Theoretical Computer Science 429 (2012) 74–86

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Asynchronous P systems with active membranes

Pierluigi Frisco [a,*], Gordon Govan [a], Alberto Leporati [b]

[a] School of Math. and Comp. Sciences, Heriot–Watt University, EH14 4AS Edinburgh, UK

[b] Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano – Bicocca, Viale Sarca 336/14, 20126 Milano, Italy

A B S T R A C T

We prove that recognising P systems with active membranes operating in *asynchronous* mode are able to solve in a semi-uniform way both **NP-complete** and **PP-complete** problems in linear time (in the best case) and exponential space, when using different sets of rules. Precisely, the proposed solution of k-SAT, $k \geq 3$, uses evolution and communication rules, as well as membranes creation and division of non-elementary membranes; however, it does not use neither polarisations nor membrane dissolution rules. Our solution of MAJORITY-SAT makes use of polarisations as well as evolution and communication rules, together with rules for dividing non-elementary membranes.

We also prove that these systems can simulate partially blind register machines; the converse simulation holds for a constrained version of our P systems.

1. Introduction

Membrane systems (*P systems*) are an abstract model of computation inspired by the compartmentalisation present in eukaryotic cells [10,11,5,12]. P systems can be used as a parallel model of computation, utilising the inherent parallelism that is present in biological cells.

P systems with active membranes are able to compute by using a set of rules that describe the movement and evolution of symbols and compartments. Symbols can move in and out of compartments and evolve into other symbols. Compartments can be divided into more compartments, have their polarity (the electrical charge of the membrane) changed, etc.

Due to their parallelism, P systems are able to solve **NP-complete** problems in linear time, but to do so they have to create an exponentially large workspace by membrane division rules, and try all possible solutions to the problem at the same time. In fact, when membrane division is not allowed then the resulting P systems can be shown to be no more powerful than polynomial-time Turing machines [18].

A P system can run in different operating modes: among the others, maximally parallel, minimally parallel, and with maximal strategy. In this paper, we study P systems operating in the *asynchronous* mode. This means that in each transition *at least* one rule is applied and no upper bound is put to the number of times a given rule can be applied in a transition step. This operation mode is thus more general than maximal parallelism and other operating modes considered in the literature. We try to clarify this concept with an example. Let us assume that in a certain configuration a P system operating under maximal parallelism applies two times rule 1 and one time rule 2. We denote this rule application by $(1, 1, 2)$. If the P system run in asynchronous mode, then in the same configuration different multisets of rules could be applied: (1), $(1, 1)$, (2), $(1, 2)$ and $(1, 1, 2)$. Clearly, the asynchronous operation mode is more general than maximal parallelism.

When working in the asynchronous mode, in each transition a P system may apply any number of applicable rules at the time; thus for each computation we have a *lower bound* and an *upper bound* to the execution time. We will say that they are the *best* and the *worst* execution time, respectively.

* Corresponding author.
E-mail addresses: P.Frisco@hw.ac.uk (P. Frisco), gmg8@hw.ac.uk (G. Govan), alberto.leporati@unimib.it (A. Leporati).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.12.026

So we prove that recognising P systems with active membranes operating in asynchronous mode are able to solve in a semi-uniform way both **NP-complete** and **PP-complete** problems in linear time (in the best case) and exponential space. The selected **NP**-complete and **PP**-complete problems are k-SAT (with $k \geq 3$) and MAJORITY-SAT, respectively. We solve them by using different kinds of rules. Precisely, for solving k-SAT we use evolution and communication rules, as well as membranes creation and division of non-elementary membranes; however, we do not use neither polarisations nor membrane dissolution rules. On the other hand, our solution of MAJORITY-SAT makes use of polarisations as well as evolution and communication rules, together with rules for dividing non-elementary membranes.

In the last part of the paper we also show that P systems with active membranes operating in the asynchronous mode are able to simulate partially blind register machines. To this aim we have either the possibility to use polarities—and in such a case only evolution and communication rules are needed—or not to use them. In the latter case, however, we also need rules for creating and dissolving membranes. We are able to show also the converse simulation, but only for a constrained version of our P systems. The equivalence between the two models of computation, or proving that P systems are more powerful than partially blind register machines, remains an open problem.

The paper is structured as follows. In the next section we give some definitions relevant to this paper, while in Section 3 we discuss different operational modes. In Section 4 we show how **NP**-complete and **PP**-complete problems can be solved by the considered P systems, and in Section 5 we explore the relations occurring among our P systems and partially blind register machines. We end in Section 6 with some comments and directions for future research.

2. Basic definitions

We assume the reader to be familiar with Theoretical Computer Science [17], in particular with Membrane Computing [5,12] and Computational Complexity Theory [9]. Here we only introduce concepts strictly relevant to what is to be presented.

Definition 1. A *P system with active membranes* is a construct

$$\Pi = (V, \mu, L_1, \ldots, L_m, R)$$

where:

V is an alphabet;
$\mu = (W, E, pol)$ is a vertex-labelled cell-tree (where a cell-tree is a rooted tree such that the root has only one child) underlying Π, where:
$W = \{0, 1, \ldots, m\}$ contains *vertices*. Each vertex in W defines a compartment of Π. The root of the cell-tree is vertex 0 and its child is vertex 1;
$E \subseteq W \times W$ defines the *edges* among the vertices of the cell-tree;
$pol : W \rightarrow \{+, -, 0\}$ is a labelling function, called *polarity function*, associating elements of $W_P = \{+, -, 0\}$ to the vertices;
In the following, μ is depicted using the bracket representation common to P systems;
$L_i : V \rightarrow \mathbb{N}$, $i \in W \setminus \{0\}$ are multisets of symbols associated with the vertices of W;
R is a finite set of rules of the form:

(a) $[_i\ v \rightarrow w\ ]_i^p$
with $i \in W,\ p \in W_P,\ v \in V$ and $w : V \rightarrow \mathbb{N}$.
Informally: productions associated with compartments;

(b) $v\ [_i\ ]_i^p \rightarrow [_i\ w\ ]_i^{p'}$
with $i \in W,\ p, p' \in W_P,\ v \in V$ and $w \in V \cup \{\epsilon\}$.
Informally: a symbol can pass into a compartment;

(c) $[_i\ v\ ]_i^p \rightarrow [_i\ ]_i^{p'}\ w$
with $i \in W,\ p, p' \in W_P,\ v \in V$ and $w \in V \cup \{\epsilon\}$.
Informally: a symbol can pass outside a compartment;

(d) $[_i\ v\ ]_i^p \rightarrow w$
with $i \in W \setminus \{0, 1\},\ p \in W_P,\ v \in V$ and $w \in V \cup \{\epsilon\}$.
Short name: *dissolution*.
Informally: the presence of a symbol in a compartment lets the membrane defining the compartment disappear;

(e) $[_i\ v\ ]_i^p \rightarrow [_{i'}\ w\ ]_{i'}^{p'} [_{i'}\ z\ ]_{i'}^{p''}$
with $i, i' \in W \setminus \{0, 1\},\ p, p', p'' \in W_P,\ v \in V$ and $w, z \in V \cup \{\epsilon\}$.
Short name: *division of a (simple or composite) compartment*.
Informally: the presence of a symbol in a compartment lets the compartment duplicate;

(g) $[_i\ v\ ]_i^p \rightarrow [_i [_{i'}\ w\ ]_{i'}^{p'}]_i^p$
with $i, i' \in W \setminus \{0, 1\},\ p, p' \in W_P, v \in V$ and $w \in V \cup \{\epsilon\}$.
Short name: *creation of a compartment*.
Informally: the presence of a symbol in a compartment lets a new compartment to be created.

Vertex 0 identifies the environment, while vertex 1 the skin compartment of μ. The set $W_P = \{+, -, 0\}$ is called *set of polarities* and its elements are called *polarities*. Specifically, $+$ is called *positive*, $-$ is called *negative* and 0 is called *neutral* polarity. A compartment having a positive polarity is said to be a *positive compartment*; similarly for the remaining polarities. Rules of type (a) are also called *evolution* rules, whereas rules of types (b) and (c) are commonly called *communication* rules.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. A computation step consists in the application of one or more rules, according to the selected mode of operation; it changes the current configuration and produces another configuration.

The *initial configuration* of Π consists of the membrane structure μ and the multisets of symbols $L_1, \ldots, L_m$, each associated to the corresponding vertex of W. A *halting computation* of Π is a finite sequence of configurations $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable by $\mathcal{C}_i$ via a single computation step, and no rules can be applied anymore in $\mathcal{C}_k$. A *non-halting* computation $\vec{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted. For further details on the definition of these P systems, their working and their modes of operation, the reader is referred to [5,12].

P systems with active membranes can be used as *recognising* devices by employing two distinguished symbols $\texttt{yes}, \texttt{no} \in V$; exactly one of these must be sent out from the outermost membrane during the last step of each computation, in order to signal *acceptance* or *rejection*, respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not the case, then P systems are said to be *non-confluent*, and the overall result is established as for nondeterministic Turing machines: it is acceptance if and only if an accepting computation exists.

Some of the systems considered in the following do not send the symbol $\texttt{no}$ out to the environment in rejecting computations. Instead, these systems simply halt without letting any symbol pass to the environment. Even if this behaviour deviates from the above definition—which is the one usually found in the P systems literature—we will see that it does not change the results we are going to state.

In order to solve decision problems (i.e., decide languages), we use *families* of recognising P systems $\Pi = \{\Pi_x : x \in V^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq V^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ is restricted, in order to be computable efficiently; usually one of the following *uniformity conditions* is imposed.

Definition 2. A family of P systems $\Pi = \{\Pi_x : x \in V^\star\}$ is said to be *semi-uniform* if the mapping $x \mapsto \Pi_x$ can be computed in polynomial time by a deterministic Turing machine.

Any explicit encoding of Π_x is allowed as output, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space in proportion to their number. For instance, the membrane structure can be represented by brackets, and the multisets as strings (i.e., in unary notation); this is a *permissible encoding* in the sense of [8].

Definition 3. A family of P systems $\Pi = \{\Pi_x : x \in V^\star\}$ is said to be *(polynomial-time) uniform* if the mapping $x \mapsto \Pi_x$ can be computed by two deterministic polynomial-time Turing machines F (for "family") and E (for "encoding") as follows:

- The machine F, taking as input *the length n of x* in unary notation, constructs a P system Π_n with a distinguished input membrane (the P systems structure Π_n is common for all inputs of length n).
- The machine E, on input x, outputs a multiset w_x (an encoding of the specific input x).
- Finally, Π_x is simply Π_n with w_x added to the multiset placed inside its input membrane.

Notice how the uniform construction is just a restricted case of semi-uniform construction. The relations between the two kinds of uniformity have not completely been clarified yet; see [12,8] for further details on uniformity conditions, including even weaker constructions.

Notice also that the above definition of uniformity is possibly weaker than the one commonly used in membrane computing [13], where the Turing machine F maps each input x to a P system $\Pi_{s(x)}$, where $s\colon V^\star \to \mathbb{N}$ is a measure of the size of the input (in our case, $s(x)$ is always $|x|$). In particular, complexity classes defined using this restricted uniformity condition are not always formally known to be closed under polynomial-time reductions. This might complicate proofs of inclusions among complexity classes, although one can usually find a proof not relying on closure under polynomial-time reductions, as in [16]. See [8] for further details on uniformity conditions, including constructions using weaker devices than polynomial-time Turing machines.

Usually, time complexity for families of recognising P systems is measured as follows.

Definition 4. A uniform or semi-uniform family of P systems $\Pi = \{\Pi_x : x \in V^\star\}$ is said to decide the language $L \subseteq V^\star$ (in symbols $L(\Pi) = L$) in time $f\colon \mathbb{N} \to \mathbb{N}$ if and only if, for each $x \in V^\star$,

1. the system Π_x accepts if $x \in L$, and rejects if $x \notin L$;
2. each computation of Π_x halts within $f(|x|)$ computation steps.

However, since we will consider recognising P systems working in the asynchronous mode, the number of computation steps will depend upon how the system Π_x selects the rules to be applied at each transition step. Hence, we will distinguish among the *best case* (lower bound) and the *worst case* (upper bound) time complexity. The above definition corresponds to the worst case, since *each* computation of Π_x must halt within $f(|x|)$ computation steps. For the best case, we give instead the following definition.

Definition 5 (*Best Case Time Complexity*). A uniform or semi-uniform family of P systems $\Pi = \{\Pi_x : x \in V^\star\}$ is said to decide the language $L \subseteq V^\star$ (in symbols $L(\Pi) = L$) in (best case) time $f: \mathbb{N} \to \mathbb{N}$ if and only if, for each $x \in V^\star$,

1. the system Π_x accepts if $x \in L$, and rejects if $x \notin L$;
2. at least one computation of Π_x halts within $f(|x|)$ computation steps.

As usually done in the literature, we denote by $\mathbf{PMC}_{\mathcal{AM}}$ (resp., $\mathbf{PMC}^*_{\mathcal{AM}}$) the class of languages (equivalently, decision problems) which can be decided in worst case polynomial time by uniform (resp., semi-uniform) families of P systems with active membranes. Similarly, we denote by $b\mathbf{PMC}_{\mathcal{AM}}$ (resp., $b\mathbf{PMC}^*_{\mathcal{AM}}$) the best case analogues of these classes.

Concerning *space* complexity, for the purposes of this paper we just define the size $|\mathcal{C}|$ of a configuration $\mathcal{C}$ of a P system as the sum of the number of objects and the number of membranes contained in $\mathcal{C}$; this definition assumes that every component of the system requires some fixed amount of physical space, thus approximating (up to a polynomial) the size of a real cell. The space required by a halting computation $\vec{\mathcal{C}} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$ is then given by $|\vec{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \dots, |\mathcal{C}_k|\}$ and the space required by a P system Π is $|\Pi| = \max\{|\vec{\mathcal{C}}| : \vec{\mathcal{C}}$ is a computation of $\Pi\}$. Finally, we say that a family of P systems $\Pi = \{\Pi_x : x \in V^\star\}$ operates in space $f: \mathbb{N} \to \mathbb{N}$ if $|\Pi_x| \le f(|x|)$ for all $x \in V^*$.

In this paper we use semi-uniform families of recognising P systems to solve two variants of the SAT problem [9]: k-SAT and MAJORITY-SAT.

Let $\{v_1, \dots, v_n\}$ be a set of Boolean variables. An *assignment* is a function $a: \{v_1, \dots, v_n\} \to \{\text{FALSE}, \text{TRUE}\}$ that assigns one of the Boolean values $\{\text{FALSE}, \text{TRUE}\}$ to each variable v_i. Clearly, there are 2^n possible assignments to $\{v_1, \dots, v_n\}$. We say that a given assignment *a satisfies* a Boolean formula if the formula is TRUE when assigning the values of a to $\{v_1, \dots, v_n\}$ and evaluating it. A *literal* is either a variable v_i or its negation $\bar{v}_i$. A *clause* is a disjunction of literals. A *k-clause* is a disjunction of exactly k literals. A Boolean formula ψ, built over the Boolean variables $\{v_1, \dots, v_n\}$, is said to be in *conjunctive normal form* (CNF) if ψ is a conjunction of m clauses, with $m \ge 1$. Similarly, ψ is said to be in k-CNF if every clause of ψ contains exactly k literals. In what follows we always assume that no variable appears two or more times in the same clause.

Definition 6. The SAT problem is: given a Boolean formula ψ in CNF, having m clauses over n variables, is there an assignment to the variables such that ψ is satisfied (that is, is *true*)?

The k-SAT problem is defined in the same way, except that the Boolean formula ψ is given in k-CNF.

It is well known [9] that both SAT and k-SAT, for every integer $k \ge 3$, are **NP-complete** decision problems.

The complexity class **PP** arises when one considers the number of satisfying assignments of Boolean formulae. It is defined as follows.

Definition 7. The complexity class **PP** consists of all decision problems solvable in polynomial time by nondeterministic Turing machines having the following acceptance criterion: the machine accepts its input if and only if more than half the computations are accepting.

It is known that $\mathbf{NP} \subseteq \mathbf{PP}$ [9]. The following is the standard **PP-complete** problem [9, p. 256].

Definition 8. The MAJORITY-SAT problem is: given a Boolean formula ψ in CNF, having m clauses over n variables, do more than half the assignments (i.e., more than 2^{n-1} assignments) satisfy it?

Of course it is also possible to define the MAJORITY-k-SAT problem, by requiring that the Boolean formula ψ given as input to MAJORITY-SAT is in k-CNF. However, it is not currently known whether also MAJORITY-k-SAT, even for $k \ge 3$, is **PP**-complete.

Another useful **PP**-complete problem is the following [15].

Definition 9. The SQRT-3SAT problem is: given a Boolean formula ψ in 3-CNF, having m clauses over n variables, do at least $\sqrt{2^n}$ among the 2^n possible truth assignments satisfy it?

3. Operational modes in P systems

As stated above P systems can operate in different ways, where with *operate* we mean the way rules are applied. The operational mode of a P system is detached from the definition of the system itself. This means that giving a P system, it can operate in different ways. It is known [5,2] that the operational mode is quite influential in P systems. For instance, a specific P system can have different computational power depending on its operational mode.

Here we list and briefly describe some operational modes relevant for the following sections:

asynchronous: in each transition, if possible, at least one rule is applied. It is allowed in any transition and for any rule in a compartment to be applied more than once (as in maximal parallelism);

minimal parallelism: in each transition and for each membrane, if possible, at least one rule is applied. So, this operational mode has to have at least one rule applied for a membrane where one or more rules can be applied. If in a configuration there are different sets of rules that can be applied, then one of them is non-deterministically chosen.

maximal strategy: in each transition any rule that can be applied is actually applied at most once. If in a configuration there are different sets of rules that can be applied, then one of them is non-deterministically chosen. It is important to notice that this operational mode does not aim to maximise the number of applied rules but it aims to maximise the number of different applied rules;

maximal parallelism: this is the classical (most used) operational mode in P systems. It is similar to maximal strategy, but in each transition any rule can be applied more than once. So, in this operational mode it is possible to have transitions with more than one applicable set of rules where some of these sets have the same rules but applied a different number of times.

It is important to notice that in P systems two equal rules present in different compartments are regarded as two different rules. This is independent from the name on the membrane defining the compartments. This fact is particularly important in P systems with active membranes because (by the application of rules of type e) a membrane can be duplicated into two membranes with the same name. So, when such a system computes, the application of the same rule in different compartments defined by membranes with the same name is regarded as the application of two different rules.

Definition 10. Let X and Y be two different operational modes and Σ be a formal system. We say that X is *more general* than Y (or that Y is *more restrictive* than X) in Σ if:

when Σ operates in X mode it can compute all what can be computed by Σ when operating in Y mode;
when Σ operates in Y mode it cannot compute all what can be computed by Σ when operating in X mode.

It is important to notice that the above definition applies to specific formal systems, not to classes of formal systems. This is because some specific systems have the same behaviour independently of the operational mode. The simplest example is a system with only one rule which can be applied only once in the initial configuration, and when this happens the system halts. Clearly, the behaviour of this system is independent from the operation mode: in any way the only thing that can happen is the single application of the unique rule. Of course this fact is not supposed to hold for the family that specific system belongs to.

When a result for a specific formal system is stated, then it should consider the most general operational mode for which that result is valid. Not doing so could lead to confusion. For instance, let us assume that one defines a P system Π operating under maximal parallelism such that Π generates a family of languages L. If in this specific system Π it can never happen that in a transition a rule is applied more than once, then, as a matter of fact, the system Π operates under maximal strategy. It is indeed true that Π generates L when operating under maximal parallelism, but it is more proper to say that Π generates L when operating under maximal strategy.

In this respect we think it should be a good practice to explicitly mention in proofs where the restrictions dictated by the considered operation mode take place and how the absence of these restrictions (that is, the use of a more general operational mode) would invalidate the proved statement.

It should be clear that the asynchronous operational mode is the most general of the operational modes listed above. It should also be clear that systems operating in an asynchronous way have lower and upper bounds for the time needed to compute. The upper bound (also called the *worst case*) is reached when in each transition only one (any) instruction is applied, while the lower bound (also *best case*) is reached when in each transition the maximum number of applicable instructions is applied (that is, the system operates under maximal parallelism). In the following, when we indicate the time complexity of systems operating in an asynchronous way, we refer to the lower bound. The upper bound, discussed for each of the following results, can be exponential.

A natural question is: *Why one should be interested in studying systems whose time complexity can be exponential?* Because these systems give the challenge of studying something operating in rather minimalistic terms which make them time-independent, whose results can be valid also for other operational modes. With 'time-independent' we mean that systems operating in asynchronous mode do not rely on something that could define the passed time. Several proofs in Membrane Computing use symbols to trace the number of transitions the system went through. This is achieved, for instance, with a symbol whose subscript is incremented at each computation step. If something specific does not happen when the subscript reaches a certain value, then, for instance, the system never stops. This allows these systems to detect if something happened at a certain transition. Similarly, it has been proved [5] that the lack of a global clock definitely decreases the computational power of some abstract computational devices, P systems included. This provides the challenge in studying asynchronous systems.

4. Solving NP-complete and PP-complete problems

In this section we show how some families of P systems with active membranes can solve **NP-complete** and **PP-complete** problems. As indicated in the following, some of the results in this section improve some results present in the literature of Membrane Computing. The improvement is due to the use of asynchronous operational mode, a mode of operation which is more general than the one customarily used to prove similar results.

Using the terminology common to Membrane Computing [14], the results present in this section are about P systems constructed in a *semi-uniform* way (a P system is built for every instance of the problem) and working in a *weakly confluent* manner (the systems are non-deterministic in their computations, all systems halt and give the same answer independently from the non-deterministic computation).

Theorem 1. *The k-SAT problem, for any $k \geq 3$, can be solved by asynchronous recognising P systems with active membranes, without polarities, and using only rules of the kind* (a), (b), (c), (e) *and* (g). *The P system requires exponential space and, in the best case, linear time.*

Proof. Let $\psi = C_1 \wedge \cdots \wedge C_m$ be an instance of k-SAT, for a fixed $k \geq 3$, where each clause $C_j = l_{j_1} \vee \cdots \vee l_{k_j}$ is built using the variables $\{v_1, \ldots, v_n\}$. Moreover, let *true* and *false* be two functions, both from $\{v_1, \ldots, v_n\}$ to $\mathscr{P}\{C_1, \ldots, C_m\}$ (the power set of $\{C_1, \ldots, C_m\}$), such that:

$$true(v) = \{C_j \mid \exists r, 1 \leq r \leq k_j \text{ such that } l_{j_r} = v\}$$
$$false(v) = \{C_j \mid \exists r, 1 \leq r \leq k_j \text{ such that } l_{j_r} = \bar{v}\}$$

for $v \in \{v_1, \ldots, v_n\}$. So, these functions return the set of clauses verified and falsified when the Boolean value TRUE is assigned to v, respectively.

We define a recognising P system with active membranes $\Pi = (V, \mu, L_1, L_2, R)$ as follows:

$$V = \{v_i, F_i, T_i \mid 1 \leq i \leq n\} \cup \{c_j, d_j \mid 1 \leq j \leq m\} \cup \{d_{m+1}, \mathtt{yes}\};$$
$$\mu = (\{0, 1, 2\}, \{(0, 1), (1, 2)\});$$
$$L_1 = \emptyset;$$
$$L_2 = \{v_1\}.$$

The bracket representation of the initial underlying compartment structure of Π is

$$[_0\ [_1\ [_2\ v_1\ ]_2\ ]_1\ ]_0.$$

The set R consists of the following rules:

$$\begin{aligned}
1 &: [_2\ v_i\ ]_2 \rightarrow [_2\ F_i\ ]_2\ [_2\ T_i\ ]_2 \text{ for } 1 \leq i \leq n,\\
2 &: [_2\ F_i \rightarrow false(v_i)v_{i+1}\ ]_2 \text{ for } 1 \leq i \leq n-1,\\
3 &: [_2\ T_i \rightarrow true(v_i)v_{i+1}\ ]_2 \text{ for } 1 \leq i \leq n-1,\\
4 &: [_2\ F_n \rightarrow false(v_n)d_1\ ]_2\\
5 &: [_2\ T_n \rightarrow true(v_n)d_1\ ]_2\\
6 &: [_2\ c_i\ ]_2 \rightarrow [_2\ [_{i+2}\ ]_{i+2}]_2 \text{ for } 1 \leq i \leq m,\\
7 &: d_i[_{i+2}\ ]_{i+2} \rightarrow [_{i+2}\ d_i\ ]_{i+2} \text{ for } 1 \leq i \leq m,\\
8 &: [_{i+2}\ d_i\ ]_{i+2} \rightarrow [_{i+2}\ ]_{i+2}\ d_{i+1} \text{ for } 1 \leq i \leq m,\\
9 &: [_2\ d_{m+1}\ ]_2 \rightarrow [_2\ ]_2\ \mathtt{yes}\\
10 &: [_1\ \mathtt{yes}\ ]_1 \rightarrow [_1\ ]_1\ \mathtt{yes}.
\end{aligned}$$

We have numbered the (groups of) rules in order to facilitate the explanation.

In compartment 2 the truth assignments for the n variables are generated by the rules $1, \ldots, 5$. Compartment 2 alternates between its division and the introduction of symbols. During division a value (T for TRUE and F for FALSE) is associated with variable v_i; the introduced symbols are the ones associated (by the functions *false* and *true*) with the clause satisfied by v_i and $\bar{v}_i$. After $2n$ transitions all the 2^n truth assignments for ψ are generated and present in compartments 2 (called *2-named compartment*, in the following). It is important to notice that the application of rule 1 sees the duplication of elementary compartments, that is, compartments with no other compartments inside.

Every time a symbol c_i, $1 \leq i \leq n$, is generated, the application of rule 6 creates a compartment with name $i+2$. We remind the reader that, as the system operates in an asynchronous way, the presence of a c_i in a compartment does not imply the immediate application of rule 6. Moreover, as in a 2-named compartment there can be several copies of c_i, then several compartments with name $i+2$ can be created.

When rules 4 and 5 are applied in a 2-named compartment, then the introduction of d_1 primes the counting of the compartments created by rule 6 in that 2-named compartment. This counting takes place by the application of rules 7 and 8. It should be clear that the d_i symbol cannot visit the same $(i+2)$-named compartment, $1 \leq i \leq n$, more than once. This is because once such a compartment has been visited by d_i (rule 7), then the application of rule 8 increases the subscript of d to

$i+1$. It should also be clear that if an $(i+2)$-named compartment is not present (because, for instance, the truth assignment associated to that 2-named compartment does not satisfy clause i) or is not yet present in a 2-named compartment (because that particular rule 6 has not been applied, yet), then d_{i+1} cannot be generated in that 2-named compartment.

If the truth assignment associated to a 2-named compartment satisfies ψ, then, sooner or later, d_{m+1} will be present in this 2-named compartment. When this happens, first rule 9 and then rule 10 are applied so that the system lets yes pass to the environment. If no assignment satisfied ψ, then yes will never pass to the environment.

Clearly, Π uses exponential space. If Π operates either under maximal strategy, maximal parallelism, or minimal parallelism, then it requires linear time with respect to mn. In this system maximal parallelism, maximal strategy and minimal parallelism coincide as in each transition and in each compartment a rule is applied at most once. The upper bound for time complexity is exponential. This is reached when in each transition just one rule is applied. □

In [15,7] it is proved that recognising P systems with active membranes operating under maximal parallelism with polarities, using only rules of the kind (a), (b), (c) and (e) can be used to solve in polynomial time the **PP**-complete problem SQRT-3SAT. The solution is *uniform*, and rules of type (e) operate only on elementary membranes.

Since SQRT-3SAT is **PP**-complete, the same kind of recognising P systems could in principle be used to solve any decision problem in **PP**. However, as stated above, when using our restricted definition of uniformity the closure of the class $\mathbf{PMC}_{\mathcal{AM}}$ cannot be given for free. For this reason in [16] the same authors considered another problem, named THRESHOLD-3SAT. Such a problem asks, given an n-variable Boolean formula ψ in 3-CNF and a non-negative integer $k < 2^n$, whether more than k assignments (out of 2^n) satisfy ψ. The problem is proved to be **PP**-hard (it is still open whether it is **PP**-complete), and a uniform family of recognising P systems with active membranes is presented to solve it. The rules used for solving THRESHOLD-3SAT are the same as in [15], that is, (a), (b), (c) and (e), over elementary membranes. Further, an explicit reduction from any decision problem in **PP** to THRESHOLD-3SAT is shown, that allows to safely conclude that the whole class **PP** is included into $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$, the subclass of $\mathbf{PMC}_{\mathcal{AM}}$ obtained by avoiding dissolution and non-elementary division rules.

In [6] it is proved that P systems using the same rules, but operating under minimal parallelism, can solve MAJORITY-SAT, which is also **PP**-complete. The following theorem improves these results, as we define a similar system to solve MAJORITY-SAT operating asynchronously.

Theorem 2. *MAJORITY-SAT can be solved by asynchronous recognising P systems with active membranes with polarities, and using only rules of the kind* (a), (b), (c), *and* (e). *The P system requires exponential space and, in the best case, linear time.*

Proof. Let ψ be an instance of MAJORITY-SAT, that is, a Boolean formula in CNF having m clauses built on the variables $\{v_1, \dots, v_n\}$. Moreover, let *true* and *false* be the functions defined as in Theorem 1.

We define a recognising P system with active membranes $\Pi = (V, \mu, L_1, \dots, L_{m+n+2}, R)$ as follows:

$$\begin{aligned}
V &= \{v_i, F_i, T_i \mid 1 \le i \le n\} \cup \{c_j, d_j \mid 1 \le j \le m\} \cup \\
&\quad \{d_{m+1}, e, e_1, \dots, e_{n-2}, g_1, \dots, g_{n-1}, p_0, \dots, p_{n-2}, \texttt{yes}\};\\
L_2 &= \{v_1\};\\
L_{k+i} &= \{p_{n-2-i}\},\ k = m+3, \quad 0 \le i \le n-2;\\
L_j &= \emptyset,\ 1 \le j \le m+n+2, j \ne 2, j \ne k+i, k = m+3, \quad 0 \le i \le n-2.
\end{aligned}$$

The bracket representation of the initial underlying compartment structure of Π is

$$[_0\ [_1\ [_k\ p_{n-2}\ ]_k^0\ \dots\ [_{k+n-2}\ p_0\ ]_{k+n-2}^0\ [_{k+n-1}\ ]_{k+n-1}^0\ [_2\ v_1\ [_3\ ]_3^0\ \dots [_{2+m}\ ]_{2+m}^0\ ]_2^0\ ]_1^0\ ]_0.$$

The set R consists of the following rules:

$$\begin{aligned}
1 &: [_2\ v_i\ ]_2^0 \to [_2\ F_i\ ]_2^0\ [_2\ T_i\ ]_2^0 \quad \text{for } 1 \le i \le n,\\
2 &: [_2\ F_i \to \mathit{false}(v_i) v_{i+1}\ ]_2^0 \quad \text{for } 1 \le i \le n-1,\\
3 &: [_2\ T_i \to \mathit{true}(v_i) v_{i+1}\ ]_2^0 \quad \text{for } 1 \le i \le n-1,\\
4 &: [_2\ F_n \to \mathit{false}(v_n) d_1\ ]_2^0\\
5 &: [_2\ T_n \to \mathit{true}(v_n) d_1\ ]_2^0\\
6 &: c_i\ [_{2+i}\ ]_{2+i}^0 \to [_{2+i}\ ]_{2+i}^+ \quad \text{for } 1 \le i \le m,\\
7 &: d_i\ [_{2+i}\ ]_{2+i}^+ \to [_{2+i}\ d_i\ ]_{2+i}^+ \quad \text{for } 1 \le i \le m,\\
8 &: [_{i+2}\ d_i\ ]_{i+2}^+ \to [_{i+2}\ ]_{i+2}^-\ d_{i+1} \quad \text{for } 1 \le i \le m,\\
9 &: [_2\ d_{m+1}\ ]_2^0 \to [_2\ ]_2^0\ d_{m+1}\\
10 &: [_{k+j}\ p_i\ ]_{k+j}^0 \to [_{k+j}\ p_{i-1}\ ]_{k+j}^0\ [_{k+j}\ p_{i-1}\ ]_{k+j}^0 \quad \text{for } 0 \le j \le n-2, 1 \le i \le n-2\\
11 &: [_{k+j}\ p_0\ ]_{k+j}^0 \to [_{k+j}\ ]_{k+j}^+\ [_{k+j}\ ]_{k+j}^+ \quad \text{for } 0 \le j \le n-2\\
12 &: d_{m+1}\ [_k\ ]_k^+ \to [_k\ e\ ]_k^-
\end{aligned}$$

$13: d_{m+1}\ [_k\ ]_k^- \to [_k\ ]_k^0$
$14: [_k\ e\ ]_k^0 \to [_k\ ]_k^0\ g_1$
$15: g_i\ [_{k+i}\ ]_{k+i}^+ \to [_{k+i}\ e_i\ ]_{k+i}^-$ for $1 \le i \le n-2$
$16: g_i\ [_{k+i}\ ]_{k+i}^- \to [_{k+i}\ ]_{k+i}^0$ for $1 \le i \le n-2$
$17: [_{k+i}\ e_i\ ]_{k+i}^0 \to [_{k+i}\ ]_{k+i}^0\ g_{i+1}$ for $1 \le i \le n-2$
$18: d_{m+1}\ [_{k+n-1}\ ]_{k+n-1}^0 \to [_{k+n-1}\ ]_{k+n-1}^+$
$19: g_{n-1}\ [_{k+n-1}\ ]_{k+n-1}^+ \to [_{k+n-1}\ \mathtt{yes}\ ]_{k+n-1}^-$
$20: [_{k+n-1}\ \mathtt{yes}\ ]_{k+n-1}^- \to \mathtt{yes}$
$21: [_1\ \mathtt{yes}\ ]_1^0 \to [_1\ ]_1^0\ \mathtt{yes}$

where $k = m + 3$.

We have numbered the (groups of) rules in order to facilitate the explanation.

The first part of this proof is very similar to the proof of Theorem 1. In compartment 2 the truth assignments for the n variables are generated by the rules $1, \ldots, 5$. Compartment 2 alternates between its division and the introduction of symbols. During division a value (T for TRUE and F for FALSE) is associated with variable v_i; the introduced symbols are the ones associated (by the functions *false* and *true*) with the clause satisfied by v_i and $\bar{v}_i$. After $2n$ transitions all the 2^n truth assignments for ψ are generated and present in compartments 2 (called *2-named compartment*, in the following). It is important to notice that the application of rule 1 sees the duplication of non-elementary compartments, that is, compartments with other compartments inside.

Every time a symbol c_i, $1 \le i \le n$, is generated, the application of rule 6 lets this symbol enter compartment $2 + i$ and makes this compartment change polarity. This means that in each 2-named compartment, rule 6 for a particular i can be applied at most once. We remind the reader that, as the system operates in an asynchronous way, the presence of a c_i in a compartment does not imply the immediate application of rule 6.

When rules 4 and 5 are applied in a 2-named compartment, the introduction of d_1 primes the counting of the compartments given a positive polarity by rule 6 in that 2-named compartment. This counting takes place by the application of rules 7 and 8. It should be clear that the d_i symbol cannot visit the same $(i + 2)$-named compartment, $1 \le i \le n$, more than once. This is because once such a compartment has been visited by d_i (rule 7), then the application of rule 8 let the polarity of that compartment to change such that rule 7 cannot be applied again for a specific i. It should also be clear that if an $(i + 2)$-named compartment cannot be present (because, for instance, the truth assignment associated to that 2-named compartment does not satisfy clause i) or is not yet present in a 2-named compartment (because that particular rule 6 has not been applied, yet), then d_{i+1} cannot be generated in that 2-named compartment.

If the truth assignment associated to a 2-named compartment satisfies ψ, then, sooner or later, d_{m+1} will be present in this 2-named compartment. When this happens rule 9 is applied, so that the system lets d_{m+1} pass to compartment 1. Eventually there will be a d_{m+1} in compartment 1 for each of the assignments satisfying ψ.

The compartments $k + i$, $k = m + 3$, $0 \le i \le n - 2$ present in the current system are used to count the number of assignments satisfying ψ. If this number is bigger than 2^{n-1}, then yes will pass to the environment.

We describe how the count of the assignments satisfying ψ takes place in Π with the help of an example. The generalisation to the computations of Π should be straightforward.

Consider, for example, that $n = 5$, so that there are $2^5 = 32$ assignments. Π let yes pass to the environment only if at least $16 + z$, $z \ge 1$, assignments verify ψ. If less than $16 + z$ assignments verify ψ, then yes will never pass to the environment. So, the system must count the number of assignments verifying ψ. This counting has to consider that Π operates in an asynchronous way (so the d_{m+1} symbols can arrive in compartment 1 at any time) and that the number of d_{m+1} may be exponential in the input size. This means that the counting of the d_{m+1} has to be done in an efficient way: it cannot be done in a 'linear' way, one after the other, because in this case the best time complexity would be exponential.

Let us assume that there are $16 + z$, $z \ge 1$, assignments which verify ψ. 16 of these assignments are counted in an efficient way. As $n = 5$, initially compartment 1 contains: $[_k\ p_3\ ]_k^0$, $[_{k+1}\ p_2\ ]_{k+1}^0$, $[_{k+2}\ p_1\ ]_{k+2}^0$ and $[_{k+3}\ p_0\ ]_{k+3}^0$. Rules 10 and 11 let these compartments duplicate such that at the end of this process:

8 copies of $[_k\ ]_k^+$
4 copies of $[_{k+1}\ ]_{k+1}^+$
2 copies of $[_{k+2}\ ]_{k+2}^+$
1 copy of $[_{k+3}\ ]_{k+3}^+$

are present. These compartments are given a positive polarity during their last division to identify that they have finished dividing.

As we will see:

8 pairs (that is, 16 copies) of d_{m+1} can enter the 8 k-named compartments. 8 copies of g_1 will be generated by these 8 k-named compartments;

4 pairs (that is, 8 copies) of g_1 can enter the 4 $(k+1)$-named compartments. 4 copies of g_2 will be generated by these $(k+1)$-named compartments;

2 pairs (that is, 4 copies) of g_2 can enter the 2 $(k+2)$-named compartments. 2 copies of g_3 will be generated by these $(k+1)$-named compartments;

1 pair (that is, 2 copies) of g_3 can enter the only $(k+3)$-named compartment. 1 copy of g_4 will be generated by this $(k+3)$-named compartment.

So, if g_4 is present in Π, then at least half of the assignments verify ψ. It should be clear that the counting of the $x = 16$ (in this example) copies of d_{m+1} is efficient: the number of d_{m+1} is halved, then the result is halved, and so on. As this process can take place in parallel, then in the best case the counting will require $\log_2 2^{n-1} = n - 1$ steps.

Let us indicate now what rules in R perform what we have just described in the example. As already said, group of rules 10 and 11 create the compartments needed for the counting. Rules 12, 13 and 14 let the first 'halving' to take place:

rule 12 allows one d_{m+1} to pass into a k-named compartment, changing its polarity from positive to negative. While doing so d_{m+1} becomes e;

rule 13 allows a second d_{m+1} to pass into a negatively charged k-named compartment. Doing so the polarity of the compartment changes into neutral;

rule 14 allows the e present in a neutral k-named compartment to go outside it as a g_1. As the polarity of this compartment remains negative, no other rule can be applied to it.

Rules 15, 16 and 17 operate in a similar way: they keep 'halving' the number of g symbols until, eventually, a copy of g_{n-1} appears in compartment 1.

Now the system has to do a very simple thing: check if compartment 1 contains a copy of g_{n-1} and at least one copy of d_{m+1}. If this is the case, then a `yes` passes to the environment. This check is performed by rules 18 and 19. Rule 18 allows a d_{m+1} to change the polarity of a neutrally charged $(k+n-1)$-named compartment into positive. After this happened, the application of rule 19 let the g_{n-1} to pass into this compartment as `yes` and to change the polarity to negative. When this happens `yes` passes to compartment 1 (rule 20) and from there to the environment (rule 21).

It should be clear that the statement holds. □

The system Π defined in the proof of Theorem 2 is such that `yes` passes to the environment only if the instance of MAJORITY-SAT is satisfied. Otherwise Π halts without letting anything pass to the environment. As stated in Section 2, this is not the behaviour usually assumed in the literature: recognising P systems should instead send a `no` to the environment in rejecting computations. However, the system Π can be used to create a recognising system Π' that behaves exactly in this way. The system Π' can be logically divided in two sub-systems Π'_1 and Π'_2, operating in parallel:

Π'_1 does exactly what Π does;

Π'_2 checks (in a way similar to Π) if at least 2^{n-1} assignments verify ψ', where ψ' is as ψ but such that each clause is negated. If this is the case, then `no` passes to the environment.

Indeed, the class **PP** is closed under complementation, and the above construction can be replicated for all **PP** problems thanks to the following theorem.

Theorem 3. *Let $L \in$ **PP**. Then L can be solved by asynchronous recognising P systems with active membranes with polarities, and using only rules of the kind* (a), (b), (c) *and* (e). *The P system requires exponential space and, in the best case, linear time.*

Proof. Let R be the deterministic Turing machine that computes the polynomial time reduction from L to MAJORITY-SAT. Moreover, let M the deterministic Turing machine that, given any instance ψ of MAJORITY-SAT, computes in polynomial time (w.r.t. the size of ψ) the encoding of the recognising P system $\Pi_{\psi}^{MAJ-SAT}$ that solves such an instance of MAJORITY-SAT, as described in Theorem 2.

We can then define a deterministic Turing machine T that, given any instance x of L, first simulates R to produce an equivalent instance ψ of MAJORITY-SAT, and then simulates M to produce the recognising P system $\Pi_{\psi}^{MAJ-SAT}$. Let us call Π_x^L such a P system. By construction, the system Π_x^L lets pass the symbol `yes` to the environment if and only if ψ is a positive instance of MAJORITY-SAT. By reduction, this happens if and only if x is a positive instance of L. Hence T can be seen as a deterministic Turing machine that, given any instance x of L, produces in polynomial time an encoding of the recognising P system Π_x^L that solves x. This concludes the proof. □

Clearly the same construction can be done for all problems in **NP**, relying on Theorem 1. For the sake of completeness, we explicit it in the following.

Theorem 4. *Let $L \in$ **NP**. Then L can be solved by asynchronous recognising P systems with active membranes without polarities, and using only rules of the kind* (a), (b), (c), (e) *and* (g). *The P system requires exponential space and, in the best case, linear time.*

Proof. Let R be the deterministic Turing machine that computes the polynomial time reduction from L to k-SAT, for a fixed value of $k \geq 3$. Moreover, let M the deterministic Turing machine that, given any instance ψ of k-SAT, computes in polynomial time (w.r.t. the size of ψ) the encoding of the recognising P system Π_ψ^{k-SAT} that solves such an instance of k-SAT, as described in Theorem 1.

We can then define a deterministic Turing machine T that, given any instance x of L, first simulates R to produce an equivalent instance ψ of k-SAT, and then simulates M to produce the recognising P system Π_ψ^{k-SAT}. Let us call Π_x^L such a P system. By construction, the system Π_x^L lets pass the symbol `yes` to the environment if and only if ψ is a positive instance of k-SAT. By reduction, this happens if and only if x is a positive instance of L. Hence T can be seen as a deterministic Turing machine that, given any instance x of L, produces in polynomial time an encoding of the recognising P system Π_x^L that solves x. This concludes the proof. □

Recalling that by $b\mathbf{PMC}^*_{\mathcal{AM}}$ we denote the class of languages (equivalently, decision problems) that can be recognised (resp., solved) in (best case) polynomial time by semi-uniform families of recognising P systems with active membranes operating in asynchronous mode, we can summarise the results we have obtained as follows:

$$\mathbf{PP} \subseteq b\mathbf{PMC}^*_{\mathcal{AM}} \quad \text{and} \quad \mathbf{NP} \subseteq b\mathbf{PMC}^*_{\mathcal{AM}}.$$

Since $\mathbf{NP} \subseteq \mathbf{PP}$, we can also write $\mathbf{NP} \subseteq \mathbf{PP} \subseteq b\mathbf{PMC}^*_{\mathcal{AM}}$. Moreover, since the class **PP** is closed under complementation, all recognising P systems solving a **PP** problem can be assumed to let pass the symbol `no` to the environment in the last transition of all rejecting computations. Note that we cannot make the same assumption regarding recognising P systems solving **NP** problems, since it is not currently known whether the class **NP** is closed under complementation (indeed, it is believed that the classes **NP** and **coNP** are disjoint).

5. Computational power

In [1], Theorem 3, it is proved that P systems with active membranes, using polarities and rules of type (a), (b) and (c) and operating under minimal parallelism can simulate register machines. In the following we prove that in these P systems the presence of polarities can be replaced by rules of type (d) and (g).

As expected (by some results in [5,4]), if the operational mode of the P system in [1, Th.3] is changed into asynchronous, then the resulting system is equivalent to a partially blind program machine.

Lemma 1. *Asynchronous P systems with active membranes without polarities, using only rules of type* (a)–(d) *and* (g) *can simulate partially blind register machines.*

Proof. Let $M = (S, I, s_1, s_f)$ be a partially blind register machine with n registers γ_i, $1 \leq i \leq n$, whose initial value is 0. We describe a P system with active membranes $\Pi = (V, \mu, L_1, L_\#, R)$ simulating M where:

$$\begin{aligned} V &= S \cup \{c_i \mid 1 \leq i \leq n\} \cup \{d_{i,v} \mid (s, \gamma_i^-, v) \in I\} \cup \{\#\}; \\ \mu &= (\{0, 1, \#\}, \{(0, 1), (1, \#)\}); \\ L_1 &= \{s_1\}; \\ L_\# &= \{\#\}. \end{aligned}$$

The bracket representation of the initial underlying compartment structure of Π is

$$[_0 [_1\ s_1\ ]_1 [_\#\ \#]_\#]_0.$$

The rules in R are given in the following.

For each instruction of the kind $(s, \gamma_i^+, v) \in I$ the set R contains:

$$\begin{aligned} 1 &: [_1\ s \rightarrow c_i\ ]_1 \\ 2 &: [_1\ c_i\ ]_1 \rightarrow [_1 [_{c_i}\ v\ ]_{c_i}]_1 \\ 3 &: [_{c_i}\ v\ ]_{c_i} \rightarrow [_{c_i}\ ]_{c_i}\ v \end{aligned}$$

while for each rule of the kind $(s, \gamma_i^-, v) \in I$ the set R contains:

$$\begin{aligned} 4 &: [_1\ s \rightarrow d_{i,v}\ ]_1 \\ 5 &: d_{i,v}\ [_{c_i}]_{c_i} \rightarrow [_{c_i}\ d_{i,v}\ ]_{c_i} \\ 6 &: [_1 [_{c_i}\ d_{i,v}\ ]_{c_i}]_1 \rightarrow [_1\ v\ ]_1. \end{aligned}$$

The set R contains also:

$$\begin{aligned} 7 &: [_\#\ \#\ ]_\# \rightarrow [_\#]_\#\ \# \\ 8 &: \#\ [_\#]_\# \rightarrow [_\#\ \#\ ]_\# \\ 9 &: s_f\ [_\#]_\# \rightarrow [_\#\ s_f\ ]_\# \\ 10 &: [_1 [_\#\ s_f\ ]_\#]_1 \rightarrow [_1\ s_f\ ]_1. \end{aligned}$$

As usual, we have numbered the rules to facilitate the explanation. The addition of 1 to register γ_i in M is simulated in Π by the creation of a compartment c_i inside compartment 1. The subtraction of 1 to register γ_i in M is simulated in Π by the dissolution of a c_i compartment present in compartment 1. The symbol s_1, indicating the simulation of M being in its initial state, is initially present in compartment 1. In general, in Π the presence of an element $s \in S$ in compartment 1 indicates the simulation of M being in state s. The system Π simulates M faithfully: Π can halt with a number of c_i compartments equal to $val(\gamma_i)$, $1 \leq i \leq n$, if and only if M halts with its registers having values $val(\gamma_i)$, respectively. If Π simulates an instruction of the kind (s, γ_i^-, v) on an empty register, then Π never halts.

Rules 7 and 8 let Π compute forever as they let symbol # to go in and out from compartment #. This can be interrupted only by the application of rules 9 and 10: s_f, the final state of M can enter compartment # and dissolve it so that rules 7 and 8 can no longer be applied. This can happen only if all instructions have been simulated accordingly.

The simulation of instructions of the kind (s, γ_i^+, v) is performed by the sequential application of rules 1, 2 and 3, while the simulation of instructions of the kind (s, γ_i^-, v) is performed by the sequential application of rules 4, 5 and 6. Of course, because of the operational mode, these rules performing simulations can be interleaved by (or occur in parallel with) rules 7 and 8. This does not affect the overall computation. □

We were not able to prove the reverse of the inclusion stated in Lemma 1, that is that partially blind register machines can simulate asynchronous P systems with active membranes without polarities, using only rules of type (a)–(d) and (g). We were only able to prove that partially blind register machines can simulate rules of the type (a)–(c), as shown in Lemma 3. The possibility to simulate rules of types (d) and (g) remains an open problem.

As stated by the following results the relation between partially blind register machines and asynchronous P systems with active membranes is close to be an equality when these P systems use only rules of type (a)–(c) as well as polarities.

Lemma 2. *Asynchronous P systems with active membranes with polarities, using only rules of type* (a)–(c) *can simulate partially blind register machines.*

Proof. Let $M = (S, I, s_1, s_f)$ be a partially blind register machine with n registers γ_i, $1 \leq i \leq n$, whose initial value is 0. We describe a P system with active membranes $\Pi = (V, \mu, L_1, L_\#, R)$ as in the statement, simulating M, where:

$$\begin{aligned}
V &= S \cup \{c_{i,v} \mid (s, \gamma_i^+, v) \in I\} \cup \{d_{i,v} \mid (s, \gamma_i^-, v) \in I\} \cup \{o, \#\};\\
\mu &= (\{0, 1, c_1, \ldots, c_n, \#\}, \{(0, 1), (1, c_1), \ldots, (1, c_n), (1, \#)\});\\
L_1 &= \{s_1\};\\
L_{c_i} &= \emptyset \quad \text{for } 1 \leq i \leq n;\\
L_\# &= \{\#\}.
\end{aligned}$$

The bracket representation of the initial underlying compartment structure of Π is

$$[_0[_1\ s_1\ ]_1[_{c_1}]_{c_1} \cdots [_{c_n}]_{c_n}[_\#\ \#]_\#]_0.$$

The rules in R are given in the following.

For each instruction of the kind $(s, \gamma_i^+, v) \in I$ the set R contains:

$$\begin{aligned}
1 &: [_1\ s \to c_{i,v}\ ]_1\\
2 &: c_{i,v}\ [_{c_i}]_{c_i}^0 \to [_{c_i}\ ov\ ]_{c_i}^0\\
3 &: [_{c_i}\ v\ ]_{c_i}^0 \to [_{c_i}\ ]_{c_i}^0\ v
\end{aligned}$$

while for each rule of the kind $(s, \gamma_i^-, v) \in I$ the set R contains:

$$\begin{aligned}
4 &: [_1\ s \to d_{i,v}\ ]_1\\
5 &: d_{i,v}\ [_{c_i}]_{c_i}^0 \to [_{c_i}\ d_{i,v}\ ]_{c_i}^+\\
6 &: [_{c_i}\ o\ ]_{c_i}^+ \to [_{c_i}\ ]_{c_i}^-\\
7 &: [_{c_i}\ d_{i,v}\ ]_{c_i}^- \to [_{c_i}\ ]_{c_i}^0\ v.
\end{aligned}$$

The set R contains also:

$$\begin{aligned}
8 &: [_\#\ \#\ ]_\#^0 \to [_\#]_\#^0\ \#\\
9 &: \#\ [_\#]_\#^0 \to [_\#\ \#\ ]_\#^0\\
10 &: s_f\ [_\#]_\#^0 \to [_\#\ s_f\ ]_\#^+.
\end{aligned}$$

In order to facilitate the explanation, rules have been numbered. The addition of 1 to register γ_i in M is simulated in Π by the addition of a copy of symbol o in compartment c_i. The subtraction of 1 to register γ_i in M is simulated in Π by the removal of a copy of symbol o in compartment c_i. The symbol s_1, indicating the simulation of M being in its initial state, is initially present in compartment 1. In general, in Π the presence of an element $s \in S$ in compartment 1 indicates the simulation of M being in state s. The system Π simulates M faithfully: Π can halt with a number of o symbols in compartments c_i equal to $val(\gamma_i)$, $1 \leq i \leq n$, if and only if M halts with its registers having values $val(\gamma_i)$, respectively. If Π simulates an instruction of the kind (s, γ_i^-, v) on an empty register, then Π never halts.

Rules 8 and 9 let Π compute forever as they let symbol # to go in and out from compartment # when this compartment has neutral charge. This can be interrupted only by the application of rule 10: s_f, the final state of M, can enter compartment # and change its polarity to positive so that rules 8 and 9 can no longer be applied. This can happen only if all instructions have been simulated accordingly.

The simulation of instructions of the kind (s, γ_i^+, v) is performed by the sequential application of rules 1, 2 and 3, while the simulation of instructions of the kind (s, γ_i^-, v) is performed by the sequential application of rules 4, 5, 6 and 7. Of course, because of the operational mode, these rules performing simulations can be interleaved by (or occur in parallel with) rules 8 and 9. This does not effect the overall computation. □

Lemma 3. *Partially blind register machines can simulate rules of type* (a)–(c) *in asynchronous P systems with active membranes with polarities.*

Proof. Let $\Pi = (V, \mu, L_1, \ldots, L_n, R)$ a P system as in the statement. The system Π uses a total of $|V|$ symbols and, because of the available rules, there will be always n compartments in it. The partially blind program machine $M = (S, I, s_1, s_f)$ has $n|V|$ registers with names $\gamma_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq |V|$. In general, the content of register $\gamma_{i,j}$ reflects how many instances of symbol j are present in compartment i. The rules of Π are simulated by M in a non-deterministic way: any of the rules in R can be chosen to be simulated. If the simulation is completed, then M can go on trying to simulate another rule, otherwise M halts in a non-final state.

The finite state control of M keeps track of the compartment hierarchy and the polarity of each compartment (clearly, this information is linear in the size of the input). For instance, we can assume that this information is stored as a subscript in each state of S. This information is used when rules of type (b) and (c) are simulated (because it is needed to know where the symbols are moved from/to) and it is changed when the application of a rule let the polarity of a compartment to change.

The simulation of rules of type (a) $([_i\ v \rightarrow w\ ]_i^p, w = w_1 \ldots w_{|w|})$ is performed by first decreasing by one the content of $\gamma_{i,v}$ and then sequentially increasing by one the content of γ_{i,w_k}, for all $1 \leq k \leq |w|$. The simulation of the other kinds of rules is performed in a similar way.

It should be clear that the simulation is faithful. □

Now one would expect to have a theorem showing that the computational power of asynchronous P systems with active membranes with polarities using only rules of type (a)–(c) is equivalent to the one of partially blind register machines. Unfortunately we do not have such a general statement. The problem we were not able to overcome is the simulation of the termination of a P system by such a register machine. If the termination of the P system is due to, for instance, a specific symbol entering/leaving a compartment, then the equivalence is straightforward. This kind of termination in asynchronous P system has been used in, for instance, [3]. If instead it is assumed that such a P system terminates when there is no applicable rule, then the equivalence cannot be there as a partially blind program machine cannot test if something cannot be done.

6. Final remarks

Semi-uniform families of recognising P systems with active membranes have been proved to be able to solve all **NP** and all **PP** problems in (best case) polynomial time, by using either polarities or creation and dissolution rules, when working in the asynchronous operational mode. In short, this can be written as $\mathbf{NP} \subseteq \mathbf{PP} \subseteq b\mathbf{PMC}^*_{\mathcal{AM}}$. The worst case execution time, as well as the amount of space used to perform the computations, remain exponential. An outstanding open problem is whether the class $b\mathbf{PMC}^*_{\mathcal{AM}}$ coincides with the standard complexity class **PSPACE**.

In the previous section we have also shown that the model of P systems studied in this paper are able to simulate partially blind register machines, either using polarities or trading them with creation and dissolution rules. The possibility to perform the converse simulation has also been discussed, highlighting the difficulties that did not allow us to prove the equivalence between the two computational models.

As we said in Section 3, in [5,4,2] some studies relating different operational modes are present. Unfortunately, this study is far from being complete. It would be very interesting to study how different operational modes relate to each other, if they form a hierarchy and if this hierarchy is valid for all formal systems or not. We continue to investigate also on this further.

Another natural question that may be posed is: *is asynchronous operational mode the most general?* The answer is negative, because one can think to even more general operational modes. Asynchronous mode assumes that different kinds of rules need the same time to be completed and that the application of each rule is atomic. One could, for instance, assume that rules of type e take more time to be completed than rules of other types and that the implementation of this kind of rules is sequential: first one compartment is created and then the other one. Results obtained with asynchronous operational mode would not necessarily be valid with these relaxed assumptions.

References

[1] G. Ciobanu, L. Pan, G. Păun, M. J. Pérez-Jiménez, P systems with minimal parallelism, Theoretical Computer Science 378 (1) (2007) 117–130.

[2] R. Freund, S. Verlan, A formal framework for static (tissue) P systems, in: G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 2007, Revised Selected and Invited Papers, in: LNCS, vol. 4860, Springer-Verlag, 2007, pp. 271–284.

[3] P. Frisco, The conformon-P system: a molecular and cell biology-inspired computability model, Theoretical Computer Science 312 (2–3) (2004) 295–319.

[4] P. Frisco, A hierarchy of computational processes. Technical report, Heriot-Watt University, 2008. HW-MACS-TR-0059 http://www.macs.hw.ac.uk:8080/~techreps/index.html.

[5] P. Frisco, Computing with Cells. Advances in Membrane Computing, Oxford University Press, 2009.

[6] P. Frisco, G. Govan, P systems with active membranes operating under minimal parallelism. In: 12th Conference on Membrane Computing, 2011 (in press).

[7] G. Mauri, A. Leporati, A.E. Porreca, C. Zandron, Recent complexity-theoretic results on P systems with active membranes, Journal of Logic and Computation (2011) (in press).

[8] N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Natural Computing 10 (1) (2011) 613–632.

[9] C.H. Papadimitriou, Computational Complexity, Addison-Wesley Pub. Co, 1994.

[10] G. Păun, Computing with membranes, Journal of Computer and System Sciences 1 (61) (2000) 108–143.

[11] G. Păun, Membrane Computing. An Introduction, Springer-Verlag, 2002.

[12] G. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2010.

[13] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity classes in models of cellular computing with membranes, Natural Computing 2 (3) (2003) 265–285.

[14] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Computationally hard problems addressed through P systems, in: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.), Applications of Membrane Computing, Springer-Verlag, 2006, pp. 313–346.

[15] A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems with elementary active membranes: beyond NP and coNP, in: Proceedings of the 11th International Conference on Membrane Computing, CMC 10, in: LNCS, vol. 6501, Springer-Verlag, 2010, pp. 338–347.

[16] A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, Elementary active membranes have the power of counting, International Journal of Natural Computing Research 2 (3) (2011) 35–48. doi:10.4018/jncr.2011070104.

[17] J. Van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier and MIT Press, 1990.

[18] C. Zandron, C. Ferretti, G Mauri, Solving NP-complete problems using P systems with active membranes, in: Proceedings of the Second International Conference on Unconventional Models of Computation, Springer-Verlag, 2001, pp. 289–301.