Polarizationless P Systems with Active Membranes Working in the Minimally Parallel Mode

Rudolf Freund[1], Gheorghe Păun[2,3], Mario J. Pérez-Jiménez[3]

[1] Institute of Computer Languages
Vienna University of Technology
Favoritenstr. 9, Wien, Austria
rudi@emcc.at
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
george.paun@imar.ro, gpaun@us.es
[3] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Summary. We investigate the computing power and the efficiency of P systems with active membranes without polarizations, working in the minimally parallel mode. We prove that such systems are computationally complete and able to solve **NP**-complete problems even when the rules are of a restricted form, e.g., for establishing computational completeness we only need rules handling single objects and no division of non-elementary membranes is used.

1 Introduction

P systems with active membranes basically use five types of rules: (a) evolution rules, by which a single object evolves to a multiset of objects, (b) send-in, and (c) send-out rules, by which an object is introduced in or expelled from a membrane, maybe modified during this operation into another object, (d) dissolution rules, by which a membrane is dissolved, under the influence of an object, which may be modified into another object by this operation, and (e) membrane division rules; this last type of rules can be used both for elementary and non-elementary membranes, or only for elementary membranes. As introduced in [10], all these types of rules also use polarizations for membranes, "electrical charges" $+, -, 0$, controlling the application of the rules.

Systems with rules of types (a), (b), (c) were shown to be equivalent in computational power with Turing machines [11], even when using only two polarizations

[4], while P systems using all types of rules were shown to be universal even without using polarizations [1]. Another important class of results concerns the possibility of using P systems with active membranes to provide polynomial solutions to computationally hard problems. Several papers have shown that both decision and numerical **NP**-complete problems can be solved in a polynomial time (often, even linear time) by means of P systems with three polarizations [11], [13], then the number of polarizations was decreased to two [3], [4]. The systems constructed in these solutions use only division of elementary membranes. At the price of using division also for non-elementary membranes, the polarizations can be completely removed, [2].

All papers mentioned above apply the rules in the maximally parallel mode: in each step, the assignment of objects to the rules in the chosen multiset of rules to be applied in parallel is maximal, i.e., no further rule could be added to this chosen multiset of rules in such a way that the rules in the resulting extended multiset still could be applied in parallel. Recently, [5] a more relaxed strategy of using the rules was introduced, the so-called minimal parallelism: in each step, the assignment of objects to the rules in the chosen multiset of rules to be applied in parallel does not allow for extending it by any rule out of a set of rules from which no rule has been chosen so far for this multiset of rules. This introduces an additional degree of non-determinism in the system evolution, but still computational completeness and polynomial solutions to `SAT` were obtained in the new framework by using P systems with active membranes, with three polarizations and division of only elementary membranes.

In this paper we continue the study of P systems working in the minimally parallel way, and we prove that the polarizations can be avoided, at the price of using all five types of rules for computational completeness and the division of non-elementary membranes for computational efficiency. Moreover, in the proof for establishing computational completeness we restrict the form of the rules to handling only single objects in all types of rules (we call this the one-normal form for P systems with active membranes).

2 Prerequisites

We suppose that the reader is familiar with the basic elements of Turing computability [6], and of membrane computing [11]. We here, in a rather informal way, introduce only the necessary notions and notation.

For an alphabet A, by A^* we denote 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 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 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").

In our proofs showing computational completeness 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 labeled instructions which specify how the numbers stored in registers can change, and which instruction should follow after the instruction just carried out. There are three types of instructions:

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

A *register machine* is a construct $M = (n, B, l_0, l_h, I)$, 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 I is the set of instructions. Each label of B labels only one instruction from I, thus precisely identifying it. A register machine M generates a set $N(M)$ of numbers in the following way: having initially all registers empty (i.e., storing the number zero), start with the instruction labeled 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 therefore 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 [8]) that in this way we can compute all the sets of numbers which are Turing-computable, even using register machines with only three registers as well as registers two and three being empty whenever the register machine halts.

A register machine can also work in the accepting mode. The number to be accepted is introduced in register 1, with all other registers being empty. We start computing with the instruction labeled 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 request the register machines to be deterministic, namely with all instructions $l_i : (\texttt{ADD}, l_j, l_k)$ having $l_j = l_k$. It is known that deterministic accepting register machines characterize NRE even with only three registers and all registers being empty whenever the register machine halts.

3 P Systems with Active Membranes

We first introduce the P systems with active membranes in the general form, and then we describe the restricted version we investigate in this paper.

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, labeled (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) $[_h a\,]_h^{e_1} \to b[_h\,]_h^{e_2}$,
   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. $h_o \in H$ or $h_o = env$ indicates the output region.

In the maximally parallel mode, in each step (a global clock is assumed) we apply multisets of rules in such a way that no further rule can be applied to the remaining objects or membranes. In each step, each object and each membrane can be involved in only one rule. Because this way of using the rules is standard in membrane computing, we do not give further details, and we present only the minimally parallel mode, which here means the following:

All the rules of any type involving a membrane h constitute 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 $a[_h\,]_h^{e_1} \to [_h a]_h^{e_2}$, 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, we choose a multiset of rules taken from the sets $R_{h,i}$, $h \in H$, in such a way that after having assigned objects to all the rules in this multiset, no rule from any of the sets $R_{h,i}$ from which no rule has been taken so far, could be used in addition.

Whenever relevant, 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 be used 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 region h_o at the end of the computation; this is a region of the system if $h_o \in H$ (and in this case, for a computation to be successful, exactly one membrane with label h_o should be present in the halting configuration), or it is the environment if $h_o = env$.

We shall also consider the following normal form for P systems with active membranes: A system Π is said to be in the *one-normal form* if the membranes have no polarization (it is the same as saying that always all membranes have the same polarization, say 0, which therefore is irrelevant and thus omitted) and the rules are of the forms $[_h a \to b]_h$, $a[_h\,]_h \to [_h b]_h$, $[_h a\,]_h \to b[_h\,]_h$, $[_h a\,]_h \to b$, and $[_h a\,]_h \to [_h b\,]_h [_h c\,]_h$, for $a, b, c \in O$, such that $a \neq b, a \neq c, b \neq c$. Note that the labels of membranes are never changed.

The set of numbers generated in the minimally parallel way by a system Π is denoted by $N_{min}(\Pi)$. The family of sets $N_{min}(\Pi)$, generated by systems with

rules of the non-restricted form, having initially at most n_1 membranes and using configurations with at most n_2 membranes during any computation is denoted by $N_{min}OP_{n_1,n_2}((a),(b),(c),(d),(e))$; when a type of rules is not used, it is not mentioned in the notation. If any of the parameters n_1, n_2 is not bounded, then it is replaced by $*$. If the systems do not use polarizations for membranes, then we write $(a_0),(b_0),(c_0),(d_0),(e_0)$ instead of $(a),(b),(c),(d),(e)$. When the system Π is in the one-normal form, then we write $N_{min}OP_{n_1,n_2}((a_1),(b_1),(c_1),(d_1),(e_1))$.

When considering families of numbers generated by P systems with active membrane working in the maximally parallel mode, the subscript min is replaced by max in the previous notations. For precise definitions, we refer to [11] and to the papers mentioned below.

4 Computational Completeness

The following results are well known:

Theorem 1. (i) $N_{max}OP_{3,3}((a),(b),(c)) = NRE$, *[11].*
(ii) $N_{max}OP_{*,*}((a_0),(b_0),(c_0),(d_0),(e_0)) = NRE$, *[1].*
(iii) $N_{min}OP_{3,3}((a),(b),(c)) = NRE$, *[5].*

In turn, the following inclusions follow directly from the definitions:

Lemma 1. $N_{mode}OP_{n_1,n_2}((a_1),(b_1),(c_1),(d_1),(e_1)) \subseteq$
$N_{mode}OP_{n_1,n_2}((a_0),(b_0),(c_0),(d_0),(e_0)) \subseteq$
$N_{mode}OP_{n_1,n_2}((a),(b),(c),(d),(e))$,
for all $n_1, n_2 \geq 1$, *mode* $\in \{max, min\}$.

We now improve the equalities from Theorem 1 in certain respects, starting with proving the computational completeness of P systems with active membranes in the one-normal form when working in the maximally parallel mode, and then we extend this result to the minimal parallelism.

Theorem 2. $N_{max}OP_{n_1,*}((a_1),(b_1),(c_1),(d_1),(e_1)) = NRE$, *for all* $n_1 \geq 5$.

Proof. We only prove the inclusion

$$NRE \subseteq N_{max}OP_{5,*}((a_1),(b_1),(c_1),(d_1),(e_1)).$$

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

$$\Pi = (O, H, \mu, w_s, w_h, w_1, w_2, w_3, R, env),$$

of the initial degree 5, with

$$
\begin{aligned}
O &= \{d_i \mid 0 \le i \le 5\} \cup \{g, \#, \#', p, p', p'', c, c', c''\} \cup B \cup \{l' \mid l \in B\} \\
&\quad \cup \{l_{iu} \mid l_i \text{ is the label of an ADD instruction in } I, 1 \le u \le 4\} \\
&\quad \cup \{l_{iu0} \mid l_i \text{ is the label of a SUB instruction in } I, 1 \le u \le 4\} \\
&\quad \cup \{l_{iu+} \mid l_i \text{ is the label of a SUB instruction in } I, 1 \le u \le 6\}, \\
H &= \{s, h, 1, 2, 3\}, \\
\mu &= [_s[_1\,]_1\,[_2\,]_2\,[_3\,]_3[_h\,]_h]_s, \\
w_s &= l_0 d_0, w_\alpha = \lambda, \text{ for all } \alpha \in H - \{s\},
\end{aligned}
$$

and with the rules constructed as described below.

The value stored in a register $r = 1, 2, 3$ of M, in Π is represented by the number of copies of membranes with label r plus one (if the value of the register r is k, then we have $k + 1$ membranes with label r). The membrane with label h is auxiliary, it is used for controlling the correct simulation of instructions of M by computations in Π. Each step of a computation in M, i.e., using an ADD or a SUB instruction, corresponds to six steps of a computation in Π. We start with all membranes being empty, except for the skin region, which contains the initial label l_0 of M and the auxiliary object d_0. If the computation in M halts, that is, the object l_h appears in the skin region of Π, then we pass to producing one object c for each membrane with label 1 present in the system, except one; in this way, the number of copies of c sent to the environment represents the correct result of the computation in M.

We first indicate the rules used in each of the six steps of simulating instructions ADD and SUB, and then we present the rules for producing the result of the computation; in each configuration, only the membranes and the objects relevant for the simulation of the respective instruction are specified. In particular, we ignore the "garbage" object g, because once introduced it remains idle for the whole computation.

The **simulation of an instruction** $l_i : (\texttt{ADD}(r), l_j, l_k)$ uses the rules from Table 1.

Table 1. The simulation of an ADD instruction

Step	Main rules	Auxiliary rules	Configuration
–	–	–	$[_s l_i d_0\,[_r\,]_r \cdots\, [_h\,]_h]_s$
1	$l_i[_r\,]_r \to [_r l_i']_r$	$[_s d_0 \to d_1]_s$	$[_s d_1\,[_r l_i']_r \cdots\, [_h\,]_h]_s$
2	$[_r l_i']_r \to [_r l_{i1}]_r [_r g]_r$	$[_s d_1 \to d_2]_s$	$[_s d_2\,[_r l_{i1}]_r [_r g]_r \cdots\, [_h\,]_h]_s$
3	$[_r l_{i1}]_r \to l_{i2}[_r\,]_r$	$d_2[_h\,]_h \to [_h d_3]_h$	$[_s l_{i2}\,[_r\,]_r [_r\,]_r \cdots\, [_h d_3]_h]_s$
4	$[_s l_{i2} \to l_{i3}]_s$	$[_h d_3]_h \to d_4[_h\,]_h$	$[_s l_{i3} d_4[_r\,]_r [_r\,]_r \cdots\, [_h\,]_h]_s$
5	$[_s l_{i3} \to l_{i4}]_s$	$[_s d_4 \to d_5]_s$	$[_s l_{i4} d_5[_r\,]_r [_r\,]_r \cdots\, [_h\,]_h]_s$
6	$[_s l_{i4} \to l_t]_s, t \in \{j, k\}$	$[_s d_5 \to d_0]_s$	$[_s l_t d_0[_r\,]_r [_r\,]_r \cdots\, [_h\,]_h]_s$

The label object l_i enters into the correct membrane r (even if the register r is empty, there is at least one membrane with label r) and in the next step divides it. The object l_{i2} exits the newly produced membrane, but g remains inside; l_{i2} will evolve three further steps, just to synchronize with the evolution of the auxiliary objects $d_u, u \geq 0$, and the auxiliary membrane h, so that in the sixth step we end with the label l_j or l_k of the next instruction to be simulated present in the skin membrane, together with d_0; note that the number of copies of membrane r was increased by one.

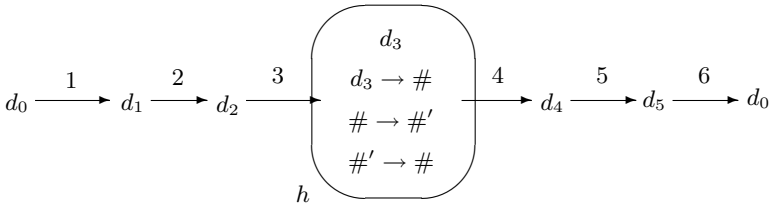

Fig. 1. The evolution of auxiliary objects

The auxiliary objects $d_u, u \geq 0$, and the auxiliary membrane h are used in the simulation of SUB instructions, as we will see immediately; in step 4, there also are other rules to be used in membrane h, introducing the trap-object #, but using such rules will make the computation never halt, hence, they will not produce an unwanted result.

The evolution of objects $d_u, u \geq 0$, is represented graphically in Figure 1; membrane h is only used in step 3, to send an object inside, and in the next step, for sending an object out of it. On each arrow we indicate the step, according to Table 1; the steps made by using rules of types (b), (c) are pointed out by drawing the respective arrows crossing the membranes, thus suggesting the *in/out* actions.

The way the "ADD module" works is suggested in Figure 2; the division operation from step 2 is indicated by a branching arrow.

The **simulation of an instruction** $l_i : (\texttt{SUB}(r), l_j, l_k)$ is done with the help of the auxiliary membrane h. The object l_i enters a membrane r (there is at least one copy of it) and divides it, and on this occasion makes a non-deterministic choice between trying to continue as having register r non-empty or as having it empty. If the guess has been correct, then the correct action is done (decrementing the register in the first case, doing nothing in the second case) and the correct next label is introduced, i.e., l_j or l_k, respectively. If the guess has not been correct, then the trap object # is introduced. For all membranes x of the system we consider the rules $[_x \# \to \#']_x$, $[_x \#' \to \#]_x$, hence, the appearance of # will make the computation last forever.

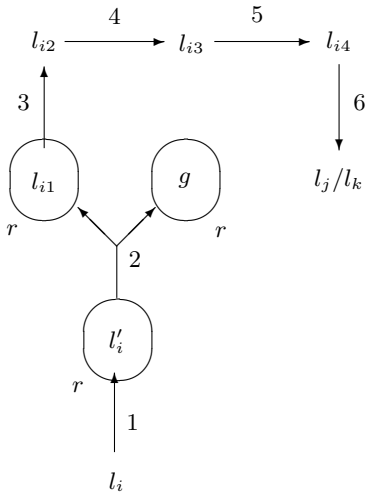

Fig. 2. The work of the ADD module

In Table 2 we present the rules used in the case of guessing that the register r is not empty. Like in the case of simulating an ADD instruction, the auxiliary objects and membranes do not play any role in this case.

Table 2. The simulation of a SUB instruction, guessing that register r is not empty

Step	Main rules	Auxiliary rules	Configuration
–	–	–	$[_s l_i d_0\ [_r\ ]_r \cdots [_h\ ]_h]_s$
1	$l_i[_r\ ]_r \to [_r l'_i]_r$	$[_s d_0 \to d_1]_s$	$[_s d_1\ [_r l'_i]_r \cdots [_h\ ]_h]_s$
2	$[_r l'_i]_r \to [_r l_{i1+}]_r [_r l_{i2+}]_r$	$[_s d_1 \to d_2]_s$	$[_s d_2 [_r l_{i1+}]_r [_r l_{i2+}]_r \cdots [_h\ ]_h]_s$
3	$[_r l_{i1+}]_r \to l_{i3+}$ $[_r l_{i2+}]_r \to l_{i4+}[_r\ ]_r$	$d_2[_h\ ]_h \to [_h d_3]_h$	$[_s l_{i3+} l_{i4+} [_r\ ]_r \cdots [_h d_3]_h]_s$
4	$l_{i3+}[_r\ ]_r \to [_r g]_r$ $l_{i4+}[_r\ ]_r \to [_r l_{i5+}]_r$ or $[_s l_{i3+} \to \#]_s, [_s l_{i4+} \to \#]_s$	$[_h d_3]_h \to d_4[_h\ ]_h$	$[_s d_4 [_r l_{i5+}]_r [_r g]_r \cdots [_h\ ]_h]_s$ $[_s d_4 \# [_r\ ]_r \cdots [_h\ ]_h]_s$
5	$[_r l_{i5+}]_r \to l_{i6+}$	$[_s d_4 \to d_5]_s$	$[_s d_5 l_{i6+} [_r\ ]_r \cdots [_h\ ]_h]_s$
7	$[_s l_{i6+} \to l_j]_s$	$[_s d_5 \to d_0]_s$	$[_s l_j d_0 [_r\ ]_r \cdots [_h\ ]_h]_s$

In step 2 we divide the membrane r containing the object l'_i and the objects l_{i1+}, l_{i2+} are introduced in the two new membranes. One of them is immediately dissolved, thus the number of copies of membrane r remains unchanged; the objects l_{i3+}, l_{i4+} are introduced in this step. In the next step (the fourth one of the simulation), objects l_{i3+}, l_{i4+} look for membranes r in the skin region. If both of them find such membranes – and this is the correct/desired continuation – then both of them enter such membranes; l_{i3+} becomes g and l_{i4+} becomes l_{i5+}. If only

one of them finds a membrane r, then the other one has to evolve to object # in the skin membrane and the computation never halts.

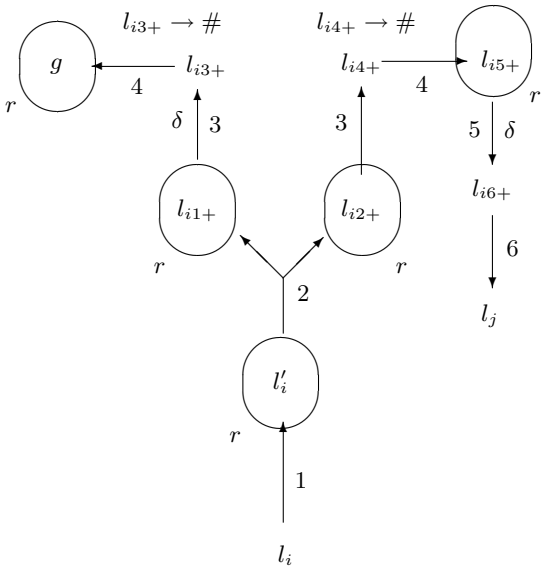

Fig. 3. The simulation of a SUB instruction when guessing that the register is non-empty

If we had enough membranes r and l_{i3+}, l_{i4+} went there, then in the next step l_{i5+} exits and is changed to l_{i6+}. In the sixth step, l_{i6+} becomes l_j (simultaneously with introducing d_0), and this completes the simulation. Thus, the computation continues without having # present in the system if and only if the guess made in step 2 has been correct, i.e., if register r has been non-empty. Because in step 5 a membrane r was dissolved, the number of these membranes was decreased by one, and this corresponds to subtracting one from register r.

Figure 3 indicates the evolution of objects l_{iu+} in the six steps mentioned in Table 2; when a dissolving operation is used, this is indicated by writing δ near the respective arrow.

However, in step 2, instead of the rule $[_r l'_i]_r \to [_r l_{i1+}]_r [_r l_{i2+}]_r$ we can use the rule $[_r l'_i]_r \to [_r l_{i10}]_r [_r l_{i20}]_r$, with the intention to simulate the subtract instruction in the case when the register r is empty – that is, only one membrane with label r is present in the system. The rules used in the six steps of the simulation are given in Table 3.

In this case, the auxiliary objects $d_u, u \geq 0$, and the auxiliary membrane h play an essential role.

Table 3. The simulation of a SUB instruction, guessing that register r is empty

Step	Main rules	Auxiliary rules	Configuration
–	–	–	$[_s l_i d_0\ [_r\ ]_r \cdots [_h\ ]_h]_s$
1	$l_i[_r\ ]_r \to [_r l_i']_r$	$[_s d_0 \to d_1]_s$	$[_s d_1\ [_r l_i']_r \cdots [_h\ ]_h]_s$
2	$[_r l_i']_r \to [_r l_{i10}]_r [_r l_{i20}]_r$	$[_s d_1 \to d_2]_s$	$[_s d_2\ [_r l_{i10}]_r [_r l_{i20}]_r \cdots [_h\ ]_h]_s$
3	$[_r l_{i10}]_r \to l_{i30}$ $[_r l_{i20}]_r \to l_{i40}[_r\ ]_r$	$d_2[_h\ ]_h \to [_h d_3]_h$	$[_s l_{i30} l_{i40}\ [_r\ ]_r \cdots [_h d_3]_h]_s$
4	$[_r l_{i30}]_r \to [_r g]_r$, l_{i40} waits or $l_{i40}[_r\ ]_r \to [_r \#]_r$ or $l_{i40}[_h\ ]_h \to [_h l_k']_h$	$[_h d_3]_h \to d_4[_h\ ]_h$ $[_h d_3 \to \#]_h$	$[_s l_{i40} d_4 [_r g]_r \cdots [_h\ ]_h]_s$ $[_s d_4 [_r \#]_r \cdots [_h\ ]_h]_s$ $[_s [_r g]_r \cdots [_h l_k' \#]_h]_s$
5	$l_{i40}[_h\ ]_h \to [_h l_k']_h$	$[_s d_4 \to d_5]_s$	$[_s d_5 [_r\ ]_r \cdots [_h l_k']_h]_s$
6	$[_h l_k']_h \to l_k[_h\ ]_h$	$[_s d_5 \to d_0]_s$	$[_s l_k d_0 [_r\ ]_r \cdots [_h\ ]_h]_s$

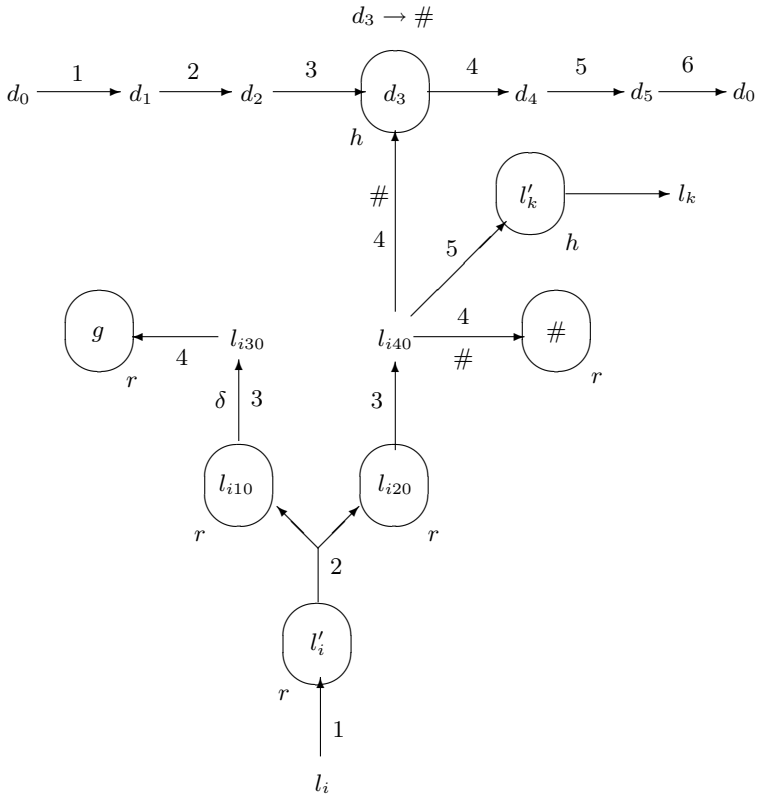

Fig. 4. The simulation of a SUB instruction when guessing that the register is empty

Until step 3 we proceed exactly as above, but now we introduce the objects l_{i30}, l_{i40}. The first one can enter the available membrane r, there evolving to g. If

there is a second membrane r, i.e., if the guess has been incorrect, then the rule $l_{i40}[_r\ ]_r \to [_r\#]_r$ should be used simultaneously (step 4), and the computation never ends. If there is no second membrane r, then in step 4 l_{i40}, can also enter membrane h, but then the trap object is produced here by the rule $[_h d_3 \to \#]_h$. The only way not to introduce the object # is (i) not to have a second membrane r, (ii) to use the rule $[_h d_3]_h \to d_4[_h\ ]_h$, thus preventing the use of the rule $l_{i40}[_h\ ]_h \to [_h l'_k]_h$, and (ii) not sending l_{i40} to the unique membrane r. This means that during step 4 the object l_{i40} should wait unchanged in the skin region.

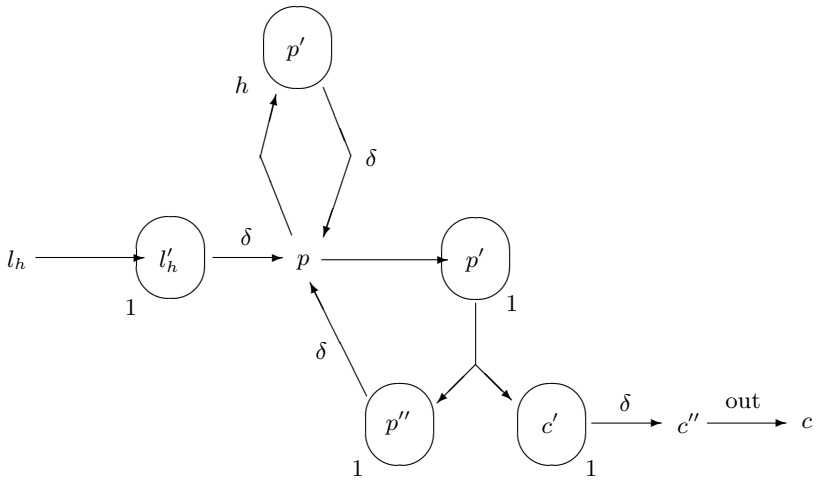

Fig. 5. The end of computations

In the next step, l_{i40} can enter membrane h (or the unique membrane r, but it becomes # there). In the next step, l_k is released from membrane h, at the same time with producing d_0, hence, the simulation is completed correctly and the system can pass to simulating another instruction.

The six steps of the computation are shown in Figure 4, this time with the evolution of the auxiliary objects being indicated, too. The arrows marked with # correspond to moves which are not desired.

For both guesses, the simulation of the SUB instruction works correctly, and the process can be iterated.

If the computation in M halts, i.e., if l_h is reached, i.e., if the object l_h is introduced in the skin region, we can start to use the following rules:

$$l_h[_1\ ]_1 \to [_1 l'_h]_1,$$
$$[_1 l'_h]_1 \to p,$$
$$p[_1\ ]_1 \to [_1 p']_1,$$
$$[_1 p']_1 \to [_1 p'']_1 [_1 c']_1,$$

$$[_1 p'']_1 \to p,$$
$$[_1 c']_1 \to c'',$$
$$[_s c'']_s \to c[_s\ ]_s,$$
$$p[_h\ ]_h \to [_h p']_h,$$
$$[_h p']_h \to p.$$

The object l_h dissolves one membrane with label 1 (thus, the remaining membranes with this label are now as many as the value of register 1 of M), and gets transformed into p. This object enters each membrane with label 1, divides it, reproduces itself (after passing through p' and p'') and also produces a copy of the object c''; this happens when dissolving the two membranes with label 1 obtained by division (rule $[_1 p']_1 \to [_1 p'']_1 [_1 c']_1$), hence, the number of copies of the membrane with label 1 has decreased by one. The object c'' immediately exits the system, thereby being changed into c.

At any time, the object p can also enter membrane h and then dissolves it. The computation can stop only after having dissolved all membranes with label 1 (i.e., a corresponding number of copies of c has been sent out) and after having dissolved the membrane h, hence, the evolution of objects $d_u, u \geq 0$, also stops.

The function of this final module is described in Figure 5.

Consequently, $N(M) = N_{max}(\Pi)$, and this concludes the proof. □

Theorem 3 gives the same result as Theorem 3 from [1], with the additional constraint to have evolution rules with exactly one object on the right-hand side (neither producing more objects, nor erasing objects, as it is the case in [1]).

The previous proof can easily be changed in order to obtain the computational completeness of P systems in the one-normal form also in the case of minimal parallelism. Specifically, we can introduce additional membranes in order to avoid having two or more evolution rules to be applied in the same region or one or more evolution rules and one rule of types (b) – (e) which involve the same membrane (these are the only situations where the minimal parallelism does not correspond to the maximal parallelism; note that all rules of types (b) – (e) are associated with membranes which are involved in the use of rules, hence, no two rules of these types can be applied for the same membrane).

In the previous construction, situations as above which must be avoided are the following ones:

- Rules $[_s l_{i3+} \to \#]_s$ and $[_s l_{i4+} \to \#]_s$, in step 4 of the simulation of SUB for the guess of a non-empty register. However, no other rule is applicable in the skin region at that time. Applying at least one rule of this type means introducing the trap object, hence, applying one of these rules is the same for the fate of the computation as applying all possible rules of that kind.
- Steps 5 and 6 of the auxiliary module (see again Figure 1) are performed in the skin region, in parallel with steps of modules from Figures 2 and 3. This can be avoided by introducing one further auxiliary membrane, h', in the skin

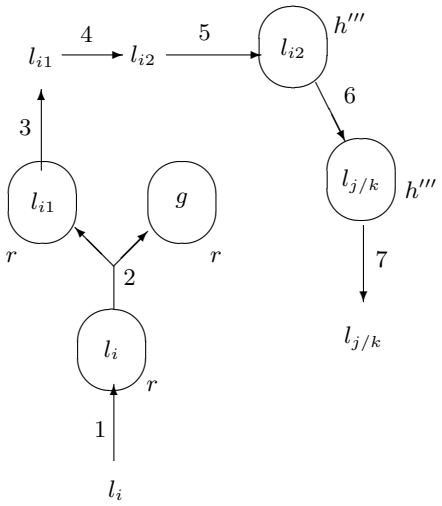

Fig. 6. The work of module ADD, with equal objects in the rules

region, and replacing the rules $[_s d_4 \to d_5]_s$, $[_s d_5 \to d_0]_s$ (both of them from R_s) by the rules $d_4[_{h'}\,]_{h'} \to [_{h'} d_5]_{h'}$ and $[_{h'} d_5]_{h'} \to d_0[_{h'}\,]_{h'}$. These rules are associated with membrane h', hence, they should be used in parallel with the rules from the skin region.

– A further case when the maximal parallelism is important in the previous construction is in step 4 of simulating a SUB instruction for the guess that the register is empty (Figure 4): if rule $l_{i40}[_h\,]_h \to [_h l'_k]_h$ is used in step 4, then also the rule $[_h d_3 \to \#]_h$ must be used. In the minimally parallel mode this can be ensured if we replace this latter rule by $d_3[_{h''}\,]_{h''} \to [_{h''}\#]_{h''}$, where h'' is one further membrane, provided in the initial configuration inside membrane h. In this way, the rule $d_3[_{h''}\,]_{h''} \to [_{h''}\#]_{h''}$ involves the new membrane h'', hence, it should be used in parallel with the rule $l_{i40}[_h\,]_h \to [_h l'_k]_h$. Of course, we also need the rules $[_{h''}\# \to \#']_{h''}$, $[_{h''}\#' \to \#]_{h''}$, instead of the corresponding rules from membrane h.

In this way, we obtain the following counterpart of Theorem 2 (note that we use two further membranes, h' and h''):

Theorem 3. *$N_{min}OP_{n_1,*}((a_1),(b_1),(c_1),(d_1),(e_1)) = NRE$, for all $n_1 \geq 7$.*

In the one-normal form we require that objects appearing on the left-hand side of a rule are different from those appearing on the right-hand side. This restriction can be reversed for rules of types (b) – (d): these rules can be of the forms $a[_h\,]_h \to [_h b]_h$, $[_h a\,]_h \to b[_h\,]_h$, $[_h a\,]_h \to b$, with $a = b$. The changes to be made in the previous proofs are as suggested in Figures 6, 7, 8, 9, which present the modules for simulating ADD and SUB instructions, in the two possible guesses,

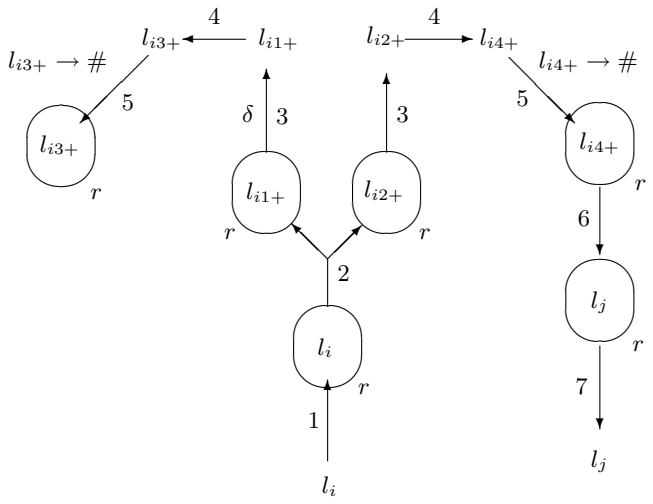

Fig. 7. The work of module SUB, when guessing that the register is non-empty, with equal objects in the rules

as well as the final module, directly for the case of the minimally parallel way of using the rules. This time we need seven steps for simulating an instruction of the register machine and additional membranes h, h', h'', h''', with several evolution steps done inside new membranes in order to change the objects (for instance, this is the case for step 6 from Figure 6, which is done by using the rule $[_{h'''} l_{i2} \to l_\alpha]_{h'''}$ for $\alpha \in \{j, k\}$. The technical details are left to the reader.

One further observation is that the previous systems can easily be modified in order to work in the accepting mode: we introduce the number to be analyzed in the form of the multiplicity of membranes with label 1 initially present in the skin membrane (for number n we start with $n+1$ membranes $[_1\,]_1$ in the system) and we work exactly as above, with the final module now having only the task of dissolving the auxiliary membrane with label h, thus halting the computation (the module can be changed, because it is no longer necessary to send out of the system objects c corresponding to the copies of membrane with label 1 present in the end of the computation, but this is not essential).

Consequently, all the previous results, for both minimally and maximally parallel use of rules, are valid also for accepting P systems in the one-normal form.

5 Efficiency Results

We now pass to proving the efficiency result mentioned in the Introduction – this time, the system is not in the one-normal form, and it remains as an interesting research topic to check whether this is possible or not.

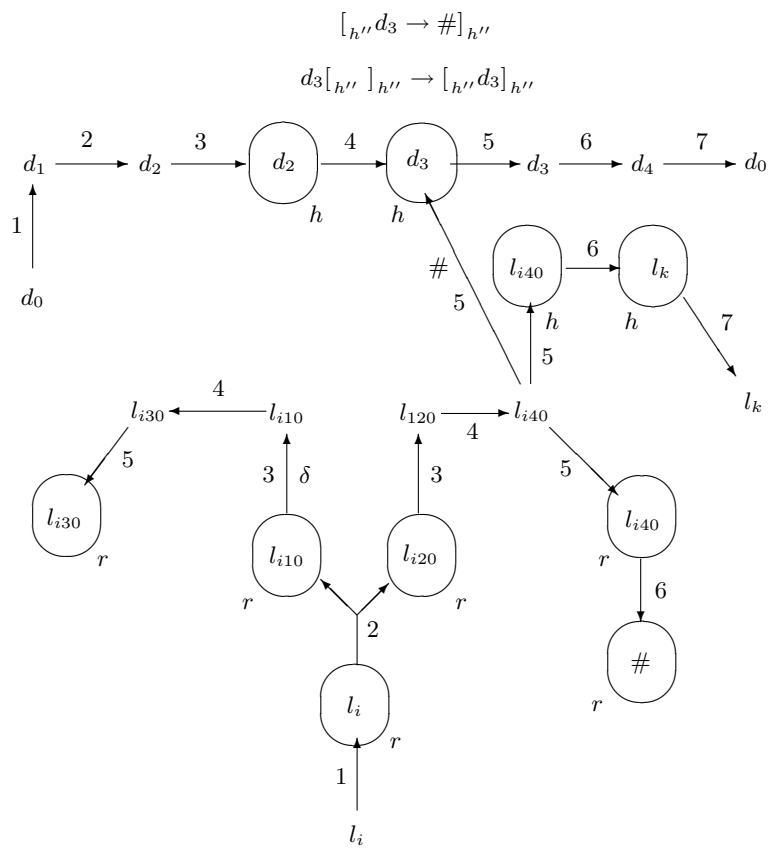

Fig. 8. The work of module SUB, when guessing that the register is empty, with equal objects in the rules

5.1 Solving SAT by Polarizationless P Systems

Below we give an efficient semi–uniform solution to the satisfiability problem by using P systems with polarizationless active membranes working in the minimally parallel mode.

Theorem 4. *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 polarizationless P system with active membranes, constructed in a semi-uniform way in linear time with respect to n and m, and working in the minimally parallel mode.*

Proof. Let us consider a propositional formula $\varphi = C_1 \wedge \ldots \wedge C_m$ such that each clause C_j, $1 \leq j \leq m$, is of the form $C_j = y_{j,1} \vee \ldots \vee y_{j,k_j}$, $k_j \geq 1$, for $y_{j,r} \in \{x_i, \neg x_i \mid 1 \leq i \leq n\}$. For each $i = 1, 2, \ldots, n$, let us denote

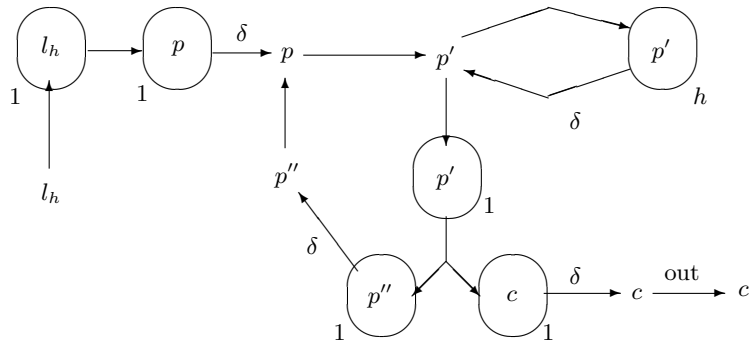

Fig. 9. The work of the final module, with equal objects in the rules

$$t(x_i) = \{c_j \mid \text{there is } r, 1 \le r \le k_j, \text{ such that } y_{j,r} = x_i\},$$
$$f(x_i) = \{c_j \mid \text{there is } r, 1 \le r \le k_j, \text{ such that } y_{j,r} = \neg x_i\}.$$

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

We construct the P system $\Pi(\varphi)$ with the following components (the output membrane is not necessary, because the result is obtained in the environment):

$$\begin{aligned}
O &= \{a_i, f_i, t_i \mid 1 \le i \le n\} \\
&\quad \cup \{c_j, d_j \mid 1 \le j \le m\} \\
&\quad \cup \{p_i \mid 1 \le i \le 2n+7\} \\
&\quad \cup \{q_i \mid 1 \le i \le 2n+1\} \\
&\quad \cup \{r_i \mid 1 \le i \le 2n+5\} \\
&\quad \cup \{b_1, b_2, y, \texttt{yes}, \texttt{no}\}, \\
H &= \{s, s', p, q, r, 0, 1, 2, \ldots, m\}, \\
\mu &= [_s[_{s'}[_p\,]_p[_0[_q\,]_q[_r]_r[_1\,]_1[_2\,]_2 \cdots [_m\,]_m]_0]_{s'}]_s, \\
w_p &= p_1,\ w_q = q_1,\ w_r = r_1,\ w_0 = a_1, \\
w_s &= w_{s'} = w_j = \lambda, \text{ for all } j = 1, 2, \ldots, m.
\end{aligned}$$

The set of evolution rules, R, consists of the following rules:

(1) $[p_i \to p_{i+1}]_p$, for all $1 \le i \le 2n+6$,
$[q_i \to q_{i+1}]_q$, for all $1 \le i \le 2n$,
$[r_i \to r_{i+1}]_r$, for all $1 \le i \le 2n+4$.
These rules are used for evolving counters p_i, q_i, and r_i in membranes with labels p, q, and r, respectively.

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

$[\, f_n \to f(x_n)]_0$,
$[\, t_n \to t(x_n)]_0$.
The goal of these rules is to generate the truth assignments of the n variables $x_1, \ldots, x_n$, and to analyse the clauses satisfied by x_i and $\neg x_i$, respectively.

(3) $c_j[\ ]_j \to [\, c_j]_j$ and $[\, c_j]_j \to d_j$, for all $1 \leq j \leq m$.
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 in order to dissolve it and to send objects d_j to membrane 0.

(4) $[\, q_{2n+1}\ ]_q \to q_{2n+1}[\ ]_q$,
$[\, q_{2n+1}\ \to b_1]_0$.
By using these rules the counter q produces an object b_1 in each membrane 0.

(5) $b_1[\ ]_j \to [\, b_1]_j$ and $[\, b_1]_j \to b_2$, for all $1 \leq j \leq m$,
$[\, b_2\ ]_0 \to b_2$.
These rules allow to detect whether the truth assignment associated with a membrane 0 assigns the value *false* to the formula (in that case, the membrane 0 will be dissolved).

(6) $[\, p_{2n+7}\ ]_p \to p_{2n+7}[\ ]_p$,
$[\, p_{2n+7}\ ]_{s'} \to \texttt{no}[\ ]_{s'}$,
$[\, \texttt{no}\ ]_s \to \texttt{no}[\ ]_s$,
$[\, r_{2n+5}\ ]_r \to r_{2n+5}$,
$[\, r_{2n+5}\ ]_0 \to y[\ ]_0$,
$[\, y\ ]_{s'} \to \texttt{yes}$,
$[\, \texttt{yes}\ ]_s \to \texttt{yes}[\ ]_s$.
These rules produce the answer of the P system.

An overview of the computations in the P system

For the sake of readability, the initial configuration is given in Figure 10.

The membranes with labels p, q, and r, with the corresponding objects p_i, q_i, and r_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 rules of type (1).

In parallel with these rules, membrane 0 evolves by means of the rules of type (2). In odd steps (from step 1 to step $2n$), we divide the (non-elementary) membrane 0 (with f_i, t_i corresponding to the truth values *false, true*, respectively, for variable x_i); in even steps we introduce the clauses satisfied by $x_i, \neg x_i$, respectively. When we divide membrane 0, all inner objects and membranes are replicated; in particular, all membranes with labels $1, 2, \ldots, m$, as well as membranes q and r, are replicated, hence, they are present in all membranes with label 0.

This process lasts $2n$ steps. At the end of this phase, all 2^n truth assignments for the n variables have been generated and they are encoded in membranes labeled by 0.

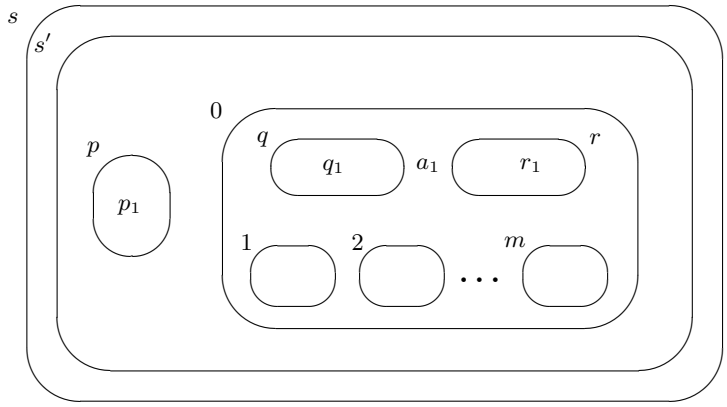

Fig. 10. The initial configuration of the system from the proof of Theorem 4

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 first rule of type (3), permitting its dissolution by means of the second rule of that type and sending objects d_j to membrane 0.

This is done also in step $2n+1$, in parallel with using the rules of type (1) and (4) for evolving membranes p, q, and r.

In step $2n+2$, the counters p_i and r_i keep evolving and the second rule of type (4) produces an object b_1 in each membrane 0.

Thus, after $2n+2$ steps, the configuration $\mathcal{C}_{2n+2}$ of the system consists of 2^n copies of membrane 0, each of them containing the membrane p empty, the membrane r with the object r_{2n+3}, possible objects c_j and d_j, $1 \leq j \leq m$, as well as copies of those membranes with labels $1, 2, \ldots, m$ corresponding to clauses which were not satisfied by the truth assignment generated in that copy of membrane 0. The membranes associated with clauses satisfied by the truth assignments generated have been dissolved by the corresponding object c_j. Moreover, in that configuration the membrane p contains the object p_{2n+3}, and membranes s' and s are empty.

Therefore, formula φ is satisfied if and only if there is a membrane 0 where all membranes $1, 2, \ldots, m$ have been dissolved. In order to check this last condition, we proceed as follows.

In step $2n+3$ we use the first rule of type (5) which introduces the object b_1 in a membrane j which has not been dissolved (this is made in a non–deterministically manner). In parallel, the counters q and r follow with evolving. The object b_1 in membrane j (step $2n+4$) dissolves that membrane producing an object b_2 in membrane 0.

In step $2n+5$ the counter r_{2n+5} exits from membrane r and, simultaneously, each membrane 0 containing an object b_2 is dissolved by the third rule of type (5).

Then, formula φ is satisfied if and only if in the configuration $\mathcal{C}_{2n+5}$ there exists a membrane 0 that has not been dissolved (and thus containing the object r_{2n+5}).

In the next step, the counter q_i evolves to q_{2n+7} in membrane q, and if there is a membrane 0 that has not been dissolved, the object r_{2n+5} sends an object y to membrane s'. On the contrary, only the counter q_i evolves.

In step $2n+7$ the counter p_{2n+7} exits from membrane p to membrane s', by applying the first rule of type (6). If the formula φ is satisfiable then an object y dissolves the membrane s' by applying the sixth rule of type (6) producing an object `yes` in the skin. In the next step, this object is sent to the environment and the P system halts. On the contrary, if membrane s' has not been dissolved, the object p_{2n+7} in membrane s' produces an object `no` in the skin, by using the second rule of type (6); in the next step an object `no` is sent to the environment and the system halts.

Therefore, if the formula is satisfiable, then the object `yes` exits the system in step $2n+8$, and, if the formula is not satisfiable, then the object `no` exits the system in step $2n+9$. In both cases, this is the last step of the computation.

The system $\Pi(\varphi)$ uses $9n+2m+18$ objects, $m+6$ initial membranes, containing in total 4 objects, and $8n+4m+21$ rules. The length of any rule is bounded by $m+3$. Clearly, all computations stop (after at most $2n+9$ steps) and all give the same answer, `yes` or `no`, to the question whether formula φ is satisfiable, hence, the system is weakly confluent. These observations conclude the proof. □

Remark 1. 1. The system used in the previous proof is not in the one-normal form, because the rules of type (b) are of the form $[_h a \to u]_h$ with u being an arbitrary multiset. Because the maximal length of such strings u is known ($m+1$), we can replace each rule of this form by m rules, each of them introducing two objects; using rules of the form $[_h a \to b]_h$, we can also synchronize the applications of rules, hence, at the price of getting a computation m times longer, we can obtain a sort of two-normal form. Of course, the rules of types (b), (c), (d) can be also arranged to have either different objects or identical objects. We leave the details as a task for the reader, together with the more interesting issue of finding a polynomial solution to `SAT` by a system in the one-normal form.

2. Let us note that we can design a *deterministic* P system $\Pi(\varphi)$ working in minimally parallel mode which decides the satisfiability of φ. To this aim, it is enough to have m copies of the object b_1 in each membrane 0 of the configuration $\mathcal{C}_{2n+2}$. For that, the rule $[q_{2n+1} \to b_1]_0$ can be replaced by $[q_{2n+1} \to b_1^m]_0$.
3. As a consequence of Theorem 4 we obtain the inclusion of $\mathbf{NP} \cup \mathbf{co}-\mathbf{NP}$ in the class of all decision problems solvable in polynomial time in a semi–uniform way by a family of P systems with polarizationless active membranes working in the minimally parallel mode (let us recall that due to the confluence of the P systems solving a decision problem, every P system that decides an instance working in minimally parallel mode, also decides it working in the maximally parallel mode, but the reciprocal is not true).

5.2 A Formal Verification

In order to assure that the system $\Pi(\varphi)$ decides the instance φ, two main properties must to be proved: (a) if *there exists an* accepting computation of the P system processing φ, answering `yes`, then the problem also answers `yes` for that instance (*soundness*), and (b) if the problem answers *yes*, then *any* computation of the P system processing that instance answers *yes* (*completeness*). Hence, the system $\Pi(\varphi)$ must satisfy a condition of *confluence*: every computation of the system has the same output.

To prove that the system $\Pi(\varphi)$ is sound and complete with respect to the **SAT** problem, it is sufficient to show that a truth assignment makes true the formula φ if and only if in the configuration $\mathcal{C}_{2n+5}$ there is at least a membrane labeled by 0 that has not been dissolved.

If σ is a truth assignment on a set of variables $\{x_1, \ldots, x_n\}$, then we write $\sigma(x_i) = f$ or $\sigma(x_i) = t$.

Lemma 2. *For each i $(1 \leq i \leq n)$ the following conditions hold true:*

1. $\mathcal{C}_{2i-1}(p) = \{p_{2i}\}$, $\mathcal{C}_{2i-1}(s) = \mathcal{C}_{2i-1}(s') = \emptyset$.
2. *For each truth assignment σ on $\{x_1, \ldots, x_i\}$ there exists a unique membrane 0 such that its contents is*

$$\{(\sigma(x_i))_i\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i-2} (\sigma(x_l))(x_l) \wedge i \geq 3\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 3\}.$$

   Moreover, it inside contains a membrane labeled by q which contains the object q_{2i}, a membrane labeled by r which contains the object r_{2i}, and empty inner membranes j, where $1 \leq j \leq m$; moreover, if $i \geq 2$, then:

   (a) $c_j \notin \bigcap_{1 \leq l \leq i-2} (\sigma(x_l))(x_l)$,

   (b) if $c_j \in (\sigma(x_{i-1}))(x_{i-1}) - \bigcap_{1 \leq l \leq i-2} (\sigma(x_l))(x_l)$, *then* $\mathcal{C}_{2i-1}(j) = \{c_j\}$.
3. $\mathcal{C}_{2i}(p) = \{p_{2i+1}\}$, $\mathcal{C}_{2i}(s) = \mathcal{C}_{2i}(s') = \emptyset$.
4. *For each truth assignment σ on $\{x_1, \ldots, x_i\}$ there exists a unique membrane 0 such that its contents is*

$$\{(\sigma(x_i))(x_i), a_{i+1}\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 2\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 3\},$$

   where $a_{n+1} = \lambda$.
   Furthermore, it contains inside a membrane labeled by q which contains the object q_{2i+1}, a membrane labeled by r which contains the object r_{2i+1}, and

empty inner membranes j, where $1 \leq j \leq m$; moreover, if $i \geq 2$, then $c_j \notin \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l)$.

Proof. By induction on n. It is easy to prove the base case. Let us suppose that the result holds for i $(1 \leq i < n)$.

From the induction hypothesis, we deduce that the configuration $\mathcal{C}_{2i+1}$ is obtained from $\mathcal{C}_{2i}$ by applying the rules $[p_{2i+1} \rightarrow p_{2i+2}]_p$, $[q_{2i+1} \rightarrow q_{2i+2}]_q$, $[r_{2i+1} \rightarrow r_{2i+2}]_r$, $[a_{i+1}]_0 \rightarrow [f_{i+1}]_0[t_{i+1}]_0$, and $c_j\,[\ ]_j \rightarrow [c_j]_j$ $(1 \leq j \leq m)$.

Hence, we have $\mathcal{C}_{2i+1}(p) = \{p_{2i+2}\}$ and $\mathcal{C}_{2i+1}(s) = \mathcal{C}_{2i+1}(s') = \emptyset$. Moreover, for each truth assignment σ on $\{x_1, \ldots, x_i, x_{i+1}\}$ there exists a unique membrane 0 which contains

$$\{(\sigma(x_{i+1}))_{i+1}\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 2\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i} (\sigma(x_l))(x_l) \wedge i \geq 3\}.$$

This membrane 0 inside contains a membrane labeled by q which contains the object q_{2i+2}, a membrane labeled by r which contains the object r_{2i+2}, and inner membranes j such that:

- if $i \geq 2$ and $c_j \in \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l)$, then membrane j has been dissolved,
- if $i \geq 2$ and $c_j \in (\sigma(x_i))(x_i) - \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l)$, then $\mathcal{C}_{2i+1}(j) = \{c_j\}$;
- the remaining membranes are empty.

The configuration $\mathcal{C}_{2i+2}$ is obtained from $\mathcal{C}_{2i+1}$ by applying the rules $[p_{2i+2} \rightarrow p_{2i+3}]_p$, $[q_{2i+2} \rightarrow q_{2i+3}]_q$; $[r_{2i+2} \rightarrow r_{2i+3}]_r$, $[f_{i+1} \rightarrow f(x_{i+1}a_{i+1}]_0$, $[t_{i+1} \rightarrow t(x_{i+1})a_{i+1}]_0$, where $a_{n+1} = \lambda$, and $c_j\,[\ ]_j \rightarrow [c_j]_j$ $(1 \leq j \leq m)$.

Hence, we have $\mathcal{C}_{2i+2}(p) = \{p_{2i+3}\}$ and $\mathcal{C}_{2i+2}(s) = \mathcal{C}_{2i+2}(s') = \emptyset$. Moreover, for each truth assignment σ on $\{x_1, \ldots, x_i, x_{i+1}\}$ there exists a unique membrane 0 which contains

$$\{(\sigma(x_{i+1}))(x_{i+1})\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i} (\sigma(x_l))(x_l) \wedge i \geq 2\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i} (\sigma(x_l))(x_l) \wedge i \geq 3\}$$

That membrane 0 inside contains a membrane labeled by q which contains the object q_{2i+3}, a membrane labeled by r which contains the object r_{2i+3}, and empty inner membranes j, where $1 \leq j \leq m$; if $i \geq 2$, then $c_j \notin \bigcap_{1 \leq l \leq i} (\sigma(x_l))(x_l)$. □

Lemma 3. *The configuration $\mathcal{C}_{2n+5}$ fulfills the following conditions:*

1. $\mathcal{C}_{2n+5}(p) = \{p_{2n+6}\}$, $\mathcal{C}_{2n+5}(s) = \emptyset$, *and the content of* $\mathcal{C}_{2n+5}(s')$ *is*

$$\{b_2^u, r_{2n+5}^u\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 3\},$$

   where u *is the number of truth assignment* σ *such that* $\sigma(\varphi) = 0$.
2. *For each assignment* σ *which satisfies the formula* φ *there exists a unique membrane 0 whose contents is*

$$\{r_{2n+5}\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge n \geq 3\}.$$

Corollary 1. *The formula* φ *is satisfiable if and only if in the configuration* $\mathcal{C}_{2n+5}$ *there is at least a membrane labeled by 0 that has not been dissolved.*

Proof. Indeed, a truth assignment σ is a satisfying assignment if and only if in the configuration $\mathcal{C}_{2n+4}$ the object b_1 from the membrane 0 associated with the assignment σ has not entered any membrane j $(1 \leq j \leq m)$, because all such membranes have been dissolved in previous steps. But this condition is equivalent to the existence of some membrane labeled by 0 in the configuration $\mathcal{C}_{2n+5}$. □

Corollary 2. *Let* u *be the number of truth assignments* σ *such that* $\sigma(\varphi) = 0$, *and let* $\mathcal{C}$ *be a computation of* $\Pi(\varphi)$.

1. *If the formula* φ *is satisfiable, then* $\mathcal{C}_{2n+8}(p) = \emptyset$, *the contents of* $\mathcal{C}_{2n+8}(s)$ *is*

$$\{b_2^u, r_{2n+5}^u, y^{n-u-1}, p_{2n+7}\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 3\},$$

   $\mathcal{C}_{2n+8}(env) =$ `yes`, *and the system halts giving an affirmative answer.*
2. *If the formula* φ *is not satisfiable, then* $\mathcal{C}_{2n+8}(p) = \emptyset$, *the contents of* $\mathcal{C}_{2n+8}(s')$ *is*

$$\{b_2^u, r_{2n+5}^u, y^{n-u}\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 3\},$$

   $\mathcal{C}_{2n+8}(s) = \{\texttt{no}\}$, $\mathcal{C}_{2n+9}(p) = \emptyset$, $\mathcal{C}_{2n+9}(s) = \emptyset$, $\mathcal{C}_{2n+9}(env) =$ `no`, *and the system halts giving a negative answer.*

6 Final Remarks

We have shown that P systems with polarizationless active membranes are computationally complete, even when working in the minimally parallel mode with very restricted forms of the rules, i.e., only evolving one object in or through a membrane (one-normal form). Moreover, we have also shown that `SAT` can be solved by P systems with polarizationless active membranes even when working in the minimally parallel mode, but in this case the question whether we can restrict the rules to the one-normal form remains as an open problem.

Acknowledgements

The work of Gh. Păun was partially supported by Project BioMAT 2-CEx06-11-97/19.09.06. The work of M.J. Pérez-Jiménez was supported by the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, co–financed by FEDER funds, and of the project of Excellence TIC 581 of the Junta de Andalucía.

References

1. A. Alhazov: P systems without multiplicities of symbol-objects. *Information Processing Letters*, 100 (2006), 124–129.
2. A. Alhazov, M.J. Pérez-Jiménez: Uniform solution to QSAT using polarizationless active membranes, Vol. I of Proc. of the Fourth Brainstorming Week on Membrane Computing, Sevilla (Spain), Jan 30 - Feb 3, 2006, 29–40.
3. A. Alhazov, R. Freund, On efficiency of P systems with active membranes and two polarizations. In [7], 81–94.
4. A. Alhazov, R. Freund, Gh. Păun: Computational completeness of P systems with active membranes and two polarizations. In *Machines, Computations, and Universality. 4th Intern. Conf. Sankt Petersburg, Russia, 2004*, LNCS 3354 (M. Margenstern, ed.), Springer, 2005, 82–92.
5. G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism. *Theoretical Computer Sci.*, to appear.
6. J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.
7. G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing, International Workshop, WMC5, Milano, Italy, 2004, Selected Papers*. Lecture Notes in Computer Science 3365, Springer-Verlag, Berlin, 2005.
8. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
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. Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.: *Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, February 2004*. Technical Report 01/04 of Research Group on Natural Computing, Sevilla University, Spain, 2004.
13. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: *Teoria de la complejidad en modelos de computacion celular con membranas*. Kronos Editorial, Sevilla, 2002.