Minimal Parallelism for Polarizationless P Systems

Tseren-Onolt Ishdorj

Research Group on Natural Computing
Department of Computer Science and AI, University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
tserren@yahoo.com
Computational Biomodelling Laboratory
Turku Centre for Computer Science
Åbo Akademi University, Turku 20520, Finland
tishdorj@abo.fi

Abstract. Minimal parallelism was recently introduced [3] as a way the rules of a P system are used: from each set of applicable rules associated to the same membrane, at least one must be applied. In this paper, we consider the minimal parallelism for P systems with active membranes *without polarizations*, using additional features, such as separation operations, changing membrane labels, catalytic or cooperative rules, etc. With several combinations of such features we obtain computational completeness. In cases where membrane division (of elementary or non-elementary membranes) is allowed, we show how SAT can be solved in polynomial time.

1 Introduction

Membrane systems (referred also as P systems) are a class of distributed parallel computing devices of a biochemical type, which can be seen as a general computing architecture where various types of objects can be processed in parallel by various operations. A key structural notion is that of a *membrane* by which a system is divided into compartments where chemical reactions can take place. These reactions transform multisets of objects present in the compartments into new objects, possibly transferring objects to neighboring compartments, including the environment.

For a comprehensive introduction to membrane computing we refer to [11].

As membrane systems are inspired by living cell behavior, a continuous research topic in the area was looking for as bio-realistic computing models as possible. In this framework, the application of developmental rules under different constraints is an interesting problem to study. An idea concerning the rule application that has recently attracted the attention is the *minimal parallelism*, introduced and investigated in [3]. Minimal parallelism relaxes the condition of using the rules in a maximally parallel way. More precisely, the rules are used in the *non-deterministic minimally parallel* manner: in each step, from each set of

rules R_i (associated with a membrane i of a P system) we use at least one rule (without specifying how many) provided that this is possible. The rules to be used, as well as the objects to which they are applied, are non-deterministically chosen.

In the original paper [3], certain open problems have been formulated, such as: removing the polarizations of membranes and/or decreasing the number of membranes in the universality proofs; whether or not membrane division for only elementary membranes suffices when solving computationally hard problems in polynomial time; looking for uniform constructions, or for deterministic systems.

We address here some of these problems, considering the minimal parallelism within the framework of P systems with active membranes, without using polarizations. In this framework, we have obtained the Turing completeness in accepting as well as in generative cases (Section 4) by simulating Minsky register machines, [6], and matrix grammars, [4]. Moreover, computational complexity issues (solving **NP**-complete problems) are also considered for polarizationless P systems in Section 5, where both uniform and semi-uniform solutions to `SAT` are provided.

2 Preliminaries

We assume the reader to be familiar with the basic elements of formal languages, Turing computability [5], computational complexity, [9], and membrane computing, [11]. We introduce here only some of the necessary notions and notation.

An *alphabet* is a finite set of symbols (letters), and a word (string) over an alphabet Σ is a finite sequence of letters from Σ. We denote the empty word by λ. A *multiset* over an alphabet Σ is a mapping from Σ to $\mathbf{N}$, the set of natural numbers; we represent a multiset by a string from Σ^*, where the number of occurrences of a symbol $a \in \Sigma$ in a string w represents the multiplicity of a in the multiset represented by 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"; RE denotes the family of recursively enumerable languages).

For the Turing computability proofs in the next sections, we use the characterization of NRE by means of *matrix grammars* (in a precise normal form) and *register machines*.

A matrix grammar in the *binary normal form* is a construct $G = (N, T, S, M, F)$, where $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$;
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$;
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$;
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1 (that is why one uses to write it in the form $(S \to X_{init}A_{init})$, in order to fix the symbols X, A present in it),

and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix in $m \in M$ such that starting from w and applying once each rule of m in the order specified by m, one can obtain z; a rule can be skipped if it is in F and it is not applicable.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. It is known that $MAT_{ac} = RE$.

In turn, an *n-register machine* is a construct $M = (n, B, l_0, l_h, I)$, where n is the number of registers, B is a set of labels, and I is a set of labeled instructions of the form $l_i : (op(r), l_j, l_k)$, where $op(r)$ is an operation on register r of M, and l_i, l_j, l_k are labels from the set B; l_0 is the label of the initial instruction, and l_h is the label of the halting instruction. The machine is capable of the following instructions:

1. $l_i : (\texttt{ADD}(r), l_j, l_k)$: Add one to the content of register r and proceed, in a non-deterministic way, to instruction with label l_j or to instruction with label l_k; in the deterministic variant we demand $l_j = l_k$ and then the instruction is written in the form $l_i : (\texttt{ADD}(r), l_j)$.
2. $l_i : (\texttt{SUB}(r), l_j, l_k)$: If register r is not empty, then subtract one from its contents and go to instruction with label l_j, otherwise proceed to instruction with label l_k.
3. $l_h : halt$: This instruction stops the machine and can only be assigned to the final label l_h.

A deterministic n-register machine can analyze an input $m \in \mathbf{N}$, introduced in register 1, which is accepted if and only if the machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, then the analysis was not successful. We denote by $N(M)$ the set of numbers accepted by the register machine M. If Q is a Turing computable set, then there exists a deterministic register machine M with at most three registers, such that $N(M) = Q$, [6].

It is not enough that a problem can be solved algorithmically, it is necessary that the solution comes in a reasonable time and using reasonable computing resources. Defining what "reasonable time and resources" means and classifying problems from these points of view are the main tasks of computational complexity. For details we refer to [9]. In Section 5 we show that the **NP**-complete problem SAT can be solved in polynomial time by P systems from various classes, that is why we introduce here this problem.

In SAT we are given Boolean variables $x_1, x_2, \ldots, x_n$ and a Boolean formula β involving such variables; the formula is given in a particular format called *conjunctive normal form*, that we will explain in a moment. The question is whether there is a way to assign Boolean $(true/false)$ values to the variables so that the formula is satisfied.

Boolean formulas are constructed starting from variables and applying the operators $\vee$ (that stands for OR), $\wedge$ (that stands for AND), and $\neg$ (that stands for NOT).

A *clause* is formed by taking one or more variables and connecting them with OR; for example, $(x_2 \vee \neg x_4 \vee x_5)$ is a clause. A *formula in conjunctive normal form* is the AND of a given set of clauses. For example, $(x_3 \vee \neg x_4) \wedge (x_1) \wedge (\neg x_3 \vee x_2)$ is a formula in conjunctive normal form. Note that the above formula is satisfiable, and, for example, it is satisfied by setting all the variables to *true* (there are also other possible assignments of values to the variables that would satisfy the formula). On the other hand, the formula $(x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2)$ is not satisfiable, as it can easily be observed.

3 P Systems with Active Membranes

We briefly recall now the notion of P systems with active membranes.

Informally speaking, in P systems with active membranes, the following types of rules are used: (a) multiset rewriting rules (including non-cooperative and cooperative evolution rules), (b) rules for introducing objects into membranes, (c) rules for sending objects out of membranes, (d) rules for dissolving membranes, (e) rules for dividing elementary membranes, and (f) rules for dividing non-elementary membranes. In these rules, a single object is involved. Occasionally, also (g) membrane merging rules, (h) membrane separation rules, (i) membrane release rules, (k) replicative-distribution rules for sibling membranes, and (l) replicative-distribution rules for nested membranes were used, e.g., in [1, 7, 8]. Their common feature is that they involve multisets of objects.

Formally, a *polarizationless P system with active membranes* is a construct

$$\Pi = (O, C, H, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, h_0),$$

where: $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects*; $C \subseteq O$ is the set of *catalysts* (when $C = \emptyset$, we omit writing it); H is a finite set of *labels* for membranes; μ is a *membrane structure*, consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H; $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ; h_0 is the output membrane of Π; $R_i, 1 \leq i \leq m$, are finite sets of *developmental rules* of the forms described below.

In these rules, objects a, b, d are from the alphabet O, c is a catalyst from C, h is a label from H, and u, v are multisets of objects over the alphabet O. In membrane separation operation (*sep*), $Q \subseteq O$ and $K \subset Q$; we denote the subtraction operation $Q - K$ by $\neg K$.

It is important to note that we do not use polarizations for membranes. (As considered in [10, 12], the membranes can have one of the negative, positive, neutral, "electrical charges", represented by $-$, $+$, and 0, respectively, but we do not use this feature here.) In the literature, one uses to add the subscript 0 to indicate that the rules do not use polarizations, writing, for instance, sep_0, mer_0, etc., but here we omit this subscript.

Action & Identification	Type of Rule
separate (*sep*)	$[\ Q]_h \rightarrow [\ K]_h [\ \neg K]_h$
merge (*mer*)	$[\ ]_h [\ ]_h \rightarrow [\ ]_h$
move in (*in*)	$a[\ ]_h \rightarrow [\ b]_h$
move out (*out*)	$[\ b]_h \rightarrow [\ ]_h a$
promoter (*pro*)	$[\ a \rightarrow v\vert_b]_h$
catalytic (*cat*)	$[\ ca \rightarrow cv]_h$
cooperative (*coo*)	$[\ v \rightarrow u]_h$
non-cooperative (*ncoo*)	$[\ a \rightarrow v]_h$
divide for elementary (*ediv*)	$[\ a]_h \rightarrow [\ b]_h [\ d]_h$
divide for non-elementary (*ndiv*)	$[\ a]_h \rightarrow [\ b]_h [\ d]_h$

The actions and the notations of rules are as follows: *sep* – separation rules for elementary membranes; the membrane h, containing objects from Q, is separated into two membranes with the same labels; the objects from K are placed in the first membrane, those from $Q-K$ are placed in the other membrane; we request that both K and $Q-K=\neg K$ are not empty and also that both membranes $[\ K]_h$, $[\ \neg K]_h$ are non-empty (the rule is not applied otherwise). *mer* – merging rules for elementary membranes; the two membranes are merged into a single membrane; the objects of the former membranes are put together in the new membrane. *in* – communication rules; an object is introduced in the membrane during this process. *out* – communication rules; an object is sent out of the membrane during this process. *pro* – evolution rule with promoter; the rule is applied only if the promoter object b is present in the region. *cat* – catalytic rules; $ca \rightarrow cv$, where $c \in C, a \in O-C, v \in (O-C)^*$; the catalyst c is never modified, it only assists the evolution of other objects. *coo* – evolution rules of radius greater than one; a particular case is that of catalytic rules. *ncoo* – object evolution rules, associated with membranes and depending on the label, 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. *ediv* – 2-division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label; the object specified in the rule is replaced in the two new membranes by possibly new objects and the remaining objects are duplicated. *ndiv* – 2-division rules for non-elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label; the object specified in the rule is replaced in the two new membranes by possibly new objects; the remaining objects *and membranes* contained in this membrane are duplicated, and then are part of the contents of both new copies of the membrane [2].

Each copy of an object and each copy of a membrane can be used by only one rule (a promoter can promote several rules at the same time, and a membrane can appear in several evolution rules at the same time).

In each step, the use of rules is done in the bottom-up manner (first the inner objects and inner membranes evolve, and the result is duplicated if any surrounding membrane is divided or separated).

When the rules of a given type (α) are able to change the label(s) of the involved membranes, we denote that type of rules by (α'). For example, the primed versions of the merging and the separation rules are of the following forms:

(mer'): $[\ ]_{h_1}[\ ]_{h_2} \to [\ ]_{h_3}$, for $h_1, h_2, h_3 \in H$.
(sep'): $[\ Q]_{h_1} \to [\ K]_{h_2}[\ \neg K]_{h_3}$, for $h_1, h_2, h_3 \in H, K \subset Q$.

Here we use the rules in the *minimally parallel* manner [3]. All rules of any type involving a membrane h form a set R_h. Moreover, if a membrane h appears several times in a given configuration of the system, then for each occurrence i of the membrane we consider a different set R_{h_i}. Then, in each step, from each set $R_{h_i}, h \in H$, from which at least a rule *can* be used, at least one rule *must* be used.

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.

A halting computation gives a result, in the form of the number of objects present in membrane h_0 in the end of the computation; for a computation to be successful, exactly one membrane with label h_0 should be present in the halting configuration. The set of numbers generated in this way by a system Π is denoted by $N_{gen}(\Pi)$ and the family of such sets, generated by systems having initially at most n_1 membranes and using during the computation configurations with at most n_2 membranes is denoted by $N^{gen}_{mp}OP_{n_1,n_2}$(*types-of-rules*), with the subscript *"mp"* indicating the "minimal parallelism" used in computations, and *types-of-rules* indicating the allowed types of rules. A P system can be also used in the accepting mode: we introduce a number in the system in the form of a multiset a^n, for some $a \in O$, in region h_0 and start computing; if the system halts, then the number n is accepted. The set of all numbers accepted in this way by Π is denoted by $N_{acc}(\Pi)$, and the family of such sets is denoted by $N^{acc}_{mp}OP_{n_1,n_2}$(*types-of-rules*), with the obvious meaning of the used parameters. When the number of membranes does not increase during the computation we use only the subscript n_1, denoting the maximal number of membranes initially present in the used systems. When using systems with at most r catalysts, we write cat_r for the respective type of rules.

In what follows we give several accepting and generative universality results, as well as efficiency results for polarizationless P systems working in the *minimally parallel mode*.

4 Computational Completeness Results

The Accepting Case When we remove polarizations from active membranes, additional features are in general necessary in order to reach the universality. There are several such features, for instance, cooperative rewriting rules, changing membrane labels, using promoter/inhibitor objects, priorities among rules, etc. Some of these tools will be also used in what follows.

In the proofs we will simulate register machines. In all constructions, with each register r of a register machine we associate a membrane with label r, and the number stored in register r is represented by the number of copies of object a present in membrane r.

Theorem 1. $N^{acc}_{mp}OP_n(cat_1, pro, in, out) = NRE, n \geq 7$.

Proof. Let us consider a deterministic register machine $M = (3, B, l_0, l_h, I)$ accepting an arbitrary set $N(M) \in NRE$. We construct the P system

$$\begin{aligned}
&\Pi = (O, C, H, \mu, (w_h)_{h\in H}, (R_h)_{h\in H}, 1) \text{ with}\\
&O = \{a, c\} \cup \{l, \bar{l}, l', l'', l''', l^{iv}, l^{v}, l^{vi}, l^{vii} \mid l \in B\},\\
&C = \{c\},\\
&H = \{0, 1, 1', 2, 2', 3, 3'\},\\
&\mu = [\,[\,[\,\,]_{1'}]_1[\,[\,\,]_{2'}]_2[\,[\,\,]_{3'}]_3]_0,\\
&w_0 = l_0, w_1 = w_2 = w_3 = c, w'_1 = w'_2 = w'_3 = \lambda,
\end{aligned}$$

and with the following rules in $R_h, h \in H$. (Remember that the number to be analyzed is introduced in region 1 in the form a^n.)

An instruction $l_1 : (\texttt{ADD}(r), l_2)$ is simulated by means of the following rules:

step	R_0	R_r
1.	—	$l_1[\,\,]_r \to [\,l_1]_r$
2.	—	$[\,l_1 \to l'_2 a]_r$
3.	—	$[\,l'_2]_r \to [\,\,]_r l'_2$
4.	$[\,l'_2 \to l_2]_0$	—

The label-object l_1 enters the correct membrane r, produces one further copy of a and the label l_2, primed, inside membrane r, then the label l'_2 exits to the skin region, loses its prime, and the process can be iterated.

For the simulation of a SUB instruction $l_1 : (\texttt{SUB}(r), l_2, l_3)$ we use the next rules:

step	R_0	R_r	$R_{r'}$
1.	$[l_1 \to l'_1 l''_1]_0$	—	—
2.	$[l'_1 \to l'''_1]_0$	$l''_1[\,\,]_r \to [l''_1]_r$	—
3.	$[l'''_1 \to l^{iv}_1]_0$	$[ca \to ca'|_{l''_1}]_r$	$l''_1[\,\,]_{r'} \to [l''_1]_{r'}$
4.	—	$l^{iv}_1[\,\,]_r \to [l^{iv}_1]_r$	$[l''_1 \to l^{v}_1]_{r'}$
5.	—	$[l^{iv}_1 \to \bar{l}_2|_{a'}]_r$	$[l^{v}_1 \to l^{vi}_1]_{r'}$
6.	—	$[a' \to \lambda|_{\bar{l}_2}]_r$	$[l^{vi}_1]_{r'} \to l^{vi}_1[\,\,]_{r'}$
7.	—	$[\bar{l}_2 \to l_2|_{l^{vi}_1}]_r$ or $[l^{iv}_1 \to l_3|_{l^{vi}_1}]_r$	$l^{vi}_1[\,\,]_{r'} \to [l^{vii}_1]_{r'}$
8.	—	$[l_2]_r \to l_2[\,\,]_r$ or $[l_3]_r \to l_3[\,\,]_r$	$[l^{vii}_1 \to \lambda]_{r'}$

We start the computation by producing a couple of objects l'_1 and l''_1 in the skin region by rule 1. Then l'_1 evolves to l'''_1 in the skin region, while object l''_1

enters into membrane r. In the third step, while l_1''' changes to the next primed version l_1^{iv} in the skin region, object l_1'' enters into the inner membrane r' – this happens in both cases irrespective whether object a exists or not in membrane r. If an object a was present, it evolves to a' in the presence of promoter object l_1'' by means of the catalytic evolution rule $[\ ca \to ca'|_{l_1''}]_r$ and the promoter leaves the membrane r. The catalyst c is used to prevent more objects a to evolve in the same step. At the fourth step, object l_1'' evolves to l_1^v in membrane r' and object l_1^{iv} enters into membrane r, respectively. Since no object remains in the skin region, this region will stay idle until the end of the computation. In the step five, object l_1^v evolves to l_1^{vi}, which will be sent out of the membrane r' in the following step; in membrane r, if a promoter object a' is present, then object l_1^{iv} produces the object $\bar{l}_2$. Otherwise, there is no rule to be applied here in this or the next step. In step 6, object l_1^{vi} arrives into membrane r, where the object $\bar{l}_2$ promotes the deletion rule $[a' \to \lambda|_{\bar{l}_2}]_r$ and removes the object a' previously produced. In membrane r, object l_1^{vi} promotes either object $\bar{l}_2$ or l_1^{iv}, to introduce the corresponding label-object l_2 or l_3, respectively. The correct label-object l_2 or l_3 is moved to skin region at the 8th step of the computation. The object l_1^{vi} enters membrane r' and is removed.

Note that in the simulation of instructions `ADD` and `SUB` of M in each computation step at most one rule from each set R_h has been used, hence the system works both in the minimally and in the maximally parallel mode. The starting configuration of the system is restored after each simulation, hence another instruction can be simulated. If the computation in M halts, hence l_h is reached, this means that the halting label-object l_h is introduced in the skin region, and also the computation in Π halts. Consequently, $N(M) = N_{acc}(\Pi)$, and this concludes the proof. □

Because of space restriction, the proofs of the next two results are omitted.

Theorem 2. $N_{mp}^{acc}OP_n(ncoo, in', out') = NRE, n \geq 4$.

In the next theorem we use the rather strong tool of cooperative evolution rules (with radius at most 2), but the membrane labels will not be changed during the computation.

Theorem 3. $N_{mp}^{acc}OP_n(coo, in, out) = NRE, n \geq 4$.

The Generative Case We consider now P systems working in the generative mode.

The next universality result is based on the simulation of a matrix grammar. Catalytic evolution rules, membrane merging and separation, also changing membrane labels, are used in the proof.

Theorem 4. $N_{mp}^{gen}OP_{2,5}(cat_1, sep', mer') = NRE$.

Proof. Let us consider a matrix grammar $G = (N, T, S, M, F)$ with appearance checking, in the *binary normal form*, hence with $N = N_1 \cup N_2 \cup \{S, \#\}$, $T = \{a\}$, and with the matrices of the forms as mentioned in Section 2.

Assume that all matrices are injectively labeled with elements of a set B. We construct the P system of degree 2

$$\begin{aligned}
&\Pi = (O, C, H, \mu, w_0, w_s, (R_h)_{h\in H}, 0) \text{ with}\\
&O = N_1 \cup N_2 \cup \{c, c_1, c_2, a, f, \#\},\\
&C = \{c\},\\
&H = \{m, m_1, m_2, m_1', m_1'', m_2', m_2'', m_2''' \mid m \in B\} \cup \{0, 0', 0'', s\},\\
&\mu = [\,[\,\,]_0]_s,\\
&w_0 = XAcc_1c_2, w_s = \lambda,
\end{aligned}$$

and the sets R_h containing the rules below.

The simulation of a matrix $m : (X \to Y, A \to x)$, with $X \in N_1$, $Y \in N_1$, and $A \in N_2, x \in (N_2 \cup T)^*$, $|x| \leq 2$, is done in four steps, using the following rules (the matrix with label m is encoded in the labels of the membranes created by the separation operation; we start from a configuration of the form $[\,[\,Xwcc_1c_2]_0]_s$, for some $w \in (N_2 \cup T)^*$):

step	$(X \to Y,$	$A \to x)$
1.	$[\,O]_0 \to [\,\{X\}]_{m_1}[\,\neg\{X\}]_{m_2}$	—
2.	$[\,X \to Y]_{m_1}$	$[\,O]_{m_2} \to [\,\{c, A\}]_{m_2'}[\,\neg\{c, A\}]_{m_2''}$, or
		$[\,c_1 \to \#]_{m_2}$, $[\,\# \to \#]_{m_2}$
3.	$[\,\,]_{m_1}[\,\,]_{m_2''} \to [\,\,]_{m_1'}$	$[\,cA \to cx]_{m_2'}$
4.	$[\,\,]_{m_1'}[\,\,]_{m_2'} \to [\,\,]_0$	—
—	$[\,\alpha \to \#]_{m_1'}, \alpha \in O$	$[\,\# \to \#]_{m_1'}$

We start the simulation of matrix m by separating membrane 0 under the control of object X. After separation, object X takes place in a new membrane m_1, and other objects (including c, c_1, c_2) are placed in a membrane m_2. In the second step, object X evolves to Y while objects A, c makes membrane m_2 separate. At the third step, one copy of object A evolves to x in the new membrane m_2'. Thus, the rules of matrix m have been simulated. In the meantime, membranes with labels m_1 and m_2'' are merged into a new membrane labeled m_1'. At the fourth step, objects Y, x, c, c_1, c_2 returns to membrane 0, obtained by merging membranes m_1' and m_2'. If this does not happen, i.e., the rule $[\,cA \to cx]_{m_2'}$ is used again in step 4, then the trap object $\#$ is introduced (there is at least the object Y in membrane m_1'). If the object X is present, hence membrane 0 is separated, but object A not, then in step 2 one introduces the trap object $\#$ by the rule $[\,c_1 \to \#]_{m_2}$, which must be used, because $[\,O]_{m_2} \to [\,\{c, A\}]_{m_2'}[\,\neg\{c, A\}]_{m_2''}$ cannot be used.

Thus, the matrix m is correctly simulated, and the system can pass to the simulation of another matrix.

The simulation of a matrix with appearance checking $m : (X \to Y, A \to \#)$, with $X, Y \in N_1$, and $A \in N_2$, is done in four steps using the following rules:

step	$(X \to Y,$	$A \to \#)$
1.	$[\ O]_0 \to [\ \{X, c_1\}]_{m_1}[\ \neg\{X, c_1\}]_{m_2}$	
2.	$[\ O]_{m_1} \to [\ \{X\}]_{m_1'}[\ \neg\{X\}]_{m_1''}$	$[\ O]_{m_2} \to [\ \{A\}]_{m_2'}[\ \neg\{A\}]_{m_2''}$
3.	$[\ X \to Y]_{m_1'}$	$[\ A \to \#]_{m_2'}$
	$[\ ]_{m_1''}[\ ]_{m_2} \to [\ ]_{m_2'''}$	$[\ \# \to \#]_{m_2'}$
4.	$[\ ]_{m_1'}[\ ]_{m_2'''} \to [\ ]_0$	

The computation starts with objects X, c_1 making membrane 0 to separate. At the second step, if membrane m_2 includes an object A, then it will separate into membranes m_2' including A and m_2'' including the auxiliary object c_2. If A existed, then the computation never stop. At the same time, objects X and c_1 take place in the membranes m_1' and m_1'', respectively. In the third step, X evolves to Y, and if membrane m_2 is still present, then membrane m_1'' merges with it, creating a new membrane m_2'''. Finally, membranes m_1' and m_2''' are merged into membrane 0 including all correct objects and the system returns to a configuration as the starting one.

The simulation of a matrix $m : (X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$, is done in six steps, using the following rules:

step	$(X \to f,$	$A \to x)$
1.	$[\ O]_0 \to [\ \{X\}]_{m_1}[\ \neg\{X\}]_{m_2}$	—
2.	$[\ X \to f]_{m_1}$	$[\ O]_{m_2} \to [\ \{c, A\}]_{m_2'}[\ \neg\{A\}]_{m_2''}$, or
		$[\ c_1 \to \#]_{m_2}$, $[\ \# \to \#]_{m_2}$
3.	$[\ ]_{m_1}[\ ]_{m_2''} \to [\ ]_{m_1'}$	$[\ cA \to cx]_{m_2'}$
4.	$[\ ]_{m_1'}[\ ]_{m_2'} \to [\ ]_{0'}$	—
—	$[\ \alpha \to \#]_{m_1'}, \alpha \in O$	$[\ \# \to \#]_{m_1'}$
5.	$[\ O]_{0'} \to [\ \{f\} \cup T]_0[\ \neg(\{f\} \cup T)]_{0''}$	—
6.	$[\ f \to \lambda]_0$	$[\ Z \to \#]_{0''}, Z \in N_2$
		$[\ \# \to \#]_{0''}$

We omit here the detailed explanation for the first 4 steps, because they are the same as in the simulation of matrix $m : (X \to Y, A \to x)$. At step 5, membrane $0'$ is separated into membranes with labels 0 and $0''$. The former one includes the terminal objects and the special object f. Object f evolves to λ at step 6. If there are objects Z from N_2 in membrane $0''$, then the rule $[\ Z \to \#]_{m_1'}$ is applied and the computation will never halt. Thus, the simulation is correctly completed. □

5 Computational Complexity Results

We present here both uniform and semi-uniform linear time solutions of `SAT` based on polarizationless P systems working in the minimally parallel mode. Three results are given, with the following features (types of rules, label changing, type of construction):

no.	evolution	division	communication	label changing	construction
1.	cooper.	elementary	move out	no	semi-uniform
2.	non-coop.	non-element.	move in, out	yes	uniform
3.	cooper.	non-element.	move in, out	no	uniform

A rigorous framework for dealing with complexity matters in our area is that of *recognizing P systems*, which we introduce here following [12]. First, let us consider P systems *with input*, which will allow to input a multiset encoding a decision problem, in a special membrane. Such a device is a tuple (Π, V, i_0), where:

- Π is a usual P system, with the alphabet of objects O and initial multisets $w_1, \ldots, w_m$ (associated with membranes labeled by $1, \ldots, m$, respectively).
- V is an (input) alphabet strictly contained in O and such that $w_1, \ldots, w_m$ are multisets over $O - V$.
- $i_0 \in \{1, 2, \ldots, m\}$ is the label of a distinguished membrane (of input).

If w is a multiset over V, then the initial configuration of (Π, V, i_0) with input w is $(\mu, w'_1, \ldots, w'_m)$, where $w'_i = w_i$ for $i \neq i_0$, and $w'_{i_0} = w_{i_0} \cup w$.

The computations of a P system with input are defined in a natural way, the only change is that the initial configuration is obtained by adding the input multiset w over V to the initial configuration of the system Π.

Then, a *recognizing* P system is a P system with input, (Π, V, i_0), such that:

1. The alphabet O of Π contains two distinguished elements, `yes`, `no`.
2. All computations of the system halt.
3. If $\mathcal{C}$ is a computation of Π, then either the object `yes` or the object `no` (but not both) is sent out to the environment, and only in the last step of the computation.

We say that $\mathcal{C}$ is an accepting (respectively, rejecting) computation if the object `yes` (respectively, `no`) appears in the environment in the halting configuration of $\mathcal{C}$.

To understand what solving a problem in a semi-uniform/uniform way means, we consider a decision problem X. A family $\Pi_X = (\Pi_X(1), \Pi_X(2), \ldots)$ of P systems (with active membranes in our case) is called *semi-uniform* (*uniform*) if its elements are constructible in polynomial time starting from $X(n)$ (from n, respectively), where $X(n)$ denotes the instance of size n of X. We say that X can be solved in polynomial (linear) time by the family Π_X if the system $\Pi_X(n)$ will always stop in a polynomial (linear, respectively) number of steps, sending out the object `yes` if and only if the instance $X(n)$ has a positive answer. For more details about complexity classes for P systems see [11, 12].

Note that always we have an answer, one of `yes` and `no`, but we have said nothing about the way the computations evolve, the only restriction we impose is that all of them halt (in a number of steps bounded by a known function). That is why we say that such systems are *confluent* (they may be non-deterministic, but the answer is obtained in a finite time irrespective of the possible branchings of the computations). The *deterministic* systems, where no branching is possible, are a particular case of confluent systems.

Theorem 5. *P systems constructed in a uniform manner and working in the minimally parallel mode using rules of types* $(ncoo, ndiv, in, out')$ *can solve* `SAT` *in linear time.*

Proof. Let us consider a propositional formula in the conjunctive normal form, $\alpha = C_1 \wedge \cdots \wedge C_m$, such that each clause $C_i, 1 \leq i \leq m$, is of the form $C_i = y_{i,1} \vee \cdots \vee y_{i,k_i}, k_i \geq 1,$ where $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$.

The instance α is encoded as a set over

$$\Sigma(\langle n, m\rangle) = \{x_{i,j}, x'_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}.$$

The object $x_{i,j}$ represents the variable x_j appearing in the clause C_i without negation, and object $x'_{i,j}$ represent the variable x_j appearing in the clause C_i with negation. Thus, the input multiset is

$$\begin{aligned} w = &\{x_{i,j} \mid x_j \in \{y_{i,j} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq m, 1 \leq j \leq n\} \\ &\cup \{x'_{i,j} \mid \neg x_j \in \{y_{i,j} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq m, 1 \leq j \leq n\}. \end{aligned}$$

For given $(n, m) \in \mathbf{N}^2$, we construct a recognizing P system $(\Pi(\langle n, m\rangle), \Sigma(\langle n, m\rangle), 0)$ with:

$$\begin{aligned} \Pi(\langle n, m\rangle) &= (O(\langle n, m\rangle), H, \mu, (w_h)_{h\in H}, (R_h)_{h\in H}), \text{where} \\ O(\langle n, m\rangle) &= \{x_{i,j}, x'_{i,j}, t_{j,i}, f_{j,i} \mid 1 \leq i \leq m, 1 \leq j \leq n\} \\ &\quad \cup \{b_i, c_i, a'_i, a''_i, a'''_i \mid 1 \leq i \leq n\} \cup \{\mathtt{yes}, \mathtt{no}, d\} \\ &\quad \cup \{l'_i \mid 0 \leq i \leq n\} \cup \{l_i, e_i, \mid 1 \leq i \leq m\} \\ &\quad \cup \{a_i \mid 0 \leq i \leq n+2\} \cup \{d_i \mid 0 \leq i \leq 6n+m+7\}, \end{aligned}$$

$$\begin{aligned} \mu &= [\,[\,[\,[\,]_m \cdots [\,]_2 [\,]_1]_0]_a [\,]_d]_s, \\ w_a &= a_0, w_d = d_0, w_0 = l'_0, w_s = w_i = \lambda, 1 \leq i \leq m, \\ H &= \{s, a, b, d, y, n\} \cup \{i \mid 0 \leq i \leq m\} \cup \{i', i'' \mid 0 \leq i', i'' \leq n\}, \end{aligned}$$

and the following rules (we also give explanations about their use):

1. $[\, d_i \to d_{i+1}]_d, 0 \leq i \leq 6n+m+5.$

Object d_i counts the computation steps in membrane d.

Initialization phase:

2. $x_{i,j}[\,]_i \to [\, x_{i,j}]_i,$
   $x'_{i,j}[\,]_i \to [\, x'_{i,j}]_i, 1 \leq i \leq m, 1 \leq j \leq n.$

We re-encode the problem α into m membranes in n steps.

3. $[\, a_i \to a_{i+1}]_a, 0 \leq i \leq n+2.$
4. $[\, l'_i \to l'_{i+1}]_0, 0 \leq i \leq n-1.$

Simultaneously with the use of rule 2, the evolution rules 3 and 4 are applied in membranes a and 0, respectively.

5. $[\ l'_n \to l_1 l_2 \dots l_m]_0$.
6. $l_i[\ ]_j \to [\ l_i]_j$, $1 \le i, j \le m$.
7. $[\ l_i]_j \to [\ ]_{0'} d$, $1 \le i, j \le m$.
8. $a_{n+2}[\ ]_0 \to [\ a_1]_0$.

At step $n+1$, object l'_n evolves to $l_1, l_2, \dots, l_m$. Consequently, in order to change the membrane labels $j, 1 \le j \le m$, by $0'$, those objects enter into and are sent out of membranes in two steps, by means rules 6 and 7. Meanwhile, object a_{n+2} evolves to a_1 and enters into membrane 0, by rule 8. Thus, the initialization phase has been completed in $n+3$ steps.

Checking phase:

9. $[\ a_i]_0 \to [\ b_i]_0 [\ c_i]_0, 1 \le i \le n$.
10. $[\ b_i \to t_{1,i} t_{2,i} \dots t_{m,i} a'_i]_0$,
    $[\ c_i \to f_{1,i} f_{2,i} \dots f_{m,i} a'_i]_0, 1 \le i \le n$.

We generate 2^n membranes with label 0 by using non-elementary membrane division rule 9. In each step, b_i and c_i correspond to the truth values *true* and *false*, respectively, for variable x_i. By rule 10, objects b_i and c_i evolve. Rules 9 and 10 are performed in 2 steps.

11. $t_{i,j}[\ ]_{(j-1)'} \to [\ t_{i,j}]_{(j-1)'}$, or
    $t_{i,j}[\ ]_{(j-1)''} \to [\ t_{i,j}]_{(j-1)''}$,
    $f_{i,j}[\ ]_{(j-1)'} \to [\ f_{i,j}]_{(j-1)'}$, or
    $f_{i,j}[\ ]_{(j-1)''} \to [\ f_{i,j}]_{(j-1)''}$, $1 \le i \le m, 1 \le j \le n$.
12. $[\ a'_j \to a''_j]_0$.

The groups of objects $t_{1,j}, t_{2,j}, \dots, t_{m,j}$ and $f_{1,j}, f_{2,j}, \dots, f_{m,j}$, corresponding to variable x_j, are introduced into inner membranes with labels $(j-1)'$ or $(j-1)''$ by rule 11, and object a'_j is evolved to a''_j by rule 12. These two rules are applied simultaneously.

13. $[\ t_{i,j}]_{(j-1)'} \to [\ ]_{j'} d$, $[\ t_{i,j}]_{(j-1)''} \to [\ ]_{j'} d$,
    $[\ f_{i,j}]_{(j-1)'} \to [\ ]_{j''} d$, $[\ f_{i,j}]_{(j-1)''} \to [\ ]_{j''} d$, $1 \le i \le m, 1 \le j \le n$.
14. $[\ a''_j \to a'''_j]_0$.

By using rule 13, objects $t_{i,j}$ and $f_{i,j}$ (corresponding to the truth value *true* and *false* for variable x_j of a clause C_i) leave membranes with labels $(j-1)'$ or $(j-1)''$ and change the labels to j' and j'', respectively. Object a'''_j is produced in membrane 0 by rule 14. Now (using rules 15) we will check which clauses are satisfied by truth values.

15. $[\ x_{i,j} \to e_i]_{j'}$,

$[\ x'_{i,j} \rightarrow e_i]_{j''}, 1 \leq i \leq m, 1 \leq j \leq n.$

16. $[\ a'''_j \rightarrow a_{j+1}]_0$.

We remind that single and double primes of the labels j' and j'' indicate *true* and $false$ values, respectively. By rule 15, object $x_{i,j}$ in membrane j' and object $x'_{i,j}$ in membrane j'' evolve to e_i. More precisely, the object without negation has to evolve to object e_l, because the membrane label k' has a single prime and this indicates the *true* value, and the subscript of e_l must be the same with the first subscript of the object $x_{l,k}$. Similarly, the object with negation has to evolve to e_l because of k'' which is double-primed and indicates the value *false*.

In the membranes 0, the objects a'''_i lose their primes and increase the subscript a_{i+1}. We continue in this way the operations of membrane division and construction of truth assignments corresponding to the next variable x_{i+1} (rules 9-16). Between two divisions 5 steps are performed. For having all possible 2^n truth assignments, we need $5n$ division processes. In this way, we check the satisfiability of all clauses and this process ends at the $6n+3$th step of the computation. At that step, objects a_{n+1} are removed from membrane 0 changing its label to 1, by means of rule 17.

17. $[\ a_{n+1}]_0 \rightarrow [\ \ ]_1 d$.

Recognizing phase:

After the $6n+3$ steps, there are 2^n membranes with label 1 and each of them contains m membranes labeled by n' or n''.

18. $[\ e_i]_{n'} \rightarrow [\ \ ]_n e_i$,
$[\ e_i]_{n''} \rightarrow [\ \ ]_n e_i, 1 \leq i \leq m.$

By rule 18, objects $e_i, 1 \leq i \leq m$, are introduced into membrane 1 changing the former membrane labels from n' and n'' to n. If all objects $e_1, e_2, \ldots, e_m$ are present in a membrane with label 1, that means that all clauses $C_1, C_2, \ldots, C_m$ are satisfied.

19. $[\ e_i]_i \rightarrow [\ \ ]_{i+1} e_i, 1 \leq i \leq m.$

Objects $e_i, 1 \leq i \leq n$, leave membrane i one by one increasing the label. In m steps, we can get objects e_m to appear in membrane a. One of them is non-deterministically chosen and expelled out of membrane a, which changes the label to b, by means of rule 20.

20. $[\ e_m]_a \rightarrow [\ \ ]_b e_m$.
21. $[\ e_m \rightarrow \texttt{yes}]_s$.
22. $[\ \texttt{yes}]_s \rightarrow [\ \ ]_y \texttt{yes}$.

If an object e_m had appeared in the skin membrane, then it will evolve to `yes`; this object will be sent to the environment, changing the skin membrane label to y by using rules 21, 22, in the $6n + m + 7$th step of the computation.

23. $[\ d_{6n+m+6}]_d \rightarrow [\ \ ]_d d_{6n+m+7}$.
24. $[\ d_{6n+m+7}]_s \rightarrow [\ \ ]_n \texttt{no}$.

If the formula is not satisfiable, the object `no` is sent to environment in the step $6n + m + 8$ by rule 24.

Thus, the problem is solved in a uniform manner, in a linear time, with the system working in both the minimally parallel and the maximally parallel modes. □

Due to space restriction, we again give next theorems without the proofs. The proof ideas are the same as for Theorem 5, that is, we re-encode the input multiset in m membranes, and in this way we avoid the maximal parallel applications of rules in the same membrane.

Theorem 6. *P systems constructed in a uniform manner, working in the minimally parallel mode using rules of types* $(coo, ndiv, in, out)$ *can solve* `SAT` *in linear time.*

Theorem 7. *P systems working in the minimally parallel mode with rules of types* $(coo, ediv, out)$ *and constructed in a semi-uniform manner can solve* `SAT` *in linear time with respect to the number of the variables and the number of clauses.*

6 Final Remarks

The main contribution of this paper is the use of the minimal parallelism in the framework of P systems with active membranes, without using membrane polarizations. Both universality and efficiency results were proved in this framework, for various combinations of types of rules.

Besides possible improvements of the previous theorems, it also remains to investigate the possibility of obtaining similar universality and efficiency results for other classes of P systems working in the minimally parallel mode, in particular, for the case of using rules of the forms $ncoo, out, sep$.

Acknowledgment. The work of the author is supported by the Center for International Mobility (CIMO) Finland, grant TM-06-4036.

References

1. A. Alhazov, T.-O. Ishdorj, Membrane Operations in P Systems with Active Membranes. In: Gh. Păun, et al. (eds.) *Second Brainstorming Week on Membrane Computing*, Sevilla, 2-7 February, 2004, **TR** 01/2004, University of Sevilla, 37–44.
2. A. Alhazov, L. Pan, Gh. Păun, Trading Polarizations for Labels in P Systems with Active Membranes, *Acta Informaticae*, 41, 2-3 (2004), 111–144.
3. G. Ciobanu, L. Pan, Gh. Păun, M. J. Pérez-Jiménez, P Systems with Minimal Parallelism. Submitted 2005.

4. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
5. J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.
6. M. L. Minsky, *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, 1967.
7. L. Pan, A. Alhazov, T.-O. Ishdorj, Further Remarks on P Systems with Active Membranes, Separation, Merging and Release Rules. *Soft Computing*, 8 (2004), 1–5.
8. L. Pan, T.-O. Ishdorj, P Systems with Active Membranes and Separation Rules. *Journal of Universal Computer Science*, 10, 5 (2004), 630–649.
9. Ch. P. Papadimitriou, *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.
10. Gh. Păun, P Systems with Active Membranes: Attacking NP-Complete Problems. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 75–90.
11. Gh. Păun, *Membrane Computing: An Introduction*. Springer-Verlag, Berlin, 2002.
12. M. J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity Classes in Models of Cellular Computation with Membranes. *Natural Computing*, 2, 3 (2003), 265–285.