Replicative - Distribution Rules in P Systems with Active Membranes

Tseren-Onolt ISHDORJ[1,2] and Mihai IONESCU[2]

[1] Computer Science and Information Technology School
Mongolian State University of Education
Baga Toiruu-14, 210648 Ulaanbaatar, Mongolia
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain

Abstract. P systems (known also as *membrane systems*) are biologically motivated theoretical models of distributed and parallel computing. The two most interesting questions in the area are completeness (solving every solvable problem) and efficiency (solving a hard problem in feasible time). In this paper we define a general class of P systems covering some biological operations with membranes. We introduce a new operation, called *replicative-distribution*, into P systems with active membranes. This operation is well motivated from a biological point of view, and elegant from a mathematical point of view. It is both computationally powerful and efficient. More precisely, the P systems with active membranes using replicative-distribution rules can compute exactly what Turing machines can compute, and can solve **NP**-complete problems, particularly `SAT`, in linear time.

1 Introduction

Traditionally, theoretical computer science has played the role of a scout that explores novel approaches towards computing well in advance of other sciences. This did also occur in the case of membrane computation.

In the history of computing, electronic computers are only the latest in a long chain of man's attempts to use the best technology available for doing computations. While it is true that their appearance, some 50 years ago, has revolutionized computing, computing does not start with electronic computers, and there is no reason why it should end with them. Indeed, even electronic computers have their limitations: there is only so much data they can store and their speed thresholds determined by physical laws will soon be reached. The latest attempt to break down these barriers is to replace, once more, the tools for doing computations: instead of electrical use biological ones.

An important achievement in this direction was brought by Leonard Adleman in 1994, [1], when he surprised the scientific community by using the tools of molecular biology to solve a hard computational problem. Adleman's experiment, solving an instance of the Directed Hamiltonian Path Problem by manipulating DNA strands marked the first instance of a mathematical problem

being solved by biological means. The experiment provoked an avalanche of computer science/molecular biology/biochemistry/physics research, while generating at the same time a multitude of open problems.

The understanding of computations in nature – evolutionary computing, neural computing, and molecular computing – belong to the emerging area of *natural computing*, which is concerned with computing which go on in nature or is inspired by nature.

Membrane computing is a novel emerging branch of natural computing, introduced by Gheorghe Păun in [10]. This area starts from the observation that certain processes which take place in the complex structure of living cells can be considered as computations. 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 by various operations. For a detailed description of various P system models we refer to [12].

In membrane computing, P systems with active membranes have a special place, because they provide biologically inspired tools to solve computationally hard problems. Using the possibility to divide or separate membranes, one can create an exponential working space in linear time, which can then be used in a parallel computation for solving, e.g., **NP**-complete problems in polynomial or even linear time. Details can be found in [2, 11, 12], as well as in the comprehensive page from the web address `http://psystems.disco. unimib.it.` Informally speaking, in P systems with active membranes one uses the following types of rules: (a_0) multiset rewriting rules, (b_0) rules for introducing objects into membranes, (c_0) rules for sending objects out of membranes, (d_0) rules for dissolving membranes, (e_0) rules for dividing elementary membranes, and (f_0) rules for dividing non-elementary membranes, see [3]. In these rules, a single object is involved. The following rules are introduced in [2]: (g_0) membrane merging rules, (h_0) membrane separation rules, and (i_0) membrane release rules, whose common feature is that they involve multisets of objects.

In this paper, we introduce a new developing rule in P systems, called *replicative-distribution rule*, which is motivated from the specific structure and functioning of living neural-cell (neuron). The universality and efficiency related to replicative-distribution rules are investigated here.

Let us briefly mention the biological background of our new developing rules. A neuron has a *body*, the *dendrites*, which form a very fine filamentary bush around the body of the neuron, and the *axon*, a unique, long filament, which in turn also ends with a fine filamentous bush; each of the filaments from the end of the axon is terminated with a small bulb. It is by means of these end-bulbs and the dendrites that the neurons are linked to each other: the impulses are sent through the axon, from the body of the neuron to the end-bulbs, and the end-bulbs transmit the impulses to the neurons whose dendrites they touch. Such a contact junction between an end-bulb of an axon and dendrites of another neuron is called cleft. An end-bulb releases an impulse into cleft, the impulse is replicated in the cleft and distributed into the connected dendrites. Moreover, in the axon of the neuron, chemicals are replicated at the so-called Ranvier nodes

and transmitted to the adjacent nodes in opposite directions through the axon. For more details about neural biology, we refer to [17].

2 Preliminaries

We assume the reader to be familiar to the fundamentals of formal language theory and complexity theory, for instance, from [9, 15, 16], as well as to the basics of membrane computing, from [12]. We only mention here some notions and results from formal language theory, complexity theory, as well as from membrane computing, which are used in this paper.

2.1 Formal Languages

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, the length of a word w by $|w|$, and the number of ocuurences of a symbol a in w by $|w|_a$. The concatenation of two words x and y is denoted by $x \cdot y$ or simply xy.

A *language* over Σ is a (possibly infinite) set of words over Σ. The language consisting of all words over Σ is denoted by Σ^*, and Σ^+ denotes the language $\Sigma^* - \{\lambda\}$. A set of languages containing at least one language not equal to $\oslash$ or $\{\lambda\}$ is also called a family of languages.

We denote by `REG`, `LIN`, `CF`, `CS`, `RE` the families of languages generated by regular, linear, context-free, context-sensitive, and of arbitrary grammars, respectively (`RE` stands for recursively enumerable languages). By `FIN` we denote the family of finite languages. The following strict inclusions hold:

$$\texttt{FIN} \subset \texttt{REG} \subset \texttt{LIN} \subset \texttt{CF} \subset \texttt{CS} \subset \texttt{RE}.$$

This is the Chomsky hierarchy.

For a family `FL` of languages, `NFL` denotes the family of length sets of languages in `FL`. Therefore, `NRE` is the family of Turing computable sets of natural numbers. For $a \in \Sigma$ and $x \in \Sigma^*$ we denote by $|x|_a$ the number of occurrences of a in x. Then, for $\Sigma = \{a_1, \cdots, a_n\}$, the *Parikh mapping* associated with Σ is the mapping on Σ^* defined by $\Psi_\Sigma(x) = (|x|_{a_1}, \cdots, |x|_{a_n})$ for each $x \in \Sigma^*$. The Parikh images of languages `RE` is denoted by $PsRE$ (this is the family of all recursively enumerable sets of vectors of natural numbers).

The multisets over a given finite support (alphabet) are represented by strings of symbols. The order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string. Clearly, using strings is only one of many ways to specify multisets. We suggest the readers refer to [4].

We will now introduce the notion of matrix grammars, used below in proofs.

A *matrix grammar with appearance checking* is a computationally universal rewriting system. Details can be found in [5]. For each matrix grammar there is an equivalent matrix grammar in the binary normal form.

A matrix grammar $G = (N, T, S, M, F)$ is in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, 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 applying once each rule of m to w 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 $MAT_{ac} = RE$.

2.2 P Systems with Active Membranes

In this subsection, we describe P systems with active membranes following the concept defined in [12], where more details can also be found.

A *membrane structure* is represented by a Venn diagram and is identified by a string of correctly matching parentheses, with a unique external pair of parentheses; this external pair of parentheses corresponds to the external membrane, called the *skin*. A membrane without any other membrane inside is said to be *elementary*. For instance, the structure in Figure 1 contains 8 membranes; membranes 3, 5, 6 and 8 are elementary. The string of parentheses identifying this structure is

$$\mu = [\,[\,[\,\,]_5[\,\,]_6]_2[\,\,]_3[\,[\,[\,\,]_8]_7]_4]_1.$$

All membranes are labeled; we have used here the numbers from 1 to 8. We say that the number of membranes is the *degree* of the membrane structure, while the height of the tree associated in the usual way with the structure is its *depth*. In the example above we have a membrane structure of degree 8 and of depth 3.

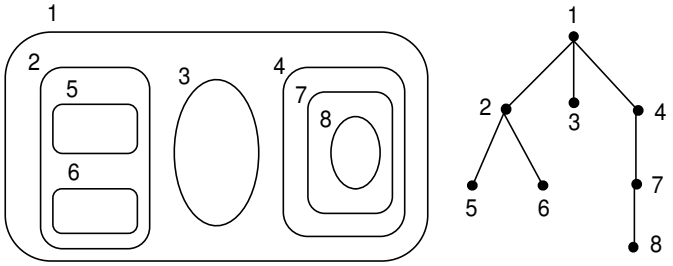

Figure 1. A membrane structure and its associated tree

The membranes delimit *regions* precisely identified by the membranes (the region of a membrane is delimited by the membrane and all membranes placed immediately inside it, if such a membrane exists). In these regions we place *objects*, which are represented by symbols of an alphabet. Several copies of the same object can be present in a region, so we work with *multisets* of objects.

We will now define the model which we work with: P systems with active membranes. A P system *with active membranes* (without electrical charges) is a construct

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

where:

- $m \geq 1$ is the initial degree of the system;
- O is the alphabet of *objects*;
- 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;
- R is a finite set of *developmental rules*, of the following forms:

  (a_0) $[\ a \to v\ ]_h$, for $h \in H, a \in O, v \in O^*$
  (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);

  (b_0) $a[\ \ ]_h \to [\ b\ ]_h$, for $h \in H, a, b \in O$
  (communication rules; an object is introduced in the membrane during this process);

  (c_0) $[\ a\ ]_h \to [\ \ ]_h b$, for $h \in H, a, b \in O$
  (communication rules; an objects sent out of the membrane during this process);

  (d_0) $[\ a\ ]_h \to b$, for $h \in H, a, b \in O$
  (dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

  (e_0) $[\ a\ ]_h \to [\ b\ ]_h [\ c\ ]_h$, for $h \in H, a, b, c \in O$
  (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);

  (f_0) $[\ a\ ]_h \to [\ b\ ]_h [\ c\ ]_h$, for $h \in H, a, b, c \in O$
  (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);

(g_0) $[\,]_{h_1}[\,]_{h_2} \to [\,]_{h_3}$, for $h_i \in H, 1 \leq i \leq 3$
(merging rules for elementary membranes; in reaction of two membranes, they are merged into a single membrane; the objects of the former membranes are put together in the new membrane);

(h_0) $[\,O]_h \to [\,U]_h[\,O-U]_h$, for $h \in H, U \subset O$
(separation rules for elementary membranes; the membrane is separated into two membranes with the same labels; the objects from U are placed in the first membrane, those from $U - O$ are placed in the other membrane);

(i_0) $[\,[\,O]_{h_1}]_{h_2} \to [\,]_{h_2}O$, for $h_1, h_2 \in H$
(release rule; the objects in a membrane are released out of a membrane, surrounding it, while the first membrane disappears).

The rules of types (a_0), (b_0), (c_0), (d_0), (e_0), and (f_0) are the polarizationless version of the corresponding rules in [12]; the rules of (g_0), (h_0), and (i_0) are introduced in [2]. (In all cases, the subscript 0 indicates the fact that we do not use polarization for membranes; in [11], [13] the membranes can have one of the "electrical charges negative, positive, neutral, represented by $-$, $+$, 0, respectively. Note that, following [3], we have omitted the label of the left parenthesis from a pair of parentheses which identifies a membrane.)

The rules of type (a_0) are applied in the parallel way (all objects which can evolve by such rules have to evolve), while the rules of types (b_0), (c_0), (d_0), (e_0), (f_0), (g_0), (h_0), and (i_0) are used sequentially, in the sense that one membrane can be used by at most one rule of these types at a time. In total, the rules are used in the non-deterministic maximally parallel manner: all objects and all membranes which can evolve, should evolve.

The result of a halting computation is the vector of natural numbers describing the multiplicity of objects expelled into the environment during the computation; the set of vectors computed in this way by all possible halting computations of Π is denoted by $Ps(\Pi)$. A P system is called *deterministic* if there is a single computation. A P system is called *confluent* if all of its computations reach the same halting configuration.

By $PsOP_m(r)$ we denote the family of sets $Ps(\Pi)$ computed as described above by P systems with at most m membranes using rules of types listed in r.

When the rules of a given type (α_0) are able to change the label(s) of the involved membranes, then we denote that type of rules by (α_0').

P systems with certain combinations of these rules are universal and efficient. Further details can be found in [2, 3, 8].

To understand what solving a problem in a semi-uniform/uniform way means, we briefly recall here some related notions. Consider a decisional problem X. A family $\Pi_X = (\Pi_X(1), \Pi_X(2), \cdots)$ 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 [12, 13].

3 Replicative-Distribution Rules

The biological motivations of *replicative-distribution* operations are mentioned in Section 1. Mathematically, we capture the idea of replicative-distribution rules as following:

(k_0) $a[\,]_{h_1}[\,]_{h_2} \to [\,u]_{h_1}[\,v]_{h_2}$, for $h_1, h_2 \in H, a \in O, u, v \in O^*$
(replicative-distribution rule (for sibling membranes); an object is replicated and distributed into inner two adjacent membranes);
(l_0) $[\,a[\,]_{h_1}]_{h_2} \to [\,[\,u]_{h_1}]_{h_2} v$, for $h_1, h_2 \in H, a \in O, u, v \in O^*$
(replicative-distribution rule (for nested membranes); an object is replicated and distributed into a directly inner membrane and outside the directly surrounding membrane).

The rules are applied non-deterministically, in the maximally parallel manner. Note that the multisets u and v might be empty.

As we have mentioned before, we use the primed versions to indicate the fact that the labels of membranes can be changed. The primed versions of replicative-distribution rules are of the following form:

(k_0') $a[\,]_{h_1}[\,]_{h_2} \to [\,u]_{h_3}[\,v]_{h_4}$ for $h_i \in H, 1 \leq i \leq 4$
(the label of both or only one membrane can be changed);
(l_0') $[\,a[\,]_{h_1}]_{h_2} \to [\,[\,u]_{h_3}]_{h_4} v$, for $h_i \in H, 1 \leq i \leq 4$
(the label of both or only one membrane can be changed).

3.1 Computational Universality

P systems with active membranes and with particular combinations of several types of rules can reach universality. Here, we show that P systems with active membranes and with only one type of rules, namely (l_0'), is Turing complete. The proof is based on the simulation of matrix grammars with appearance checking.

Theorem 1. $PsOP_4(l_0') = PsRE$.

Proof. It is enough to prove that any recursively enumerable set of vectors of non-negative integers can be generated by a P system with active membranes using rules of type (l_0') and four membranes.

Consider a matrix grammar $G = (N, T, S, M, F)$ with appearance checking, in the binary normal form, hence with $N = N_1 \cup N_2 \cup \{S, \#\}$ and with the matrices of the four forms introduced in Section 2.1. Assume that all matrices are injectively labeled with elements of a set B. Replace the rule $X \to \lambda$ from matrices of type 4 by $X \to f$, where f is a new symbol.

We construct the P system of degree 4

$$\begin{aligned}
\Pi &= (O, H, \mu, w_0, w_1, w_2, w_{X_{init}}, R),\\
O &= T \cup N_2 \cup \{A_m \mid A \in N_2, m \in B\} \cup \{c, c', c'', c''', \lambda, \#\},\\
H &= N_1 \cup \{X_m \mid X \in N_1, m \in B\} \cup \{0, 1, 2, f\},\\
\mu &= [\,[\,[\,[\,\,]_2]_1]_{X_{init}}]_0,\\
w_0 &= cA_{init}, w_{X_{init}} = w_1 = w_2 = \lambda.
\end{aligned}$$

and the set R containing the rules below.

The simulation of a matrix $m : (X \to Y, A \to x)$, with $X \in N_1, Y \in N_1 \cup \{f\}$, and $A \in N_2$, is done in two steps, using the following replicative-distribution rules:

1. $[\,A[\,\,]_X]_0 \to [\,[\,A_m]_{Y_m}]_0\lambda$,
2. $[\,A_m[\,\,]_1]_{Y_m} \to [\,[\,\lambda]_1]_Y x$.

The first rule of the matrix is simulated by the change of the label of membrane X, and the correctness of this operation is obvious (one cannot simulate one rule of the matrix without simulating at the same time also the other rule).

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

3. $[\,c[\,\,]_X]_0 \to [\,[\,c']_{Y_m}]_0\lambda$,
4. $[\,c'[\,\,]_1]_{Y_m} \to [\,[\,c'']_1]_{Y'_m}\lambda$,
5. $[\,A[\,\,]_{Y'_m}]_0 \to [\,[\,\#]_f]_0\lambda$,
   $[\,c''[\,\,]_2]_1 \to [\,[\,\lambda]_2]_1 c'''$,
6. $[\,c'''[\,\,]_1]_{Y'_m} \to [\,[\,\lambda]_1]_Y c$.

By using rule 3, object c replicates to c' and λ which are distributed, in the same time, as follows: c' enters membrane X changing its label to Y_m and λ is send out of the skin membrane. The second step (rule 4) makes c' to evolve to λ and c''; c'' will be sent to membrane 1 and λ gets out of membrane Y_m changing it to Y'_m. In the next step, if any copy of A is present, then, it introduces the trap-object $\#$ and the computation never stops. Otherwise, c'' following the same replicative-distribution rule transforms into λ and c''', which enter membranes 1 and Y'_m, respectively. The last computational step produces the result we were looking for by replicating c''' to λ and c and distributing λ to membrane 1 and c to the skin membrane, changing label Y_m to Y. Now, the process can be iterated having c in the skin membrane as in its initial configuration.

We also consider the following rules (applicable in the case A is present in the skin membrane):

7. $[\,\#[\,]_1]_f \to [\,[\,\lambda]_1]_f\#$,
8. $[\,\#[\,\,]_f]_0 \to [\,[\,\#]_f]_0\lambda$.

The equality $\Psi_T(L(G)) = Ps(\Pi)$ easily follows from the above explanations.

□

3.2 Efficiency Result Using Pre-computed Resources

The SAT problem (satisfiability of propositional formula in the conjunctive normal form) is probably the most known **NP**-complete problem [6]. It asks whether or not for a given formula in the conjunctive normal form there is a truth-assignment of variables such that the formula assumes the value *true*.

Let us consider a propositional formula in the conjunctive normal form:

$$\begin{aligned}\beta &= C_1 \wedge \cdots \wedge C_m,\\ C_i &= y_{i,1} \vee \cdots \vee y_{i,l_i},\ 1 \le i \le m, \text{ where}\\ y_{i,k} &\in \{x_j, \neg x_j \mid 1 \le j \le n\},\ 1 \le i \le m, 1 \le k \le l_i.\end{aligned}$$

The instance β of SAT will be encoded in the rules of P system by multisets v_j and v'_j of symbols, corresponding to the clauses satisfied by *true* and *false* assignment of x_j, respectively:

$$\begin{aligned}v_j &= \{c_i \mid x_j \in \{y_{i,k} \mid 1 \le k \le l_i\}, 1 \le i \le m\}, 1 \le j \le n,\\ v'_j &= \{c_i \mid \neg x_j \in \{y_{i,k} \mid 1 \le k \le l_i\}, 1 \le i \le m\}, 1 \le j \le n.\end{aligned}$$

A computation in a P systems with active membranes always starts from a given initial configuration, and we usually create an exponential workspace in linear time by membrane division, membrane creation, string replication, and membrane separation (all of them with biological motivation). In this subsection we use a different strategy (which is already discussed in [12]): we start from an arbitrarily large initial membrane structure, without objects placed in its regions, and we trigger a computation by introducing objects related to a given problem in a specified membrane. We use object replicative-distribution rules, as discussed in the previous sections. In this way, the number of objects can increase exponentially.

Theorem 2. *P systems with rules of types* (k'_0) *and* (l'_0)*, constructed in a semi-uniform manner, can solve* SAT *in linear time.*

Proof. Given a propositional formula β as above, we construct the P system

$$\begin{aligned}\Pi &= (O, H, \mu, w_a, w_b, w_c, w_d, w_e, w_0, w_1, w_2, w_{4n+5-m}, w_{4n+6}, R), \text{with}\\ O &= \{a_i \mid 0 \le i \le n\} \cup \{d_i \mid 1 \le i \le m\} \cup \{e_i \mid 0 \le i \le 4n+m+1\}\\ &\quad \cup \{t_i, f_i \mid 1 \le i \le n\} \cup \{c_i \mid 1 \le i \le m\} \cup \{\mathtt{yes}, \mathtt{no}, d, \lambda\},\\ \mu &= \underbrace{[\,[\,[\,[\,]_a]_b]_c[\cdots[\,]_d[\,]_e\cdots]_1[\cdots[\,]_d[\,]_e\cdots]_2]_0}_{2^{n+4}+2},\\ w_0 &= a_0, w_c = e_0, w_1 = w_2 = w_a = w_b = w_d = w_e = w_{4n+5-m} = w_{4n+6} = \lambda,\\ H &= \{0, \cdots, 4n+6, a, b, c, d, e\}.\end{aligned}$$

The membrane structure has to be a complete binary tree with $n+2$ internal levels for the constructions of truth value assignments except the membranes for global counting. In the skin membrane, 3 "nested" membranes with labels

a, b, and c are used to count the computations of the system. The "sibling membranes", those placed in the same upper membrane, directly under it, are labeled with 1 and 2. We consider the skin membrane as being level 0 (root) of our binary tree. It is obvious that on level 1 we have 2 (2^1) membranes, on level 2 we have 4 (2^2) membranes, and so on. The membranes on levels $1, \cdots, n$ are labeled 1 and 2, of level $n+1$ are labeled $4n+5-m$ and $4n+6$, and elementary membranes are labeled by d and e. The skin membrane is labeled 0.

We give the set of rules R accompanying them with their use explanations:

– Global control:

E1. $[\ e_i[\ ]_b]_c \rightarrow [\ [\ e_{i+1}]_b]_c\lambda$,

E2. $[\ e_i[\ ]_a]_b \rightarrow [\ [\ \lambda]_a]_b e_{i+1}, 0 \leq i \leq 4n+m-1$,

The control variables e_i count the computing steps in the "nested" control membranes. As we shall see at the end of the description of the whole algorithm, after $4n+m$ derivation steps in the corresponding P system Π the answer `yes` appears outside the skin membrane if the given satisfiability problem has a solution, whereas in the case that no solution exits, in one or two more steps the answer `no` appears in the environment.

The main task of the algorithm is accomplished in the generation phase where, for each possible truth assignment to the n variables. After $2n-1$ steps it will contain all the informations needed to decide whether it represents a solution to the given problem or not:

– Generation phase:

G1. $a_i[\ ]_1[\ ]_2 \rightarrow [\ a_{i+1}t_{i+1}]_3[\ a_{i+1}f_{i+1}]_4, 0 \leq i \leq n-1$,

G2. $t_i[\ ]_{i+2}[\ ]_{i+3} \rightarrow [\ t_i]_{i+4}[\ t_i]_{i+5}$,
$f_i[\ ]_{i+2}[\ ]_{i+3} \rightarrow [\ f_i]_{i+4}[\ f_i]_{i+5}, 1 \leq i \leq n$,

Starting the computation (rule G1 in skin membrane), object a_0 is replicated into objects a_1t_1 and a_1f_1, which are distributed into direct inner membranes with label 1 and 2 of level 2, changing the labels to 3 and 4.

Let us consider step i of the generation phase: By applying rule G1 in the membranes of level i, 2^i number of couples of objects a_it_i and a_if_i are produced and took place in the 2^i number of membranes, which will change the label from 1 or 2 to 3 or 4, respectively. Each membrane among the 2^{i-j} membranes on levels $i-j$, $1 \leq j \leq [\ i/2]$ of hierarchal binary tree structure contains couple of objects p_{i-j} and q_j, $(p_r, q_r \in \{t_r, f_r\})$, $1 \leq j \leq [\ i/2]$ if j is an odd number. Otherwise, in the membranes of $[\ i/2]$ th level only objects p_{i-j} are placed. Up to now, $t_1, \cdots, t_i, f_1, \cdots, f_i$, and a_i different 2^i number of objects have been produced and distributed in the membranes of levels between $i-j$ and i, $1 \leq j \leq [\ i/2]$ presenting the truth value assignments for variables $x_1, \cdots, x_i$ of β. Objects t_k, $1 \leq k \leq i$ correspond to the *true* value of variables x_k, and objects f_k correspond to the $false$ value of variables x_k, $1 \leq k \leq i$. In the next step, rule G1 is applied and objects $a_{i+1}t_{i+1}$ and $a_{i+1}f_{i+1}$ are produced and took place in inner membranes, changing the labels of them to 3 and 4. At the same time, rules

$$t_k[\ ]_{k+2}[\ ]_{k+3} \rightarrow [\ t_k]_{k+4}[\ t_k]_{k+5},$$

$$f_k[\,]_{k+2}[\,]_{k+3} \rightarrow [\,f_k]_{k+4}[\,f_k]_{k+5}$$

are applied simultaneously in each membranes of levels $i-j$, $1 \leq j \leq [\,i/2]$, objects t_k and f_k are replicated and distributed in one deeper level. Membrane labels are changed from $k+2$ and $k+3$ to $k+4$ and $k+5$, respectively, which guarantees that in each step only a single object, t_k or f_k, enters into a membrane, since active membranes work in sequential manner.

At the nth step of the computation, 2^n number of couple objects $a_n p_n$, $p_n \in \{t_n, f_n\}$, were in 2^n membranes with label 3 and 4 on nth level, then rule G1 will not apply anymore since there is no membrane with label 1 and 2 at the next level. The iteration is continued $n-1$ more steps, all objects t_k, f_k, $1 \leq k \leq n-1$ are within the membranes of level n, then, those membranes' label being $2n+1$ and $2n+2$. Therefore all possible 2^n truth assignments of variables $x_1, x_2, \cdots, x_n$ are generated and placed in the corresponding 2^n membranes. Objects t_i correspond to the *true* value of variables x_i, and objects f_i correspond to the $false$ value of variables $\neg x_i$.

G3. $t_i[\,]_{4n+5-m}[\,]_{4n+6} \rightarrow [\,v_i]_{4n+5-m}[\,d]_{4n+6}$,
$f_i[\,]_{4n+5-m}[\,]_{4n+6} \rightarrow [\,v_i']_{4n+5-m}[\,d]_{4n+6}, 1 \leq i \leq n.$

By using rule G3, in n steps, every object t_i and f_i evolve into objects c_i (corresponding to clauses C_i, satisfied by the *true* or $false$ values chosen for x_i) and "dummy" object d, then they are distributed into membranes with label $4n+5-m$ and $4n+6$ in one deeper level, respectively.

– Checking phase:

C1. $[\,c_i[\,]_d]_{4n+4-m+i} \rightarrow [\,[\,c_i]_d]_{4n+5-m+i} d_i, 1 \leq i \leq m.$

In the checking phase, by using rule C1, object $c_i, 1 \leq i \leq n$, is placed in membranes labeled $4n+5-m$ of level $n+1$, and replicated into object c_i and counter object d_i. Object c_i is sent into the direct inner elementary membrane with label d, which is on the deepest level $(n+2)$ of our membrane structure, and object d_i is sent out the surrounding membrane on nth level. Meanwhile, the label of the surrounding membrane is incremented by one. If at the beginning of the checking phase $c_1, \cdots, c_i$ are present $(1 \leq i < m)$, and c_{i+1} is absent, in the membrane, after $i+1$ steps rule C1 will no longer be applicable and the membrane will never change the label again. If all objects $c_i, 1 \leq i \leq m$, are present in some membrane, then after m steps, objects d_m are produced into the membranes with label $2n+1$ and $2n+2$ of level n.

– Output phase:

O1. $[\,d_m[\,]_{2n+5+2i}]_{1+2i} \rightarrow [\,[\,d]_{2n+5+2i}]_{2n+3+2i} d_m$,
O2. $[\,d_m[\,]_{2n+6+2i}]_{2+2i} \rightarrow [\,[\,d]_{2n+6+2i}]_{2n+4+2i} d_m$, $1 \leq i \leq n$
O3. $[\,d_m[\,]_{2n+5}]_0 \rightarrow [\,[\,d]_{2n+5}]_1 \texttt{yes}$,

If β has solutions, the process starts when objects d_m are placed in membranes with label $2n+1$ and/or $2n+2$ of level n. Object d_m is replicated to objects d and d_m, object d_m is sent out the current membrane, and "dummy" object d is

sent into the inner membrane with label $2n+5+2i$. The process is recurrently done following objects d_m through levels $n \cdots 1$ in n steps by using rule O1 and O2. During the output phase, in each membranes with label $1+2i$ and $2+2i$ possible taken at most two objects d_m, then one of them non-deterministically chosen send out the surrounding membrane, while membrane label is changed to $2n+3+2i$ and $2n+4+2i$. Then, in membranes $2n+3+2i$ and $2n+4+2i$ no rule can be applied. Thus, the system works fine in a sequential manner. However, in n steps, totally speaking in the $(4n+m-1)$th step of the computation, at most two objects d_m arrive in the skin membrane. Then rule O3 is applied, an object d_m ejects positive answer `yes` and changes skin membrane label to 1 in order to prevent further output. Thus, the formula is satisfiable and the computation stops. That was the $(4n+m)$th step of the whole computation.

E3. $[\ e_{4n+m(-1)}[\ ]_b]_c \to [\ [\ \lambda]_b]_c e_{4n+m(+1)}$,
E4. $[\ e_{4n+m(+1)}[\ ]_c]_0 \to [\ [\ \lambda]_c]_0 \texttt{no}$.

If β has no solution and if $4n+m-1$ is an odd step, counter object e_{4n+m-1} must be placed in the membrane a, then rules E3 and E4 are applied in two steps, the counter object e_{4n+m+1} will eject the correct answer `no` to the environment. Otherwise after one more step object e_{4n+m} will eject the correct answer `no` to the environment by applying rule E4. Since rule O3 did not apply (the case in which β has no solution), the label of the skin membrane is still 0, so rule E3 is applicable. The 3 "nested" control membranes guarantee that no object tries to cross the skin membrane at the same time with `yes`.

The labels of membranes of level i, in the constructing phase, are $3+2(i-1)$ and $4+2(i-1)$, $1 \le i \le n$, and in the output phase it would be $3+2(i+n)$ and $4+2(i+n)$, $1 \le i \le n$. □

Theorem 3. *P systems with rules of types (k_0') and (c_0'), constructed in a semi-uniform manner, can solve* `SAT` *in linear time.*

Proof. The proof of the theorem follows the idea of Theorem 2. Since rules of type (l_0') are not used in the proof, we do not need membranes of level $n+2$ in the checking phase, and we change the "nested" couple membrane structure by "eyes" structure for the global control.

We now construct the P system

$$
\begin{aligned}
\Pi &= (O, H, \mu, w_a, w_b, w_0, w_1, w_2, w_{4n+5-m}, w_{4n+6}, R), \text{with} \\
O &= \{a_i \mid 0 \le i \le n\} \cup \{d_i \mid 1 \le i \le m\} \cup \{e_i \mid 0 \le i \le 4n+m+1\} \\
&\quad \cup \{t_i, f_i \mid 1 \le i \le n\} \cup \{c_i \mid 1 \le i \le m\} \cup \{\texttt{yes}, \texttt{no}, d, \lambda\}, \\
\mu &= \underbrace{[\,[\,]_a[\,]_b[\cdots[\,]_{4n+5-m}[\,]_{4n+6}\cdots]_1[\cdots[\,]_{4n+5-m}[\,]_{4n+6}\cdots]_2]_0}_{2^{n+3}+1}, \\
w_0 &= a_0 e_0, w_1 = w_2 = w_a = w_b = w_{4n+5-m} = w_{4n+6} = \lambda, \\
H &= \{0, \cdots, 4n+6, a, b\}.
\end{aligned}
$$

The global control rules are as following:

– Global control in skin membrane:

E1. $e_i[\ ]_a[\ ]_b \to [\ e_{i+1}]_a[\ \lambda]_b$,

E2. $[\ e_i]_a \to [\ ]_a e_{i+1}, 0 \leq i \leq 4n+m+1$,

Here we use rules of types (k_0) and (c_0) for counting the computation steps of the system.

– Generation phase:

We reuse rules G1-G3 of the generation phase in Theorem 2, then generate 2^n number of truth-assignments in level n of the membrane structure.

– Checking phase:

C1. $[\ c_i]_{4n+4-m+i} \to [\ ]_{4n+5-m+i} d_i, 1 \leq i \leq m$.

In the checking phase of satisfiability truth-assignments of propositional formula, rules of type (c'_0) are used instead of rules (l'_0).

– Output phase:

O1. $[\ d_m]_{1+2i} \to [\ \ ]_{2n+3+2i} d_m$,

O2. $[\ d_m]_{2+2i} \to [\ \ ]_{2n+4+2i} d_m$, $1 \leq i \leq n$

O3. $[\ d_m]_0 \to [\ \ ]_1$`yes`,

E3. $[\ e_{4n+m(+1)}]_0 \to [\ \ ]_0$`no`.

If β has solutions, in n steps, totally speaking in $(4n+m-1)$th step of the computation, at most two objects d_m arrive in the skin membrane by using rules O1 and O2, then rule O3 is applied, an object d_m ejects positive answer `yes` and changes the skin label to 1 in order to prevent further output. Thus, the formula is satisfiable and the computation stops. That was the $(4n+m)$th step of the whole computation.

If β has no solution and if $4n+m-1$ is an odd step, after two more steps the counter object e_{4n+m+1} will eject the correct answer *no* to the environment. Otherwise, after one more step object e_{4n+m} will perform this operation. Since rule O3 did not apply (the case in which β has no solution), the label of the skin membrane is still 0, so rule E3 is applicable. □

3.3 Efficiency Result Using Membrane Division to Obtain Exponential Work Space

In Theorem 2 and Theorem 3 we have shown that the NP-complete problem `SAT` can be decided by a P system with active membranes in linear time with replicative-distribution rules of types (k'_0) and (l'_0) and replicative-distribution rules of type (k'_0) and communication rules of type (c'_0), respectively, using pre-computed exponential work space.

Here, we reuse the most investigated way to obtain exponential work space– membrane division. The following theorem shows that `SAT` can be solved by P systems with active membranes using the rules of types (f_0) and (l'_0), in linear time. We recall here the propositional formula β in Section 3.2.

Theorem 4. *P systems with rules of types* $(f_0), (l'_0)$*, constructed in a semi-uniform manner, can deterministically solve* SAT *in linear time with respect to the number of the variables and the number of clauses.*

Proof. We construct the P system

$$\begin{aligned}
\Pi &= (O, H, \mu, w_0, \cdots, w_7, R), \text{ with} \\
O &= \{d_i \mid 1 \le i \le m\} \cup \{a_i \mid 1 \le i \le n\} \\
&\quad \cup \{c_i \mid 1 \le i \le m\} \cup \{b_i \mid 0 \le i \le n\} \\
&\quad \cup \{e_i \mid 0 \le i \le 2n+m+4\} \cup \{\mathtt{yes}, \mathtt{no}\} \\
&\quad \cup \{t_i, f_i \mid 1 \le i \le n\}, \\
\mu &= [\,[\,[\,[\,\,]_3]_4]_2[\,[\,[\,\,]_6]_7]_5]_0, \\
w_2 &= a_1 \cdots a_n b_0, w_5 = e_0, w_0 = w_2 = w_3 = w_4 = w_6 = w_7 = \lambda \\
H &= \{i \mid 0 \le i \le 9\},
\end{aligned}$$

and the following rules (we accompany them with explanations about their use):

The global control rules are as follows:

– Global control:

E1. $[\, e_i[\,\,]_7]_5 \to [\,[\, e_{i+1}]_7]_5\lambda$,

E2. $[\, e_i[\,\,]_6]_7 \to [\,[\,\lambda]_6]_7 e_{i+1}, 0 \le i \le 2n+m+1$,

The "nested" membranes with label 5,7, and 6 are used only to globally control of the computation, and rules E1 and E2 are used to count the computation steps as we used in the proof of Theorem 2.

G1. $[\, a_i]_2 \to [\, t_i]_2[\, f_i]_2$, $1 \le i \le n$,

Using rule G1, with a_i non-deterministically chosen, we produce the truth values $true$ and $false$ assigned to variable x_i, placed in two separate copies of membrane 2. In this way, in n steps we assign truth values to all variables, hence we get all 2^n truth-assignments, placed in 2^n separate copies of membrane 2.

G2. $[\, b_i[\,\,]_4]_2 \to [\,[\, b_{i+1}]_4]_2\lambda$,
$[\, b_i[\,\,]_3]_4 \to [\,[\,\lambda]_3]_4 b_{i+1}$, for all $0 \le i \le n-1$,

G3. $[\, b_n[\,\,]_3]_4 \to [\,[\,\lambda]_3]_1\lambda$,

G4. $[\, b_n[\,\,]_4]_2 \to [\,[\,\lambda]_1]_2\lambda$,

Initially, object b_0 is placed in membrane 2. Rule G2 works simultaneously with division and increment the subscript of b_i by one in each step. If in the nth step of the computation, object b_n takes place in membrane 2, it was an odd number. If it was an even number, object b_n takes place in membrane 4. In the next step rule G3 or G4 perform, and change the label of membrane 4 to 1, while object b_n disappears. This ensure rule G5 will perform.

G5. $[\, t_i[\,\,]_1]_2 \to [\,[\, v_i]_1]_2\lambda$,
$[\, f_i[\,\,]_1]_2 \to [\,[\, v'_i]_1]_2\lambda$, $1 \le i \le n$.

– Checking phase:

C1. $[\ c_i[\ ]_3]_i \rightarrow [\ [\ c_i]_3]_{i+1} d_i, 1 \leq i \leq m.$

The checking phase idea is the same from the proof of Theorem 2.

– Output phase:

O1. $[\ d_m[\ ]_{m+1}]_2 \rightarrow [\ [\ \lambda]_{m+1}]_8 d_m,$

O2. $[\ d_m[\ ]_8]_0 \rightarrow [\ [\ \lambda]_8]_9 \texttt{yes},$

If β has solutions, after $2n + m + 2$ steps, objects d_m appear in the skin membrane using rules O1, and again one object d_m, non-deterministically chosen, ejects object `yes` into environment, while the skin label changes to 9 using rule O2 in order to prevent further output. Thus, the formula is satisfiable and the computation stops. That was the $(2n+m+3)$th step of the whole computation.

E3. $[\ e_{2n+m+2(3)}[\ ]_7]_5 \rightarrow [\ [\ \lambda]_7]_5 e_{2n+m+3(4)},$

E4. $[\ e_{2n+m+3(4)}[\ ]_5]_0 \rightarrow [\ [\ \lambda]_5]_0 \texttt{no}.$

If β has no solution and if $2n + m + 2$ is an odd step, after two more steps the counter object e_{2n+m+4} will eject the correct answer `no` to the environment. Otherwise, after one more step object e_{2n+m+3} will perform this operation. Since rule O2 did not apply (the case in which β has no solution), the label of the skin membrane is still 0, so rule E4 is applicable. □

4 Final Remarks

We have considered a new type of rules in P systems with active membranes: (k_0) and (l'_0) replicative-distribution rules with deep relations to cell biology. We have illustrated here how this type of rules can solve **NP**-complete problems in linear time using pre-computed resources and obtaining an exponential work space during the computation, by membrane division. Universality was also shown here, but we want to emphasize that we have used only one type of rules in our proof. However, in the efficiency results, we have used very few types of rules compared to the previous results in [2, 3, 7, 8, 11]. This reveals the fact that replicative-distribution type of rules is a powerful and efficient tool in P systems. The following problems are expecting a future work: What simulations of other classes of P systems with active membranes using these new types of rules can be obtained? What other computational hard problems can be solved with these types of rules in feasible time and space?

Acknowledgments. The first author acknowledges the State Training Fund of the Ministry of Science, Technology, Education and Culture of Mongolia. The work of second author was supported by the FPU fellowship from the Spanish Ministry of Education, Culture and Sport.

References

1. L.M. Adlmen, Molecular computation of solutions to combinatorial problems, *Science* v.266, Nov.1994, 1021–1024.
2. A. Alhazov, T.-O. Ishdorj, Membrane Operations in P Systems with Active Membranes, In: Gh. Păun, et all (eds.) *Second Brainstorming Week on Membrane Computing*, Sevilla, 2-7 February, 2004, Research Group in Natural Computing **TR 01/2004**, University of Sevilla, 37–44.
3. A. Alhazov, L. Pan, Gh. Păun, Trading Polarizations for Labels in P Systems with Active Membranes, submitted, 2003.
4. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa (eds.): *Multiset Processing*, LNCS 2235, Springer-Verlag, Berlin, 2001.
5. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
6. M.R. Garey, D.J. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.
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, Computing with membranes, *Journal of Computer and System Sciences*, 61(1), (2000), 108–143, and TUCS Research Report 208, 1998 (http://www.tucs.fi).
11. Gh. Păun, P Systems with Active Membranes: Attacking NP-Complete Problems, Journal of Automata, Languages and Combinatorics, 6, 1 (2001), 75–90.
12. Gh. Păun, *Computing with Membranes: An Introduction*, Springer-Verlag, Berlin, 2002.
13. 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.
14. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas*, Editorial Kronos, Sevilla, 2002.
15. A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.
16. A. Salomaa, G. Rozenberg (eds.), *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
17. G.M. Shepherd, *Neurobiology*, Oxford University Press, NY Oxford, 1994.