Theoretical Computer Science 567 (2015) 74–86

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computational efficiency and universality of timed P systems with active membranes [☆]

Bosheng Song, Linqiang Pan [*]

Key Laboratory of Image Information Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

ARTICLE INFO

Article history:
Received 2 May 2014
Received in revised form 15 September 2014
Accepted 31 October 2014
Available online 5 November 2014
Communicated by L. Kari

Keywords:
Membrane computing
P system
Time-free solution
Active membrane
SAT
Universality

ABSTRACT

P systems are a class of computational models inspired by the structure and the functioning of a living cell. In the semantics of P systems, there exists a global clock, which marks the time for the system, and the execution time of each rule takes exactly one time unit. However, in living cells, the execution time of different biochemical reactions is dependent on many uncontrollable factors, and it is hard to know precisely the specific execution time of a reaction. In this work, with this biological inspiration, we consider the class of P systems with active membranes that are "robust" against the execution time of rules. Specifically, we give a time-free uniform solution to the SAT problem using P systems with active membranes, where the constructed P system can solve a family of instances with an arbitrarily given size, and the execution time of the involved rules has no influence on the correctness of the solution. We also prove that any Turing computable set of numbers can be generated by a time-free P system with active membranes in the sense that the set of numbers generated by the given P system with active membranes does not depend on the execution time of rules.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Membrane computing is a vigorous branch of natural computing introduced by Gh. Păun [13]. The computation models in the field of membrane computing are known as *P systems*, which are parallel and distributed computation models. P systems are based upon the membrane structure of living cells and inspired from the cellular-level information processes involving different biochemical reactions, which can be viewed as computing processes. Roughly speaking, there are three main classes of P systems that have been considered until now: cell-like P systems [13], tissue-like P systems [7], neural-like P systems [6]. A recent coverage of membrane computing can be found in [16], and for the most up-to-date information for this area, readers can refer to the P systems web site: http://ppage.psystems.eu. The present work focuses on a class of cell-like P systems, called *P systems with active membranes*, introduced in [14].

P systems with active membranes were proved to be Turing universal by using the rules in various strategies, such as maximal parallelism, asynchronous, sequential, minimal parallelism [16]. P systems with active membranes were also used to solve computationally hard problems efficiently (in polynomial time or in linear time) [1,9–11] in the semantics that there

☆ This work was supported by National Natural Science Foundation of China (61033003, 91130034, and 61320106005), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072 and 20120142130008), and Natural Science Foundation of Hubei Province (2011CDA027).

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

http://dx.doi.org/10.1016/j.tcs.2014.10.051
0304-3975/© 2014 Elsevier B.V. All rights reserved.

exists a global clock, which marks time units, all regions evolve synchronously, and the execution time of each rule takes exactly one time unit. However, living cells can assume neither restrictions on the execution time nor the presence of global clocks synchronizing the execution of different parallel processes; moreover, the execution time of certain biological process could vary because of external uncontrollable conditions, such as biological signals, catalyst and medium temperature.

With this biological fact, *timed P systems* were proposed in [2], where an integer that represents the execution time is associated with each rule. *Time-free P systems* are a particular class of timed P systems in the sense that such P systems always generate (or accept) an identical family of vectors of natural numbers, independent of the value assigned to the execution time of each rule.

Time-free P systems were proved to be universal by simulating non-synchronized P systems [2]. In [12], *time-free spiking neural P systems* (SN P systems, for short) were investigated, where a time mapping of rules is added in general SN P systems to identify the execution time of the rules, and by simulating register machines, it was proved that such time-free systems can compute any set of Turing computable natural numbers.

Time-free solutions to computational hard problems were formulated as open problems in [5]. In [19], a family of P systems with active membranes was designed for a time-free solution to the SAT problem in the sense that the execution time of the involved rules has no influence on the correctness of the solution. The P systems designed in [19] were semi-uniform, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system from the instance. It was formulated as open problem how we can give a uniform time-free solution to the SAT problem in the sense that P systems are constructed from the size of instances of the problem (that is, a P system can solve a family of instances with an arbitrarily given size) [19].

In this work, we present a time-free solution to the SAT problem by a family of P systems with active membranes in a uniform way, which answers the corresponding open problem formulated in [19]. We also prove that any Turing computable set of numbers can be produced by a time-free P system with active membranes.

2. Preliminaries

2.1. Formal language theory

In this subsection, we only introduce some basic notions and notations from formal language theory. One can refer to [18] for more details.

An alphabet Σ is a finite and nonempty set of symbols. An ordered finite sequence of Σ forms a *string* (or *word*), which is obtained by juxtaposing symbols of Σ. The length of a string u, denoted by $|u|$, is the number of occurrences of symbols it contains. By $|u|_a$ we denote the number of occurrences of symbol a in u. The empty string is denoted by λ. The set of all strings over an alphabet Σ is denoted by Σ^* and by $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ we denote the set of all non-empty strings. A subset of Σ^* is called a *language* over Σ.

We denote by $\mathbb{N}$ the set of non-negative integers. A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset, then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$. A multiset is finite if its support is a finite set. If $m = (A, f)$ is a finite multiset over A, and $supp(m) = \{a_1, \dots, a_k\}$, then it will be denoted as $m = a_1^{f(a_1)} a_2^{f(a_2)} \dots a_k^{f(a_k)}$ or its permutations.

If $m_1 = (A, f_1)$, $m_2 = (A, f_2)$ are multisets over A, then we define the union of m_1 and m_2 as $m_1 + m_2 = (A, g)$, where $g(x) = f_1(x) + f_2(x)$.

2.2. Register machines

In the proof given in this work, we will use the characterization of *NRE* (the family of sets of numbers which are Turing computable) by means of *register machines* (also called counter machines, program machines, etc.).

Definition 1. A register machine is a tuple $M = (m, H, l_0, l_h, I)$, where:

- m is the number of registers;
- H is a set of labels;
- $l_0, l_h \in H$ are distinguished labels, where l_0 is the initial, and l_h is the halting one;
- I is a set of labeled program instructions of the following forms:
  - $l_i : (\mathtt{ADD}(r), l_j, l_k)$; add 1 to register r and continue with one of the instructions with labels l_j, l_k, non-deterministically chosen;
  - $l_i : (\mathtt{SUB}(r), l_j, l_k)$; if the contents of register r is greater than zero, subtract 1 from register r and continue with the instruction with the label l_j, otherwise the instruction with label l_k;
  - $l_h : \mathtt{HALT}$.

A register machine M generates a set $N(M)$ of numbers in the following way: the machine starts with all registers being empty (i.e., storing the number zero); the machine applies the instruction with label l_0 and continues to apply instructions