Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 378 (2007) 117–130
Theoretical Computer Science
www.elsevier.com/locate/tcs

P systems with minimal parallelism

Gabriel Ciobanu[a,b,*], Linqiang Pan[c], Gheorghe Păun[d,e], Mario J. Pérez-Jiménez[e]

[a] *Faculty of Computer Science, "A.I.Cuza" University, Blvd. Carol I no. 11, 700506 Iaşi, Romania*
[b] *Institute of Computer Science of the Romanian Academy, Iaşi, Romania*
[c] *Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China*
[d] *Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 Bucureşti, Romania*
[e] *Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain*

Received 18 November 2005; received in revised form 17 March 2007; accepted 21 March 2007

Communicated by T. Yokomori

Abstract

A current research topic in membrane computing is to find more realistic P systems from a biological point of view, and one target in this respect is to relax the condition of using the rules in a maximally parallel way. We contribute in this paper to this issue by considering the *minimal* parallelism of using the rules: if at least a rule from a set of rules associated with a membrane or a region *can* be used, then at least one rule from that membrane or region *must* be used, without any other restriction (e.g., more rules can be used, but we do not care how many). Weak as it might look, this minimal parallelism still leads to universality. We first prove this for the case of symport/antiport rules. The result is obtained both for generating and accepting P systems, in the latter case also for systems working deterministically. Then, we consider P systems with active membranes, and again the usual results are obtained: universality and the possibility to solve **NP**-complete problems in polynomial time (by trading space for time).

Keywords: Membrane computing; P System; Universality; SAT Problem

1. Introduction

A much debated topic in membrane computing (see monograph [14] for a comprehensive introduction, volume [4] for various applications, and the web site http://psystems.disco.unimib.it for recent developments) is how "realistic" various classes of P systems, hence various ingredients used in them, are from a biological point of view. Starting from the observation that there is an obvious parallelism in the cell biochemistry [1], and relying on the assumption that "if we wait enough, then all reactions which may take place will take place", a basic feature of the P systems as introduced in [12] is the maximally parallel way of using the rules (in each step, in each region of a system, we have to use a maximal multiset of rules). This condition provides a useful tool in proving various properties, because it decreases the non-determinism of the system's evolution, and it supports techniques for simulating the powerful features of

* Corresponding address: Institute of Computer Science, Formal Systems Laboratory, Blvd. Carol I, nr. 8, 700505 Iaşi, Romania. Tel.: +40 232 412820.
E-mail addresses: gabriel@info.uaic.ro, gabriel@iit.tuiasi.ro (G. Ciobanu), lqpan@mail.hust.edu.cn (L. Pan), george.paun@imar.ro, gpaun@us.es (G. Păun), marper@us.es (M.J. Pérez-Jiménez).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.03.044

appearance checking and zero testing, standard methods used to get computational completeness/universality of regulated rewriting in formal language theory [7] and of register machines [10], respectively.

For various reasons ranging from looking for more realistic models to just the mathematical challenge, the maximal parallelism was questioned, either simply criticized, or replaced with presumably less restrictive assumptions. For instance, sequential systems were considered in a series of papers, where only one rule is used in each step of a computation. An intermediate case is that of non-synchronized P systems, where any number of rules is used in each step. More details and references can be found in [8]. Various classes of systems with computations similar to the cooperation modes of grammar systems [5] are presented in [6]: in each step, in each region, for a given k, we can use exactly k, at least k, or at most k rules. In general, sequential systems and non-synchronized systems are weaker than systems based on maximal parallelism, but the universality is obtained again as soon as further control mechanisms are considered, such as priorities, bi-stable catalysts, etc.

In this paper we propose a rather natural condition which, as far as we know, was not yet considered: from each set of rules where at least a rule can be used, then at least one is actually used (maybe more, without any restriction). We say that this is a *minimal parallelism*, because this mode of using the rules ensures that all compartments of the system evolve in parallel, by using at least one rule whenever such a rule is applicable. We have two main cases given by the fact that the rules can be associated with a region of a P system (as in P systems with symbol-objects processed by rewriting-like rules), or they can be associated with membranes (as in symport/antiport P systems and in P systems with active membranes). The minimal parallelism refers to the fact that each such set is considered separately, and, if at least one rule of a set can be applied, then at least one rule must be applied. For systems with only one membrane the minimal parallelism is nothing else than non-synchronization, hence the non-trivial case is that of multi-membrane systems.

In what follows, we consider only two cases, that of P systems with symport/antiport rules, and of P systems with active membranes. Somewhat surprisingly, the universality is obtained again, in both cases. For instance, for symport/antiport systems, we again need a small number of membranes (three in the generative case, and two in the accepting case), while the symport and antiport rules are rather simple (of weight two). The universality is obtained both for systems working in the generative mode in which one collects the results of all halting computations, defined as the multiplicity of objects from an output membrane in the halting configuration, as well as in the accepting mode where a number is introduced in the system, in the form of the multiplicity of a specified object in a specified membrane, and the number is accepted if the computation halts. Moreover, the system can be deterministic in the accepting case.

Similarly, P systems with active membranes are universal even when using rules of only the types (a), (b), (c) (local evolution of objects, send-in, and send-out rules, respectively). These systems can also solve **NP**-complete problems in polynomial time — we exemplify this possibility by using the Boolean satisfiability problem (SAT). The results are as usual in the case of maximal parallelism, with the mention that we pay here a price for them: we use three polarizations in the universality proof, and division of non-elementary membranes in the efficiency proof; the construction for solving SAT is semi-uniform, and it works in a confluent way. The minimal parallelism for other classes of P systems, especially for catalytic systems, remains to be investigated.

In the next section we introduce the necessary language and automata theory prerequisites, Section 3 introduces the minimal parallelism for symport/antiport P systems, also giving their universality, both in the generative and accepting cases, Section 4 considers the minimal parallelism for P systems with active membranes, which are proved to be both universal and computationally efficient; we end the paper with some comments, in Section 5.

2. Prerequisites

We suppose that the reader is familiar with the basic elements of Turing computability [9], and of membrane computing [14]. We introduce here, in a rather informal way, only the necessary notions and notation.

An *alphabet* is a finite and non-empty set of abstract symbols. For an alphabet A, we denote by A^* the set of all strings of symbols from A including the empty string λ. A *multiset* over an alphabet A is a mapping from A to **N**, the set of natural numbers; we represent a multiset by a string from A^*, where the number of occurrences of a symbol $a \in A$ in a string w gives the multiplicity of a in the multiset associated with w (hence all strings obtained by permuting symbols in the string w represent the same multiset). The family of Turing computable sets of natural numbers is denoted by *NRE* (with RE coming from "recursively enumerable").

In our universality proofs we use the characterization of *NRE* by means of *register machines*. Such a device consists of a given number of registers, each of which can hold an arbitrarily large natural number, and a set of labelled instructions which specify how the numbers stored in registers can change, and which instruction should follow after any used instruction. There are three types of instructions:

- l_i : $(\mathtt{ADD}(r), l_j, l_k)$ add 1 to register r, and then go to one of the instructions labelled by l_j and l_k, non-deterministically chosen,
- l_i : $(\mathtt{SUB}(r), l_j, l_k)$ if register r is non-empty (non-zero), then subtract 1 from it and go to the instruction labelled by l_j, otherwise go to the instruction labelled by l_k,
- l_h : HALT the halt instruction.

A *register machine* is a construct $M = (n, B, l_0, l_h, R)$, where n is the number of registers, B is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to HALT only), and R is the set of instructions. Each label of B labels only one instruction from R, thus precisely identifying it. A register machine M generates a set $N(M)$ of numbers in the following way: having initially all registers empty (hence storing the number zero), start with the instruction labelled by l_0, and proceed to apply instructions as indicated by the labels and by the contents of registers. If we reach the halt instruction, then the number stored at that time in register 1 is said to be computed by M, and hence it is introduced in $N(M)$. Since we have a non-deterministic choice in the continuation of the computation in the case of ADD instructions, $N(M)$ can be an infinite set. It is known (see [10]) that in this way we can compute all the sets of numbers which are Turing computable, even using register machines with only three registers.

Without loss of generality, we may assume that we have $l_j \neq l_i$, $l_k \neq l_i$ whenever we use an instruction l_i : $(\mathtt{ADD}(r), l_j, l_k)$, and the unique instruction labelled by l_0 is an ADD instruction. Moreover, we may assume that the registers of the machine except register 1 are empty in the halting configuration.

A register machine can also work in an accepting mode. The number to be accepted is introduced in register 1, with all other registers empty. We start computing with the instruction labelled by l_0; if the computation halts, then the number is accepted (the contents of the registers in the halting configuration do not matter). We still denote by $N(M)$ the set of numbers accepted by a register machine M. In the accepting case, we can also consider deterministic register machines, where all the instructions l_i : $(\mathtt{ADD}, l_j, l_k)$ have $l_j = l_k$. It is known that deterministic accepting register machines characterize *NRE*.

3. Minimal parallelism for symport/antiport P systems

We first briefly recall the notion of P system with symport/antiport rules; for details, the reader is refereed to [14].

A *P system with symport/antiport rules* (of degree $n \geq 1$) is a construct of the form

$$\Pi = (O, \mu, w_1, \ldots, w_n, E, R_1, \ldots, R_n, i_o), \text{ where:}$$

(1) O is the alphabet of objects,
(2) μ is the membrane structure (of degree $n \geq 1$, with the membranes labelled in a one-to-one manner with $1, 2, \ldots, n$),
(3) $w_1, \ldots, w_n$ are strings over O representing the multisets of objects present in the n compartments of μ in the initial configuration of the system,
(4) $E \subseteq O$ is the set of objects supposed to appear in the environment in arbitrarily many copies,
(5) $R_1, \ldots, R_n$ are the (finite) sets of rules associated with the n membranes of μ,
(6) $i_o \in H$ is the label of a membrane of μ which indicates the *output* region of the system.

The rules of any set R_i can be of two types: *symport* rules of the forms (x, in), (x, out), and *antiport* rules of the form $(u, out; v, in)$, where x, u, v are strings over O. The length of x, respectively the maximum length of u, v, is called the *weight* of the corresponding (symport or antiport) rule.

In what follows, the rules are applied in a *non-deterministic minimally parallel* manner: in each step, from each set R_i $(1 \leq i \leq n)$ we use at least one rule (without specifying how many) provided that this is possible for a chosen selection of rules. The rules to be used, as well as the objects to which they are applied, are non-deterministically chosen. More specifically, we assign non-deterministically objects to rules, starting with one rule from each set R_i. If we cannot enable any rule for a certain set R_i, then this set remains idle. After having one rule enabled in each set

R_i for which this is possible, if we still have objects which can evolve, then we evolve them by any number of rules (possibly none).

We emphasize an important aspect: the competition for objects. It is possible that rules from two sets R_i, R_j associated with adjacent membranes i, j (that is, membranes which have access to a common region, either horizontally or vertically in the membrane structure) use the same objects, so that only rules from one of these sets can be used. Such conflicts are resolved in the following way: if possible, objects are assigned non-deterministically to a rule from one set, say R_i; then, if possible, other objects are assigned non-deterministically to a rule from R_j, and thus fulfilling the condition of minimal parallelism. After that, further rules from R_i or R_j can be used for the remaining objects, non-deterministically assigning objects to rules. If no rule from the other set (R_j, respectively R_i) can be used after assigning objects to a rule from R_i or R_j, then this is a correct choice of rules. It depends on the first assignment of objects to rules in order to make applicable rules from each set.

An example can illuminate this aspect: let us consider the configuration

$$[_k\, [_i\,]_i\, aab\, [_j\,]_j\,]_k,$$

with $R_i = \{(b, in)\}$, and $R_j = \{(a, in), (b, in)\}$. The objects aab are available to rules from both sets. All the following five assignments are correct: b goes to membrane i by means of the rule $(b, in) \in R_i$, and one or two copies of a is/are moved to membrane j by means of the rule $(a, in) \in R_j$; b goes to membrane j by means of the rule $(b, in) \in R_j$, no rule from R_i can be used, and zero, one, or two copies of a are also moved to membrane j by means of the rule $(a, in) \in R_j$. Moving only b to membrane i and not using at least once the rule $(a, in) \in R_j$ is not correct.

As usual in membrane computing, we consider successful only computations which halt, and with a halting computation we associate a result given by the number of objects present in the halting configuration in region i_o (note that this region is not necessarily an elementary one). External output or a squeezing through a terminal set of objects are also possible, but we do not consider these cases here. The set of numbers generated by a system Π in the way described above is denoted by $N_{gen}(\Pi)$. The family of all sets $N_{gen}(\Pi)$ generated by systems with at most $n \geq 1$ membranes, using symport rules of weight at most $p \geq 0$, and antiport rules of weight at most $q \geq 0$ is denoted by $N^{gen}_{min} OP_n(sym_p, anti_q)$, with the subscript "*min*" indicating the "minimal parallelism" used in computations.

We can also use a P system as above in the accepting mode: For a specified object a, we add a^m to the multiset of region i_o, and then we start working; if the computation stops, then the number m is accepted. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by a system Π, and by $N^{acc}_{min} OP_n(sym_p, anti_q)$ the family of all sets of numbers $N_{acc}(\Pi)$ accepted by systems with at most n membranes, using symport rules of weight at most p, and antiport rules of weight at most q.

In the accepting case we can consider deterministic systems, where at most one continuation is possible in each step. In our context, this means that zero or exactly one rule can be used from each set R_i, $1 \leq i \leq n$. We add the letter D in front of $N^{acc}_{min} OP_n(sym_p, anti_q)$ for denoting the corresponding families, $DN^{acc}_{min} OP_n(sym_p, anti_q)$.

Our expectation of introducing the minimal parallelism was to obtain a non-universal class of P systems, possibly with decidable properties. This turns out not to be true: the minimal parallelism still contains the possibility to enforce the checking for zero from SUB instructions of the register machines (this is the same with appearance checking from regulated rewriting), both for symport/antiport systems and for systems with active membranes.

Theorem 1. *$N^{gen}_{min} OP_n(sym_p, anti_q) = NRE$ for all $n \geq 3$, $p \geq 2$, $q \geq 2$.*

Proof. It is sufficient to prove the inclusion $NRE \subseteq N^{gen}_{min} OP_3(sym_2, anti_2)$; the inclusions $N^{gen}_{min} OP_n(sym_p, anti_q) \subseteq N^{gen}_{min} OP_{n'}(sym_{p'}, anti_{q'}) \subseteq NRE$, for all $1 \leq n \leq n'$, $0 \leq p \leq p'$, and $0 \leq q \leq q'$, are obvious.

Let us consider a register machine $M = (3, B, l_0, l_h, R)$; the fact that three registers suffice is not relevant in this proof. We consider an object $g(l)$ for each label $l \in B$, and denote by $g(B)$ the set of these objects. We define $B' = \{l' \mid l \in B\}$, $B'' = \{l'' \mid l \in B\}$, and $B''' = \{l''' \mid l \in B\}$, where l', l'', and l''' are new symbols associated with $l \in B$. For an arbitrary set of objects Q, let us denote by $w(Q)$ a string which contains exactly once each object from Q; thus $w(Q)$ represents the multiset which consists of exactly one occurrence of each object of Q. In general, for any alphabet Q we use the derived sets $Q' = \{b' \mid b \in Q\}$, $Q'' = \{b'' \mid b \in Q\}$, and $Q''' = \{b''' \mid b \in Q\}$, where b', b'', b''' are new symbols associated with $b \in Q$.

We construct a P system (of degree 3)

$$\Pi = (O, \mu, w_1, w_2, w_3, E, R_1, R_2, R_3, 3),$$

with

$$O = \{l, l', l'', l''', g(l) \mid l \in B\} \cup \{a_r \mid 1 \le r \le 3\} \cup \{c, d, t\},$$
$$\mu = [_1\,[_2\,[_3\;\;]_3\,]_2\,]_1,$$
$$w_1 = w(B - \{l_0\})w(B')w(B'')ct,$$
$$w_2 = w(g(B)),$$
$$w_3 = \lambda,$$
$$E = \{a_r \mid 1 \le r \le 3\} \cup B''' \cup g(B) \cup \{c, d, l_0\},$$

and with the following sets of rules[1]:

(1) **Starting the computation:**

R_1	R_2	R_3
$(c, out; ca_r, in),\ 1 \le r \le 3$	–	–
$(c, out; dl_0, in)$	–	–
–	(dl_0, in)	–

As long as c is present in region 1, we can bring copies of objects a_r $(1 \le r \le 3)$ in the system by using rule $(c, out; ca_r, in) \in R_1$. The number of copies of a_r from region 2 represents the number stored in register r of the register machine. At any moment we can use a rule $(c, out; dl_0, in)$ which brings the initial label of M inside (note that l_0 does not appear in w_1), together with the additional object d. From now on, no further object a_r can be brought in the system. In the next step, the objects dl_0 can enter region 2 by means of rule $(dl_0, in) \in R_2$, and this initiates the simulation of a computation in our register machine M.

(2) We also have the following **additional rules** which are used in interaction with rules from the other groups, including the previous group:

R_1	R_2	R_3
(l_0, out)	$(g(l), out; dt, in),\ l \in B$	–
–		$(t, out), (t, in)$

The role of these rules is to prevent "wrong" steps in Π. For instance, as soon as the instruction with label l_0 is simulated (see details below), label l_0 should be sent out of the system in order to avoid another computation in M, and this is done by means of rule $(l_0, out) \in R_1$. This rule should be used as soon as l_0 is available in region 1, and no other rule from R_1 may be applied — we will see immediately that this is ensured by the rules simulating the instructions of M. However, l_0 should not be eliminated prematurely, i.e., before simulating the instruction with label l_0. In such a case, a rule $(g(l), out; dt, in)$ for some $l \in B$ should be used once and only once, because we have only one available copy of d. In this way, the object t enters membrane 2, and the computation can never stop because of the rules $(t, out), (t, in) \in R_3$. These trap-rules are used also in order to control the correct simulation of instructions of M.

(3) **Simulating an instruction** $l_i : (\mathtt{ADD}(r), l_j, l_k)$:

Step	R_1	R_2	R_3
1	–	$(l_i, out; l_p a_r, in),\ p = j, k$, or $(l_i, out; t, in)$	–

When l_i is in region 2, the number of copies of a_r is incremented by bringing a_r from the skin region, together with any of labels l_j and l_k. Since we have provided in w_2 one copy of any label different from l_0, labels l_j and l_k are present in region 1. We have assumed that each ADD instruction of M has $l_j \neq l_i$ and $l_k \neq l_i$, hence we do not need two copies of any $l \in B$ in Π. If we have not enough copies of a_r because the initial phase was concluded prematurely (we have brought some copies of a_r in the system, but all of them were already moved in membrane 2), then we cannot use a rule $(l_i, out; l_p a_r, in)$, $p = j, k$. Because of the minimal parallelism, we should use the rule $(l_i, out; t, in)$ (no other rule of R_2 is applicable). Therefore, the computation will never end, because of the

[1] We present the rules in groups associated with various tasks during the computation, also indicating the steps when they are used.

trap-rules (t, out), $(t, in) \in R_3$. Note that no rule from R_1 can be used, excepting (l_0, out), whenever l_0 is present in the skin region (which means that the initial instruction was already simulated).

(4) **Simulating an instruction** $l_i : (\mathtt{SUB}(r), l_j, l_k)$:

Step	R_1	R_2	R_3
1	–	$(g(l_i)l_i, out; l_i', in)$	–
2	$(g(l_i), out; g(l_i)l_i''', in)$	$(l_i'a_r, out; l_i'', in)$	–
3	–	$(g(l_i)l_i''', in)$	–
4	–	$(l_i'''l_i', out; l_k, in)$, $(l_i'''l_i'', out; l_j, in)$	–
1	(l_i''', out)	·····	–
–	–	$(d, out; l_i'''t, in)$	–

With l_i present in region 2, we use a rule $(g(l_i)l_i, out; l_i', in) \in R_2$ by which the witness object $g(l_i)$ goes to the skin region, and the object l_i' is brought into region 2. During the next step, this latter object checks whether there is a copy of a_r present in region 2. In a positive case, rule $(l_i'a_r, out; l_i'', in) \in R_2$ is used in parallel with $(g(l_i), out; g(l_i)l_i''', in) \in R_1$. This last rule brings into the system a "checking" symbol l_i'''.

In the next step we have two possibilities.

If we use a rule $(g(l_i)l_i''', in) \in R_2$, then both $g(l_i)$ and l_i''' are introduced in region 2. Depending on what l_i''' finds here, one can use either $(l_i'''l_i', out; l_k, in)$ if the subtraction was not possible (because no a_r was present), or $(l_i'''l_i'', out; l_j, in)$ if the subtraction was possible. Thus, the correct label selected between l_j and l_k is brought into region 2.

If in step 3, instead of a rule $(g(l_i)l_i''', in) \in R_2$, we use again $(g(l_i), out; g(l_i)l_i''', in) \in R_1$, then the existing object l_i''' either exits the system by means of the rule $(l_i''', out) \in R_1$ and so we repeat the configuration, or we use the rule $(d, out; tl_i''', in) \in R_2$ and the computation never stops (no other rule of R_2 can be used). Similarly, if in step 3 we use the rule $(l_i''', out) \in R_1$, then at the same time the object $g(l_i)$ from region 1 either remains unchanged, or uses the rule $(g(l_i), out; g(l_i)l_i''', in) \in R_1$. In the former case, in the next step we have to use the rule $(g(l_i), out; g(l_i)l_i''', in) \in R_1$ (no other rule of R_1 can be applied). In both cases, the configuration is repeated. Of course, if we directly use the rule $(d, out; tl_i''', in) \in R_2$, then the computation never stops.

Thus, in all these cases, either the simulation of the SUB instruction is correct or the system never stops. In the next step, namely step 1 of the simulation of any type of instruction, the object l_i''' exits the system by means of the rule $(l_i''', out) \in R_1$. Note that no other rule of R_1 is used in this step, neither for ADD, nor for SUB instructions.

Consequently, both ADD and SUB instructions of M are correctly simulated. The configuration of the system is restored after each simulation, namely all the objects of $g(B)$ are in region 1, and no object of B''' is present in the system. The simulation of the instructions of M can be iterated, and thus we can simulate in Π a computation of M. If the computation of M does not stop, then the corresponding simulation in Π does not stop as well. When the computation in M stops, then the label l_h is introduced in region 2, and registers 2 and 3 are empty. Then we use the following group of rules.

(5) **Final rules:**

R_1	R_2	R_3
–	–	(l_ha_1, in)
–	–	(l_h, out)

The halting label l_h carries all the copies of a_1 from region 2 to 3, and only when this process is completed the computation can stop.

Consequently, the system Π correctly simulates (all and nothing else than) the computations of M. This means that $N_{gen}(\Pi) = N(M)$. □

Note that in this proof the output region is an elementary one, and that the minimal parallelism is essentially used in ensuring the correct simulation of the computations in the register machine M (for instance, the minimal parallelism prevents halting the computation in Π without completing the simulation of a halting computation in M).

The previous proof can be easily adapted for the accepting case: we consider l_0 already in region 1, and introduce additionally a_1^m, for m being the number we want to accept. Omitting the technical details, we mention the following result:

Corollary 2. $N^{acc}_{min}OP_n(sym_p, anti_q) = NRE$ *for all* $n \geq 3,\ p \geq 2,\ q \geq 2$.

Moreover, this last result can be strengthened by considering deterministic P systems, and also by decreasing by one the number of membranes.

Theorem 3. $DN^{acc}_{min}OP_n(sym_p, anti_q) = NRE$ *for all* $n \geq 2,\ p \geq 2,\ q \geq 2$.

Proof. Let us consider a deterministic register machine $M = (3, B, l_0, l_h, R)$ accepting an arbitrary set $N(M) \in NRE$. We again use the notation $w(Q)$ as specified in the proof of Theorem 1, and Q^u for the sets $\{b^u \mid b \in Q\}$, where u denotes various markings (priming). We construct a P system (of degree 2)

$$\Pi = (O, [_1\, [_2\,]_2\,]_1, w_1, w_2, E, R_1, R_2, 1),$$

with

$$O = \{l, l', l'', l''', l^{iv}, l^v \mid l \in B\} \cup \{a_r \mid 1 \leq r \leq 3\} \cup \{c\},$$
$$w_1 = l_0,$$
$$w_2 = w(B^{iv}),$$
$$E = O,$$

and with the following sets of rules:

(1) **Simulating an instruction** $l_i : (\mathtt{ADD}(r), l_j, l_j)$:

Step	R_1	R_2
1	$(l_i, out; l_j a_r, in)$	–

Since the environment contains the necessary objects, the ADD instruction is simulated whenever we have the object l_i in region 1.

(2) **Simulating an instruction** $l_i : (\mathtt{SUB}(r), l_j, l_k)$:

Step	R_1	R_2
1	$(l_i, out; l_i' l_i'', in)$	–
2	$(l_i' a_r, out; l_i''', in)$	$(l_i^{iv}, out; l_i'', in)$
3	$(l_i^{iv} l_i', out; l_i^{iv} l_k^v, in)$ or $(l_i^{iv} l_i'', out; l_i^{iv} l_j^v, in)$	–
4	$(l_k^v, out; cl_k, in)$, resp. $(l_j^v, out; cl_j, in)$	–
1	.....	(cl_i^{iv}, in)

With l_i present in region 1, we bring into the system the objects l_i', l_i'' from the inexhaustible environment. The object l_i' is used to subtract 1 from register r by leaving together with a copy of a_r. This operation brings into the system an object l_i'''; simultaneously, l_i'' enters region 2, releasing the "checking" symbol l_i^{iv}. In the next step, depending on what l_i^{iv} finds in membrane 1, either $(l_i^{iv} l_i', out; l_i^{iv} l_k^v, in)$ is used whenever the subtraction was not possible, or $(l_i^{iv} l_i'', out; l_i^{iv} l_j^v, in)$ is used if the subtraction was possible. The corresponding l_j^v or l_k^v is brought into the system. In the fourth step, these last objects exit the system and bring inside their corresponding non-marked objects l_j and l_k, together with an auxiliary object c. In the next step, object c helps object l_i^{iv} (which waits one step in the skin region) to return to region 2, and so making sure that the multiset of region 2 contains again all objects of B^{iv}. This step is the first one in simulating another ADD or SUB instruction, and it is important to note that in this step no other rule from R_2 can be used.

The computation starts by introducing in region 1 a multiset a_1^m encoding the number m. Note that the initial label of M is already present here. By iterating the simulations of ADD and SUB instructions, we can simulate any computation of M. Obviously, since any computation of M is deterministic, the computation in Π is also deterministic (from each set R_i, in each step, at most one rule is applicable). When the halt instruction is reached, hence l_h appears in the system, the computation stops (with a further last step when the rule $(cl_i^{iv}, in) \in R_2$ is used).

We conclude that $N_{acc}(\Pi) = N(M)$. □

Note that the deterministic construction is simpler than that from the proof of Theorem 1, because we should not initially bring "resources" into the system, and we have nothing to do in the final stage (in particular, we should not "clean" the output region of any other objects than those we need to count).

4. Minimal parallelism for P systems with active membranes

Let us consider now another important class of P systems, that with active membranes. We start by briefly recalling the basic definition, then we define the minimal parallelism for this type of P systems.

A *P system with active membranes* of the initial degree $n \geq 1$ is a construct of the form

$$\Pi = (O, H, \mu, w_1, \ldots, w_n, R, h_o),$$

where:

(1) O is the alphabet of *objects*;
(2) H is a finite set of *labels* for membranes;
(3) μ is a *membrane structure*, consisting of n membranes having initially neutral polarizations, labelled (not necessarily in a one-to-one manner) with elements of H;
(4) $w_1, \ldots, w_n$ are strings over O, describing the *multisets of objects* placed in the n initial regions of μ;
(5) R is a finite set of *developmental rules*, of the following forms:
   (a) $[_h a \to v]_h^e$,
   for $h \in H, e \in \{+, -, 0\}, a \in O, v \in O^*$
   (*object evolution* rules, associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);
   (b) $a[_h\,]_h^{e_1} \to [_h b]_h^{e_2}$,
   for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
   (*in* communication rules; a possibly modified object is introduced in a membrane; the polarization of the membrane can also be modified, but not its label);
   (c) $[_h a\,]_h^{e_1} \to [_h\,]_h^{e_2} b$,
   for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
   (*out* communication rules; a possibly modified object is sent out of the membrane; the polarization of the membrane can also be modified, but not its label);
   (d) $[_h a\,]_h^e \to b$,
   for $h \in H, e \in \{+, -, 0\}, a, b \in O$
   (*dissolving* rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);
   (e) $[_h a\,]_h^{e_1} \to [_h b\,]_h^{e_2} [_h c\,]_h^{e_3}$,
   for $h \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$
   (*division* rules for elementary or non-elementary membranes; in reaction with an object, the membrane with label h is divided into two membranes with the same label, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects; the remaining objects may evolve in the same step by rules of type (a) and the result of this evolution is duplicated in the two new membranes; if membrane h contains other membranes, then they may evolve at the same time by rules of any type and the result of their evolution is duplicated in the two new membranes);
(6) $h_o \in H$ is the label of the output membrane.

Note that, different from [13,14], etc., the rules of type (e) can be applied also to non-elementary membranes. The rules for dividing non-elementary membranes are of a different form in the literature (in several papers the division takes place only when internal membranes of opposite polarizations are present); in this paper we consider the above uniform form of rules for dividing membranes irrespective of whether they are elementary or not.

We use here the rules in a minimally parallel manner, in the following sense. All the rules of any type involving a membrane h form the set R_h (this means all the rules of type (a) of the form $[_h a \to v]_h^e$, all the rules of type (b) of the

form $a[_h\,]_h^{e_1} \to [_h b]_h^{e_2}$, and all the rules of types (c)–(e) of the form $[_h a]_h^e \to z$, with the same h, constitute the set R_h). Moreover, if a membrane h appears several times in a given configuration of the system, then for each occurrence of the membrane we consider a different set R_h; this means that we identify the ith copy of membrane h with the pair (h, i), and we consider the set of rules $R_{h,i} = R_h$. Then, in each step, from each set $R_{h,i}$, $h \in H$ (possibly, without specifying $i = 1$ in the case where membrane h appears only once) from which at least a rule *can* be used, at least one rule *must* be used.

In what follows, in order to make visible the sets of rules, we write explicitly the sets R_h, $h \in H$, instead of the global set R.

Of course, as usual for P systems with active membranes, each membrane and each object can be involved in only one rule, and the choice of rules to use and of objects and membranes to evolve is done in a non-deterministic way. We should note that for rules of type (a) the membrane is not considered to be involved: when applying $[_h a \to v]_h$, the object a cannot be used by other rules, but the membrane h can be used by any number of rules of type (a) as well as by one rule of types (b)–(e). In each step, the use of rules is done in the bottom-up manner (first the inner objects and membranes evolve, and the result is duplicated if any surrounding membrane is divided).

A halting computation provides a result given by the number of objects present in membrane h_o at the end of the computation; for a computation to be successful, exactly one membrane with label h_o should be present in the halting configuration. The set of numbers generated in this way by a system Π is denoted by $N(\Pi)$, and the family of these sets, generated by systems having initially at most n_1 membranes and using configurations with at most n_2 membranes during the computation is denoted by $N_{min}OP_{n_1,n_2}$((a)–(e)); when a type of rules is not used, it is not mentioned in the notation.

We prove now the universality of P systems with active membranes working in the minimally parallel mode; as it is the case also for the maximally parallel mode, only rules of the first three types are used — however, in contrast to [3], we use three polarizations. Actually, in the maximally parallel case the universality of P systems with active membranes can be obtained even without using polarizations (see [2]) at the price of using rules of all the five types.

Theorem 4. *$N_{min}OP_{n_1,n_2}$((a)–(c)) = NRE for all $n_1 \geq 4$, $n_2 \geq 4$.*

Proof. Let us consider a register machine $M = (3, B, l_0, l_h, R)$ generating an arbitrary set $N(M) \in NRE$. We construct a P system (of the initial degree 4)

$$\Pi = (O, H, \mu, l_0, \lambda, \lambda, \lambda, R_0, R_1, R_2, R_3, 1),$$

with

$$O = \{a\} \cup \{l, l', l'', l''', l^{iv}, l^v, l^{vi}, l^{vii}, l^{viii}, l^{ix} \mid l \in B\},$$
$$H = \{0, 1, 2, 3\},$$
$$\mu = [_0[_1\,]_1\,[_2\,]_2\,[_3\,]_3]_0,$$

and with the following rules[2]:

(1) **Simulating an instruction** $l_i : (\mathtt{ADD}(r), l_j, l_k)$, for $r = 1$, 2, or 3:

Step	R_0	R_r
1	–	$l_i[_r\,]_r^0 \to [_r l_i]_r^0$
2	–	$[_r l_i \to l'_p a]_r^0,\ p = j, k$
3	–	$[_r l'_p]_r^0 \to [_r\,]_r^0 l'_p,\ p = j, k$
4	$[_0 l'_p \to l_p]_0^0,\ p = j, k$	–

The label-object l_i enters into the correct membrane r, produces one further copy of a, and the primed labels l_j, l_k inside membrane r; the labels l_j, l_k exit to the skin region and lose their primes, and the process can be iterated. Note that in each step, at most one rule from each set R_0, R_1, R_2, R_3 can be used.

[2] A membrane with label r is defined for each register r; the number stored in a register r is represented by the number of objects a present in membrane r.

(2) **Simulating an instruction** $l_i : (\mathtt{SUB}(r), l_j, l_k)$, for $r = 1, 2, 3$ (we mention the rules to be applied in each step, if their application is possible):

Step	R_0	R_r
1	$[_0 l_i \to l_i' l_i'']_0^0$	–
2	$[_0 l_i'' \to l_i''']_0^0$	$l_i'[_r\,]_r^0 \to [_r l_i']_r^+$
3	$[_0 l_i''' \to l_i^{iv}]_0^0$	$[_r a]_r^+ \to [_r\,]_r^- a$
4	$[_0 l_i^{iv} \to l_i^v]_0^0$	$[_r l_i' \to l_j^{viii}]_r^-$
5	$[_0 l_i^v \to l_i^{vi}]_0^0$	$[_r l_j^{viii}]_r^- \to [_r\,]_r^0 l_j^{viii}$
6	$[_0 l_j^{viii} \to l_j^{ix}]_0^0$	$l_i^{vi}[_r\,]_r^0 \to [_r l_i^{vi}]_r^0, l_i^{vi}[_r\,]_r^+ \to [_r l_i^{vi}]_r^+$
7	$[_0 l_j^{ix} \to l_j]_1^0$	$[_r l_i^{vi} \to \lambda]_r^0, [_r l_i^{vi} \to l_i^{vii}]_r^+$
8	–	$[_r l_i^{vii}]_r^+ \to [_r\,]_r^0 l_i^{vii}$
9	$[_0 l_i^{vii} \to l_k]_0^0$	$[_r l_i' \to \lambda]_r^0$

We start by introducing l_i', l_i'' in the skin region; the former object enters membrane r associated with the rule to simulate, changing its polarization to $+$, the latter object evolves in the skin region to l_i'''. In the third step, while l_i''' changes to l_i^{iv} in the skin region, a copy of object a can exit membrane r, providing that such an object exists; this changes the polarization of this membrane to negative.

Let us consider separately the two possible cases.

If we have inside membrane r the object l_i', and the membrane is positively charged (which means that the contents of register r was empty), then in steps 4 and 5 we can apply rules only in the skin region, adding primes to the primed l_i present here until reaching l_i^{vi}. In step 6, this object enters membrane r, evolves (by step 7) to l_j^{vii} which exits membrane 2 in step 8, returning its polarization to neutral. In these circumstances, in step 9 l_i^{vii} introduces l_k in the skin region (this is the correct label-object for this case), and simultaneously l_i' (waiting in membrane r from the beginning of the simulation) is removed by using the rule $[_r l_i' \to \lambda]_r^0$ from R_r (l_i' cannot be removed earlier, because the polarization of membrane r was positive).

If in step 3 the inner membrane gets a negative charge (which means that the subtraction from register r was possible), then we can continue with the rules mentioned in the table above for step 4: l_i' introduces l_j^{viii}, which later will introduce l_j. In parallel, in the skin region we use the rule $[_0 l_i^{iv} \to l_i^v]_0^0$.

The next step is similar: we add one more prime to the object from the skin region, and the rule $[_r l_j^{viii}]_r^- \to [_r\,]_r^0 l_j^{viii}$ of R_r is used, thus restoring the polarization of membrane r to neutral. In step 6, the object l_j^{viii}, just expelled from membrane r, introduces l_j^{ix} in the skin region, while the "witness" l_i^{vi} enters into membrane r (now, neutral again). In step 7, l_j^{ix} introduces the correct label-object l_j in the skin region, and simultaneously l_i^{vi} is removed from membrane r. The simulation of the SUB instruction is completed.

In both cases, the simulation of the SUB instruction is correct, and the process can be iterated. Again, in each step, from each set R_0, R_1, R_2, R_3 at most one rule can be used (hence the minimal parallelism is, in this case, similar to the maximal parallelism).

We start with label l_0 in the skin region. If the computation in M halts, hence l_h is reached, this means that the object l_h is introduced in the skin region, and also the computation in Π halts. Consequently, $N(M) = N(\Pi)$, and this concludes the proof. □

It remains as an open problem whether Theorem 4 can be improved by removing the polarizations of membranes and/or by decreasing the number of membranes.

The previous system simulates the starting register machine in a deterministic manner, hence the result of the theorem remains true also for deterministic accepting P systems. We leave the obvious details as a task for the reader, and we pass to the important issue of using the membrane division as a tool of generating an exponential workspace in a linear time, and then using this space for addressing computationally hard problems. This possibility is successfully used for many problems in the maximal parallelism (we refer to [15] for details and references), and this remains pleasantly true for the minimal parallelism, at least for the SAT problem.

The framework in which we prove this assertion is the following: we start from a given instance of SAT (using n variables and having m clauses), and we construct, in polynomial time, a P system Π with active membranes of a size

polynomially bounded with respect to n and m, whose all computations halts in a polynomial time with respect to n and m; moreover, all the computations send to the environment either a special object `yes` or a special object `no`, and this happens in the last step. The propositional formula is satisfiable if and only if the object `yes` is released.

Therefore, in the rigorous terminology of [15], we construct the system in a semi-uniform manner (starting from a given instance of the problem), and the system works in a weakly confluent way (we allow the system to evolve non-deterministically, but we impose that all the computations halt, and all of them provide the same answer). Improvements from these points of view (looking for uniform constructions, of deterministic systems) are left as open problems. In comparison with the existing similar solutions to **NP**-complete problems from the membrane computing literature based on the maximal parallelism, we pay here a price for using the minimal parallelism: namely, we use the membrane division for non-elementary membranes. The issue whether or not the membrane division for only elementary membranes suffices remains also as an open problem.

Theorem 5. *The satisfiability of any propositional formula in the conjunctive normal form, using n variables and m clauses, can be decided in a linear time with respect to n by a P system with active membranes using rules of types* (a)–(e)*, and working in the minimally parallel mode. Moreover, the system is constructed in a linear time with respect to n and m.*

Proof. Let us consider a propositional formula $\sigma = C_1 \wedge \cdots \wedge C_m$, such that each clause C_i, $1 \leq i \leq m$, is of the form $C_i = y_{i,1} \vee \cdots \vee y_{i,k_i}$, $k_i \geq 1$, for $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$. For each $k = 1, 2, \ldots, n$, let us denote

$$t(x_k) = \{c_i \mid \text{there is } 1 \leq j \leq k_i \text{ such that } y_{i,j} = x_k\},$$
$$f(x_k) = \{c_i \mid \text{there is } 1 \leq j \leq k_i \text{ such that } y_{i,j} = \neg x_k\}.$$

These are the sets of clauses which assume the value *true* when x_i is *true*, respectively, *false*, respectively.

We construct a P system Π with the following components (the output membrane is not necessary, because the result is obtained in the environment):

$$\begin{aligned}
O &= \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \\
&\cup \{c_i \mid 1 \leq i \leq m\} \\
&\cup \{g_i \mid 1 \leq i \leq 2n+6\} \\
&\cup \{e_i \mid 1 \leq i \leq 2n+1\} \\
&\cup \{b, c, \mathtt{yes}, \mathtt{no}\}, \\
H &= \{s, e, g, 0, 1, 2, \ldots, m\}, \\
\mu &= [_s[_g\,]_g[_0[_e\,]_e[_1\,]_1[_2\,]_2 \cdots [_m\,]_m]_0]_s, \\
w_g &= g_1,\ w_e = e_1, w_0 = a_1, \\
w_s &= w_i = \lambda,\ \text{for all } i = 1, 2, \ldots, m,
\end{aligned}$$

and with the following rules (we present them in groups with specific tasks, also commenting their use).

For the sake of readability, the initial configuration is given in Fig. 1; initially, all the membranes have neutral polarizations.

The membranes with labels g and e, with the corresponding objects g_i and e_i, respectively, are used as counters which evolve simultaneously with the "main membrane" 0 where the truth assignments of the n variables $x_1, \ldots, x_n$ are generated; the use of separate membranes for counters makes possible the correct synchronization even for the case of the minimal parallelism. The evolution of counters is done by the following rules:

$$[_g g_i \to g_{i+1}]^0_g,\ \text{for all } 1 \leq i \leq 2n+5,$$
$$[_e e_i \to e_{i+1}]^0_e,\ \text{for all } 1 \leq i \leq 2n.$$

In parallel, membrane 0 evolves by means of the following rules:

$$[_0 a_i]^0_0 \to [_0 t_i]^0_0 [_0 f_i]^0_0,\ \text{for all } 1 \leq i \leq n,$$
$$[_0 t_i \to t(x_i) a_{i+1}]^0_0,\ \text{and}$$
$$[_0 f_i \to f(x_i) a_{i+1}]^0_0,\ \text{for all } 1 \leq i \leq n-1,$$

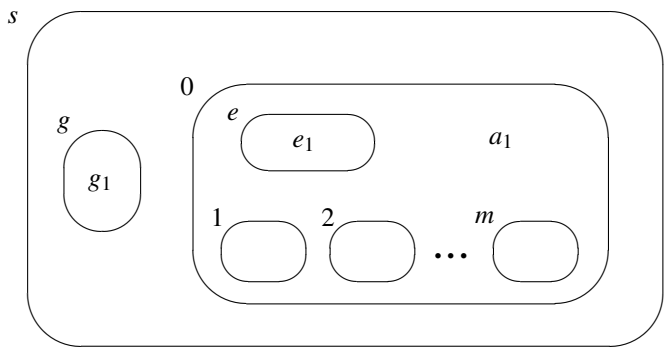

Fig. 1. The initial configuration of the system from the proof of Theorem 5.

$$[_0 t_n \to t(x_n)]_0^0,$$
$$[_0 f_n \to f(x_n)]_0^0.$$

In odd steps, we divide the (non-elementary) membrane 0 (with t_i, f_i corresponding to the truth values *true* and *false* for variable x_i); in even steps we introduce the clauses satisfied by x_i and $\neg x_i$, respectively. When we divide membrane 0, all its inner objects and membranes are replicated; in particular, all the membranes with labels $1, 2, \ldots, m$, as well as membrane e are replicated, hence they are present in all the membranes with label 0.
This process lasts $2n$ steps. At the end of this phase, all 2^n truth assignments for the n variables are generated.

In parallel with the division steps, if a clause C_j is satisfied by the previously expanded variable, then the corresponding object c_j enters membrane j by means of the rule

$$c_j[_j\,]_j^0 \to [_j c_j]_j^+,\ 1 \le j \le m,$$

thus changing the polarization of this membrane to positive.

This is done also in step $2n+1$, in parallel with the following rule for evolving membrane e:

$$[_e e_{2n+1}]_e^0 \to [_e\,]_e^0 e_{2n+1}.$$

Thus, after $2n+1$ steps, the configuration of the system consists of 2^n copies of membrane 0, each of them containing the membrane e empty, the object e_{2n+1}, possible objects c_j, $1 \le j \le m$, as well as copies of all the membranes with labels $1, 2, \ldots, m$; these membranes are either neutrally charged (if the corresponding clause was not satisfied by the truth assignments generated in that copy of membrane 0), or positively charged (in the case where the respective clause was satisfied — and hence one copy of c_j is inside membrane j). Therefore, formula σ is satisfied if and only if there is a membrane 0 where all membranes $1, 2, \ldots, m$ are positively charged. In order to check this last condition, we proceed as follows.

In step $2n+2$ we use the rule

$$[_0 e_{2n+1} \to bc]_0^0,$$

which introduces the objects b and c in each membrane 0. The latter object exits the membrane in step $2n+3$ by using the rule

$$[_0 c]_0^0 \to [_0\,]_0^+ c,$$

thus changing the polarization of the membrane to positive. Simultaneously, b enters any of the membranes $1, 2, \ldots, m$ with neutral polarization, if such a membrane exists, by means of the rule

$$b[_j\,]_j^0 \to [_j b]_j^0,\ 1 \le j \le m.$$

Therefore, if b can "get hidden" in a membrane j, this means that the truth assignment from the respective membrane 0 has not satisfied all clauses. In the opposite case, where all clauses were satisfied (hence all the membranes $1, 2, \ldots, m$ have positive polarizations), the object b waits one step in membrane 0.

If object b is not blocked in an inner membrane, then in the next step $(2n + 4)$ it exits membrane 0, by means of the rule

$$[_0 b]_0^+ \to [_0\,]_0^+ b.$$

This was not possible in the previous step, because the membrane had the neutral polarization.

In the next two steps we use the rules

$$[_s b \to \texttt{yes}]_s^0,$$
$$[_s \texttt{yes}]_s^0 \to [_s\,]_s^+ \texttt{yes},$$

hence the object yes is sent to the environment in step $2n + 6$, signalling the fact that the formula is satisfiable.

If object b is blocked in an inner membrane in all the copies of membrane 0 (hence the propositional formula is not satisfiable), then no b is evolving in steps $2n + 4$ and $2n + 5$; in these steps, the counter g increases its subscript, hence in step $2n + 6$ we can use the rule

$$[_g g_{2n+6}]_g^0 \to [_g\,]_g^0 g_{2n+6}.$$

This rule is also used when the formula is satisfied; however in that situation the skin membrane is changed to a positive polarization in step $2n + 6$, and so the following rule cannot be used in step $2n + 7$. If the formula is not satisfied, hence the skin membrane keeps its initial neutral polarization, then in step $2n + 7$ we can use the rule

$$[_s g_{2n+6} \to \texttt{no}]_s^0,$$

and the object no is sent to the environment by means of the rule

$$[_s \texttt{no}]_s^0 \to [_s\,]_s^+ \texttt{no}.$$

Therefore, if the formula is satisfiable, then the object yes exits the system in step $2n + 6$, and if the formula is not satisfiable, then the object no exits the system in step $2n + 8$. In both cases, the computation halts.

The system Π uses $7n + m + 11$ objects, $m + 4$ initial membranes (containing 3 objects at the beginning of the computation), and $7n + 2m + 14$ rules, all of these rules being of a length which is linearly bounded with respect to m (hence, the system can be constructed effectively in a time linearly bounded with respect to n and m). Clearly, all the computations stop after at most $2n + 8$ steps, hence in a time linear with respect to n, and all give the same answer, yes or no, to the question whether formula σ is satisfiable, hence the system is weakly confluent. These observations conclude the proof. □

Note also that the previous construction does not use rules of type (d), for membrane dissolution, and that the time for deciding whether a formula is satisfiable does not depend on the number of clauses, but only on the number of variables.

5. Final remarks

The minimal parallelism remains to be considered for other classes of P systems, especially for systems with catalytic rules. From a mathematical point of view, it is also of interest to look for improvements of the results presented in Theorems 1, 3 and 4 in what concerns the number of membranes in the first and third theorems, and the weights of symport and antiport rules in the first two theorems — as it is done in a series of papers for the case of maximal parallelism. It also remains to be determined which of the results known for the maximal parallelism are also valid for the minimal parallelism, and which results can be re-proven in the new framework.

Returning to the initial motivation of this research, a very interesting problem is to find mechanisms of "keeping under control" the power of P systems in order to obtain non-universal classes of systems. Minimal parallelism fails, the sequential use of rules is similarly restrictive as the maximal parallelism, non-synchronization is too loose and weak in order to be used in "programming" the P systems. What else to consider is a topic worth investigating, and it might be useful to go back to biology and get inspiration from the living cell.

For further reading

[11]

Acknowledgements

This work has been partially supported by the research grants CEEX 47/2005 BioMAT 2-CEx06-11-97/19.09.06 (Romania), the National Natural Science Foundation of China (Grant Nos. 60373089 and 60674106), as well as by the Program for the New Century Talents in Universities, and the Ph.D. Program Foundation of the Ministry of Education of China. Useful suggestions by two anonymous referees are gratefully acknowledged.

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, Molecular Biology of the Cell, 4th ed., Garland Science, New York, 2002.
[2] A. Alhazov, P systems without multiplicities of symbol-objects, Information Processing Letters 100 (2006) 124–129.
[3] A. Alhazov, R. Freund, Gh. Păun, P systems with active membranes and two polarizations, in: Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, 2004, pp. 20–36.
[4] G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.), Applications of Membrane Computing, Springer-Verlag, Berlin, 2006.
[5] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, Grammar Systems. A Grammatical Approach to Distribution and Cooperation, Gordon and Breach, London, 1994.
[6] Z. Dang, O.H. Ibarra, On P systems operating in sequential mode, in: L. Ilie, D. Wotschke (Eds.) Pre-proc. of DCFS Workshop, London, Ontario, 2004, pp. 164–177.
[7] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer-Verlag, Berlin, 1989.
[8] R. Freund, Asynchronous P systems and P systems working in the sequential mode, in: G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, International Workshop, WMC5, Milano, Italy, in: LNCS, 3365, Springer-Verlag, Berlin, 2005, pp. 36–62. Selected and invited papers.
[9] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[10] M. Minsky, Computation. Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, NJ, 1967.
[11] A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, New Generation Computing 20 (2002) 295–306.
[12] Gh. Păun, Computing with membranes, Journal of Computer and System Sciences 61 (2000) 108–143.
[13] Gh. Păun, P systems with active membranes: Attacking NP-complete problems, Journal of Automata, Languages and Combinatorics 6 (2001) 75–90.
[14] Gh. Păun, Membrane Computing. An Introduction, Springer-Verlag, Berlin, 2002.
[15] M.J. Pérez-Jiménez, A. Romero–Jiménez, F. Sancho-Caparrini, Computationally hard problems addressed through P systems, in: [4], pp. 313–346.