they can pass through membranes, between regions of the same cell, between cells, or between cells and their environment. We introduce this character also to numerical P systems, and call the new variant numerical P systems with migrating variables (MNP systems). The computational power of MNP systems is investigated both as number generators and as string generators, working in the one-parallel or the sequential modes. Specially, as number generators, MNP systems are proved to be universal working in the above two modes. As string generators, the generative capacity of such systems is investigated having as a reference the families of languages in the Chomsky hierarchy, and a characterization of recursively enumerable languages is obtained.

1. Introduction

Membrane computing is a branch of natural computing aiming to abstract computing ideas and models from the membrane structure and the functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures. The computing models in membrane computing are called *membrane systems* or *P systems*. The basic idea is to consider a hierarchical arrangement of membranes, like in a cell, delimiting compartments where various chemicals (called *objects*) evolve according to local reaction rules. Since the first P systems introduced in [8], many variants of P systems have been proposed, such as [4,6,24,25,28], and most of these variants lead to computationally universal models [9,27]. Some applications of membrane computing in biology, linguistics, and theoretical and applied computer science can be found in [2]. The recent application in fault diagnosis of power systems can be found in [17,21,22,26].

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, the values of these variables evolve by *evolution programs*, which are inspired from economics. Each program is composed of a *production function* and a *repartition protocol*. The production function, taking the local variables, computes a

* Corresponding author.
E-mail address: lqpan@mail.hust.edu.cn (L. Pan).

http://dx.doi.org/10.1016/j.tcs.2016.06.004
0304-3975/© 2016 Elsevier B.V. All rights reserved.

number. According to the repartition protocol, this number is distributed among the local variables, the variables from the immediately upper compartment, and those from the immediately lower compartments. The values assumed by a distinguished variable is the set of numbers computed by the system. A special class of numerical P systems is that of *enzymatic ones*, where a control on the use of programs is considered [13]. This class was proved to be useful in designing robot controllers [14,16,15,23]. Both for numerical P systems and the enzymatic ones, the characterizations of Turing computable sets of numbers are obtained even for rather restricted forms of programs (e.g., for polynomials of low degrees as production functions) [5,11,19,20].

Extending "general" notions in membrane computing to numerical P systems is a topic of interest. One of the most important ingredients of P systems is communication of objects through the membranes. Many forms of communication have been investigated. The basic communication operators are *here*, *in*, *out*, which mean that the objects associated with them will remain in the same region, enter into one of the immediately lower membranes, go into the immediately upper membrane. Another form of communication based on a biochemical transport mechanism is called symport/antiport [10]. Symport indicates the biological process that two chemicals can pass through a membrane only together, in the same direction; while antiport indicates the biological process that two chemicals pass only with the help of each other, but in opposite directions.

In numerical P systems, however, the variables are confined to the membrane that they initially belong to, which is not changed during the computation. A natural and interesting question is what will happen if the communication of objects through the membranes is introduced to numerical P systems. We dwell on this question by considering *numerical P systems with migrating variables* (MNP systems, in short). In these systems, the variables are not associated with the membranes, but they can migrate from one membrane to another one indicated by the communication operators *here*, *in*, *out*. Besides changing the values of the variables, the programs also indicate the membranes where the variables have to move. (Of course, a program can be applied only when all the variables mentioned in the program appear in the membrane where the program resides.)

We consider two variants of MNP systems, with or without the so-called *non-zero, NZ, assumption*, that is, a program can be applied only when all the variables of its production function appear in the membrane where the program resides and have non-zero values. (In other words, a variable with zero value cannot activate any program.) We investigate the computational power of the two variants of numerical P systems both as number generators and as string generators. Specially, we prove that these two variants of MNP systems as number generators can reach universality. As string generators, the generative capacity of such systems is investigated having as a reference the families of languages in the Chomsky hierarchy, and a characterization of recursively enumerable languages is obtained.

Migrating variables are an efficient feature for improving the computational power of numerical P systems and the enzymatic variant in the sense of less membranes, lower polynomial degree and less variables in the universal systems. In [11] it is proved that sequential numerical P systems with at least 5 membranes and polynomial production functions of degree 5, each depending on at least 5 variables, can reach universality. While for MNP systems with the NZ assumption, the corresponding variables used in the universal systems are 2, 1, 3, respectively. Specially, it is hard to answer the open problem whether deterministically sequential numerical P systems are universal; however, we proved that MNP systems with the NZ assumption are universal. The following is a comparison with results obtained for enzymatic numerical P systems. In [30], the number of membranes used in the universal sequential enzymatic numerical P systems is at least 22, and the number of programs used to simulate the ADD instruction and the SUB instruction of register machine is 3 and 7, respectively. While for enzymatic numerical P systems with migrating variables the number of membranes is decreased to 2, and the number of programs used to simulate the ADD instruction and the SUB instruction is decreased to 2 and 5, respectively.

2. Preliminary definitions

Readers are assumed to be familiar with the basic elements of membrane computing, e.g., from [9,12], and formal language theory, as available in many monographs (e.g., in [18]). 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 non-terminal 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 $A \rightarrow aB$, $A \rightarrow a$, 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.