

Active Membrane Systems Without Charges and Using Only Symmetric Elementary Division Characterise \mathbf{P}

Niall Murphy¹ and Damien Woods²

¹ Department of Computer Science, National University of Ireland, Maynooth, Ireland
nmurphy@cs.nuim.ie

² Department of Computer Science, University College Cork, Ireland
d.woods@cs.ucc.ie

Abstract. In this paper we introduce a variant of membrane systems with elementary division and without charges. We allow only elementary division where the resulting membranes are identical; we refer to this using the biological term symmetric division. We prove that this model characterises \mathbf{P} and introduce logspace uniform families. This result characterises the power of a class of membrane systems that fall under the so-called \mathbf{P} conjecture for membrane systems.

1 Introduction

The \mathbf{P} -conjecture states that recogniser membranes systems with division rules (active membranes [6]), but without charges, characterise \mathbf{P} . This was shown for a restriction of the model: without dissolution rules [4]. However, it has been shown that systems with dissolution rules and non-elementary division characterise \mathbf{PSPACE} [2,9]. In this setting, using dissolution rules allows us to jump from \mathbf{P} to \mathbf{PSPACE} . As a step towards finding a bound (upper or lower) on systems with only elementary division rules, we propose a new restriction, and show that it has an upper bound of \mathbf{P} .

Our restriction insists that the two membranes that result from an elementary division rule must be identical. This models mitosis, the biological process of cell division [1] and we refer to it using the biological term “symmetric division”. We refer to division where the two resulting daughter cells are different by the biological term “asymmetric division”. In nature asymmetric division occurs, for example, in stem cells as a way to achieve cell differentiation.

Since our model is uniform via polynomial time deterministic Turing machines, it trivially has a lower bound of \mathbf{P} . However, we introduce logspace uniformity for this model and then prove a \mathbf{P} lower bound. All recogniser membrane systems with division rules are upper bounded by \mathbf{PSPACE} [9]. In this paper we show that systems with symmetric elementary division and without charges are upper bounded by \mathbf{P} . From an algorithmic point of view, this result allows one to write a polynomial time algorithm that models certain membrane systems which use exponential numbers of membranes and objects.

2 Preliminaries

In this section we define membrane systems and complexity classes. These definitions are from Păun [6,7], and Sosík and Rodríguez-Patón [9]. We also introduce the notion of logspace uniformity for membrane systems. We give a \mathbf{P} lower bound for the model that is the main focus of this paper.

2.1 Recogniser Membrane Systems

Active membranes systems are membrane systems with membrane division rules. Division rules can either only act on elementary membranes, or else on both elementary and non-elementary membranes. An elementary membrane is one which does not contain other membranes (a leaf node, in tree terminology). In Definition 1 we make a new distinction between two types of elementary division rules. When we refer to *symmetric division* (e_s) we mean division where the resulting two child membranes are identical. When the two child membranes are not identical we refer to the rule as being *asymmetric* (e).

Definition 1. *An active membrane system without charges using elementary division is a tuple $\Pi = (V, H, \mu, w_1, \dots, w_m, R)$ where,*

1. $m > 1$ the initial number of membranes;
2. V is the alphabet of objects;
3. H is the finite set of labels for the membranes;
4. μ is a membrane structure, consisting of m membranes, labelled with elements of H ;
5. w_1, \dots, w_m are strings over V , describing the multisets of objects placed in the m regions of μ .
6. R is a finite set of developmental rules, of the following forms:
 - (a) $[a \rightarrow v]_h$,
for $h \in H, a \in V, v \in V^*$
 - (b) $a[{}_h]_h \rightarrow [{}_h b]_h$,
for $h \in H, a, b \in V$
 - (c) $[{}_h a]_h \rightarrow [{}_h]_h b$,
for $h \in H, a, b \in V$
 - (d) $[{}_h a]_h \rightarrow b$,
for $h \in H, a, b \in V$
 - (e_s) $[{}_h a]_h \rightarrow [{}_h b]_h [{}_h b]_h$,
for $h \in H, a, b \in V$
 - (e) $[{}_h a]_h \rightarrow [{}_h b]_h [{}_h c]_h$,
for $h \in H, a, b, c \in V$.

These rules are applied according to the following principles:

- All the rules are applied in maximally parallel manner. That is, in one step, one object of a membrane is used by at most one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form, must evolve.

- If at the same time a membrane labelled with h is divided by a rule of type (e) or (e_s) and there are objects in this membrane which evolve by means of rules of type (a) , then we suppose that first the evolution rules of type (a) are used, and then the division is produced. This process takes only one step.
- The rules associated with membranes labelled with h are used for membranes with that label. At one step, a membrane can be the subject of only one rule of types (b) - (e) .

In this paper we study the language recognising variant of membrane systems which solves decision problems. A distinguished region contains, at the beginning of the computation, an input — a description of an instance of a problem. The result of the computation (a solution to the instance) is “yes” if a distinguished object **yes** is expelled during the computation, otherwise the result is “no”. Such a membrane system is called *deterministic* if for each input a unique sequence of configurations exists. A membrane system is called *confluent* if it always halts and, starting from the same initial configuration, it always gives the same result, either always “yes” or always “no”. Therefore, given a fixed initial configuration, a confluent membrane system non-deterministically chooses from one from a number of valid configuration sequences and rule applications but all of them must lead to the same result.

2.2 Complexity Classes

Complexity classes have been defined for membrane systems [8]. Consider a decision problem X , i.e. a set of instances $\{x_1, x_2, \dots\}$ over some finite alphabet such that to each x_i there is an unique answer “yes” or “no”. We consider a *family* of membrane systems to solve a decision problem if each instance of the problem is solved by some class member.

We denote by $|x_i|$ the size of any instance $x_i \in X$.

Definition 2 (Polynomial uniform families of membrane systems). *Let \mathcal{D} be a class of membrane systems and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a total function. The class of problems solved by uniform families of membrane systems of type \mathcal{D} in time f , denoted by $\text{MC}_{\mathcal{D}}(f)$, contains all problems X such that:*

- *There exists a uniform family of membrane systems, $\Pi_X = (\Pi_X(1); \Pi_X(2); \dots)$ of type \mathcal{D} : each $\Pi_X(n)$ is constructable by a deterministic Turing machine with input n and in time that is polynomial of n .*
- *Each $\Pi_X(n)$ is sound: $\Pi_X(n)$ starting with an input (encoded by a deterministic Turing machine in polynomial time) $x \in X$ of size n expels out a distinguished object **yes** if and only if the answer to x is “yes”.*
- *Each $\Pi_X(n)$ is confluent: all computations of $\Pi_X(n)$ with the same input x of size n give the same result; either always “yes” or else always “no”.*
- *Π_X is f -efficient: $\Pi_X(n)$ always halts in at most $f(n)$ steps.*

Polynomial Semi-uniform families of membrane systems $\Pi_X = (\Pi_X(x_1); \Pi_X(x_2); \dots)$ whose members $\Pi_X(x_i)$ are constructable by a deterministic Turing machine with input x_i in a polynomial time with respect to $|x_i|$. In this case,

for each instance of X we have a special membrane system which therefore does not need an input. The resulting class of problems is denoted by $\mathbf{MC}_D^S(f)$. Obviously, $\mathbf{MC}_D(f) \subseteq \mathbf{MC}_D^S(f)$ for a given class \mathcal{D} and a complexity [3] function f .

We denote by

$$\mathbf{PMC}_D = \bigcup_{k \in \mathbb{N}} \mathbf{MC}_D(O(n^k)), \quad \mathbf{PMC}_D^S = \bigcup_{k \in \mathbb{N}} \mathbf{MC}_D^S(O(n^k))$$

the class of problems solvable by uniform (respectively semi-uniform) families of membrane systems in polynomial time. We denote by \mathcal{AM} the classes of membrane systems with active membranes. We denote by \mathcal{EAM} the classes of membrane systems with active membranes and only elementary membrane division. We denote by \mathcal{AM}_{-a}^0 (respectively, \mathcal{AM}_{+a}^0) the class of all recogniser membrane systems with active membranes without charges and without asymmetric division (respectively, with asymmetric division). We denote by $\mathbf{PMC}_{\mathcal{EAM}_{-a}^0}^S$ the classes of problems solvable by semi-uniform families of membrane systems in polynomial time with no charges and only symmetric elementary division. We let $\text{poly}(n)$ be the set of polynomial complexity functions of n .

2.3 (Semi-)Uniformity Via logspace Turing Machines

In Theorem 2 we prove that $\mathbf{PMC}_{\mathcal{EAM}_{-a}^0}^S$ has a \mathbf{P} upper bound. When we use (semi-)uniform families constructed in polynomial time by deterministic Turing machines we trivially have a \mathbf{P} lower bound. However, to ensure that the membrane system itself is able to solve any problem in \mathbf{P} , and is not benefiting from

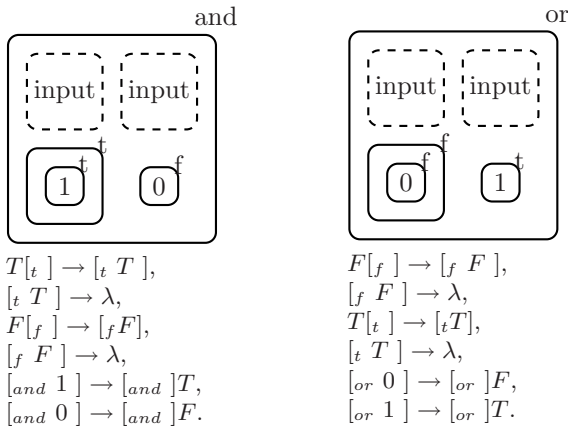


Fig. 1. AND and OR gadgets which can be nested together to simulate a circuit. The input is either a T, F, or a nested series of gadget membranes. A NOT gate membrane can be made with the rules $[not T] \rightarrow [not]F, [not F] \rightarrow [not]T$.

preprocessing by the output of the encoding Turing machine or family generating Turing machine, we restrict both of these machines to be logspace Turing machines.

In the following theorem it is understood that $\mathbf{PMC}_{\mathcal{EAM}^0_a}^S$ is logspace uniform.

Theorem 1. $\mathbf{P} \subseteq \mathbf{PMC}_{\mathcal{EAM}^0_a}^S$

Proof. A logspace Turing machine encodes an instance of the CIRCUIT VALUE problem (CVP) [5] as a $\mathbf{PMC}_{\mathcal{EAM}^0_a}^S$ membrane system using the gadgets shown in Figure 1. The resulting membrane system directly solves the instance of CVP in polynomial time. \square

The main result of this paper, Theorem 2, holds for both logspace and polynomial families of $\mathbf{PMC}_{\mathcal{EAM}^0_a}^S$.

3 An Upper Bound on $\mathbf{PMC}_{\mathcal{EAM}^0_a}^S$

In this section we give an upper bound of \mathbf{P} on the membrane class $\mathbf{PMC}_{\mathcal{EAM}^0_a}^S$. We provide a random access machine (RAM) algorithm that simulates this class using a polynomial number of registers of polynomial length, in polynomial time. We begin with an important definition followed by informal description of our contribution.

Definition 3 (Equivallence class of membranes). *An equivalence class of membranes is a multiset of membranes where: each membrane shares a single parent, each has the same label, and each has identical contents. Further, only membranes without children can be elements of an equivalence class of size greater than one; each membrane with one or more children has its own equivalence class of size one.*

Throughout the paper, when we say that a membrane system has $|E|$ equivalence classes, we mean that $|E|$ is the minimum number of equivalence classes that includes all membranes of the system.

While it is possible for a computation path of $\mathbf{PMC}_{\mathcal{EAM}^0_a}^S$ to use an exponential number of equivalence classes, our analysis guarantees that there is another, equally valid, computation path that uses at most a polynomial number of equivalence classes. Our algorithm finds this path in polynomial time. Moreover, via our algorithm, after a single timestep the increase in the number of equivalence classes is never greater than $|E_0||V|$, the product of the number of initial equivalence classes and the number of object types in the system. Since the system is confluent, our chosen computation path is just as valid to follow as any alternative path.

In Section 3.2 we prove that by using our algorithm:

- Type (a) rules do not increase the number of equivalence classes since the rule has the same effect on each membrane of a given equivalence class.
- Type (c) rules do not increase the number of equivalence classes since objects exit all child membranes for the parent membrane (which is already an equivalence class with one membrane).
- Type (d) rules do not increase the number of equivalence classes since the rule is applied to all membranes in the equivalence class. The contents and child membranes are transferred to the parent (already an equivalence class).
- Type (e_s) rules do not increase the number of equivalence classes, only the number of membranes in the existing equivalence classes simply increase.

Type (b) rules require a more detailed explanation. In Section 3.3 we show that there is a deterministic polynomial sequential time algorithm that finds a computation path that uses only a polynomial number of equivalence classes.

Our RAM algorithm operates on a number of registers that can be thought of as a data structure (see Section 3.1). The data structure stores the state of the membrane system at each timestep. It compresses the amount of information to be stored by storing equivalence classes instead of explicitly storing all membranes. Each equivalence class contains the number of membranes in the class, a reference to each of the distinct objects in one of those membranes, and the number of copies of that distinct object. Type (a) rules could therefore provide a way to create exponential space. However, we store the number of objects in binary thus we store it using space that is the logarithm of the number of objects.

Our RAM algorithm operates in a deterministic way. To introduce determinism we sort all lists of object multisets by object multiplicity, then lexicographically. We sort all equivalence classes by membrane multiplicity, then by label, and then by object. We sort all rules by rules type, matching label, matching object, and then by output object(s). The algorithm iterates through the equivalence classes and applies all rules of type (a), (c), (d), and (e_s). It then checks to see if any rules of type (b) are applicable. If so, it takes each object in its sorted order and applies it to the relevant membranes in their sorted order.

Theorem 2. $\text{PMC}_{\mathcal{EAM}_a^0}^S \subseteq \mathbf{P}$

The proof is in the remainder of this section. The result holds for both logspace and polynomial time uniform membrane systems of type $\text{PMC}_{\mathcal{EAM}_a^0}^S$.

3.1 Structure of RAM Registers

Our RAM uses a number of binary registers that is a polynomial ($\text{poly}(n)$) of the length n of the input. The length of each register is bounded by a polynomial of n . For convenience our registers are grouped together in a data structure (as illustrated in Figure 2).

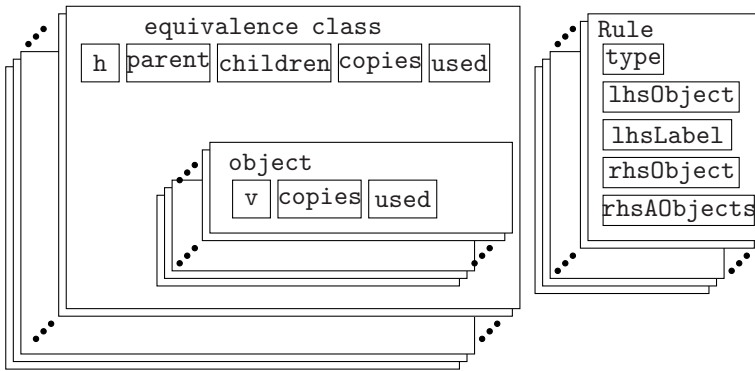


Fig. 2. A representation of our polynomial sized registers as a data structure

Object registers. For each distinct object type v_i , the following registers are used to encode the object in an equivalence class $e_k \in E$.

The register **v** represents the type of the object, $v_i \in V$ (see Definition 1). Throughout the computation, the size of the set V is fixed so this register does not grow beyond its initial size.

The register **copies** is the multiplicity of the distinct object v_i encoded in binary. At time 0 we have $|v_i|$ objects. At time 1 the worst case is that each object evolves via a type (a) rule to give a number of objects that is $\text{poly}(n)$. This is an exponential growth function, however, since we store it using binary, the register length does not grow beyond space that is $\text{poly}(n)$.

The register **used** represents the multiplicity v_i objects that have been used already in this computation step. It is always the case that $\text{used} \leq \text{copies}$ for each object type v_i .

Equivalence class registers. The following registers are used to store information about each equivalence class. To conserve space complexity we only explicitly store equivalence classes (rather than explicitly storing membranes); the number of equivalence classes is denoted $|E|$.

The register **h** stores the label of equivalence class e_k and is an element of the set H (see Definition 1). The size of register **h** is fixed and is bounded by $\text{poly}(n)$.

The register **parent** stores a reference to the equivalence class (a single membrane in this case) that contains this membrane. This value is bounded by the polynomial depth of the membrane structure. Since the depth of the membrane structure is fixed throughout a computation, the space required to store a parent reference is never greater than a logarithm of the depth.

The register **children** references all of the child equivalence classes of e_k at depth one. Its size is bounded by $\text{poly}(n)$ via Theorem 3.

The register **copies** stores the number, denoted $|e_k|$, of membranes in the equivalence class. We store this number in binary. In the worst case, the number

that is stored in copies doubles at each timestep (due to type (e_s) rules). Since we store this number in binary we use space that is $\text{poly}(n)$.

The register `used` stores the number of membranes in the equivalence class that have been used by some rule in the current timestep and so this value is $\leq |e_k|$.

Rules registers. The `rules` registers store the rules of the membrane system; their number is bounded by the polynomial $|R|$ and is fixed for all time t . The `rules` registers can not change or grow during a computation. The `type` register stores if the rule is of type (a) , (b) , (c) , (d) or (e_s) . The `lhsObject` register stores the object on the left hand side of the rule. The `lhsLabel` register stores the label on the left hand side of the rule. The `rhsObject` register stores the object on the right hand side of the rule. The `rhsAObjects` register stores the multiset of objects generated by the rule.

3.2 There Is a Computation Path That Uses Polynomially Many Equivalence Classes

In Section 3.2 we prove Theorem 3. Before proceeding to this theorem we make an important observation. Suppose we begin at an initial configuration of a recogniser membrane system. Due to non-determinism in the choice of rules and objects, after t timesteps we could be in any one of a large number of possible configurations. However all computations are *confluent*. So if we are only interested in whether the computation accepts or rejects, then it does not matter which computation path we follow.

Theorem 3 asserts that after a polynomial number of timesteps, there is at least one computation path where the number of equivalence classes of a $\text{PMC}_{\mathcal{EAM}_a^0}^S$ system is polynomially bounded. This is shown by proving that there is a computation path where the application of each rule type (a) to (e_s) , in a single timestep, leads to at most an additive polynomial increase in the number of equivalence classes.

Theorem 3. *Given an initial configuration of a $\text{PMC}_{\mathcal{EAM}_a^0}^S$ system Π with $|E_0|$ equivalence classes and $|V|$ distinct object types, then there is a computation path such that at time $t \in \text{poly}(n)$ the number of equivalence classes is $|E_t| = O(|E_0| + t|E_0||V|)$ which is $\text{poly}(n)$.*

Proof. Base case: From Definition 3, $|E_0|$ is bounded above by the (polynomial) number of membranes at time 0. Thus $|E_0| \in \text{poly}(n)$. Each of lemmata 1 to 5 gives an upper bound on the increase in the number of equivalence classes after one timestep for rule types (a) to (e_s) , respectively. Lemma 2 has an additive increase of $|E_0||V|$ and the other four lemmata have an increase of 0. Thus at time 1 there is a computation path where the number of equivalence classes is $|E_1| \leq |E_0| + |E_0||V|$. (From Definitions 1 and 2, $|V| \in \text{poly}(n)$ and $|V|$ is fixed for all t .)

Inductive step: Assume that $|E_i|$, the number of equivalence classes at time i , is polynomial in n . Then, while Lemmata 1 to 5, there exists a computation path where $|E_{i+1}| \leq |E_i| + |E_0||V|$.

After t timesteps we have $|E_t| = O(|E_0| + t|E_0||V|)$, which is polynomial in n if t is. \square

The proofs of the following five lemmata assume some ordering on the set of object types V and on the rules R . For the proof of Lemma 2, we give a specific ordering, however for the other proofs any ordering is valid.

Lemma 1. *Given a configuration C_i of a $\text{PMC}_{\mathcal{EAM}_a^0}^S$ system with $|E|$ equivalence classes. After a single timestep, where only rules of type (a) (object evolution) are applied, there exists a configuration C_{i+1} such that $C_i \vdash C_{i+1}$ and C_{i+1} has $\leq |E|$ equivalence classes.*

Proof. If a type (a) rule is applicable to an object in a membrane in equivalence class e_k , then the rule is also applicable in exactly the same way to all membranes in e_k . Due to non-determinism in the choice of rules and objects, it could be the case that the membranes in e_k evolve differently. However let us assume an ordering on the object types V and on the rules R . We apply the type (a) rules to objects using this ordering. Then all membranes in an equivalence class evolve identically in a timestep, and no new equivalence classes are created. Thus there is a computation path $C_i \vdash C_{i+1}$ where there is no increase in the number of equivalence classes. \square

Observe that type (b) rules have the potential to increase the number of equivalence classes in one timestep by sending different object types into different membranes from the same class. For example, if objects of type v_1 are sent into some of the membranes in an equivalence class, and v_2 objects are sent into the remainder, then we increase the number of equivalence classes by 1. The following lemma gives an additive polynomial upper bound on this increase.

Lemma 2. *Given a configuration C_i of a $\text{PMC}_{\mathcal{EAM}_a^0}^S$ system Π with $|E|$ equivalence classes. Let $|E_0|$ be the number of equivalence classes in the initial configuration of Π . Let $|V|$ be the number of distinct object types in Π . After a single timestep, where only rules of type (b) (incoming objects) are applied, there exists a configuration C_{i+1} such that $C_i \vdash C_{i+1}$ and C_{i+1} has $\leq |E| + |E_0||V|$ equivalence classes.*

Proof. Let e_j be a parent equivalence class, thus e_j represents one membrane (by Definition 3). If the child membranes of e_j are all parent membranes themselves, then the type (b) communication rule occurs without any increase to the number of equivalence classes. The remainder of the proof is concerned with the other case, where e_j contains a non-zero number of equivalence classes of elementary membranes; by the lemma statement this number is $\leq |E|$.

For the remainder of this proof let $V' \subseteq V$ be the set of distinct object types in the membrane defined by e_j for which there are rules in R applicable for this

timestep, let \mathbb{V} be the total number of objects in the membrane defined by e_j , let $E' \subset E$ be the set of equivalence classes that describe the children of the membrane defined by e_j , and let \mathbb{M} be the total number of membranes that are children of the membrane defined by e_j (therefore \mathbb{M} is the total number of membranes in E'). Furthermore we assume that E' is ordered by number of membranes, i.e. we let $E' = (e_1, e_2, \dots, e_{|E'|})$ where $|e_k|$ is the number of membranes in equivalence class e_k and $\forall k, |e_k| \leq |e_{k+1}|$. Similarly we assume that V' is ordered by the number of each object type, i.e. we let $V' = (v_1, v_2, \dots, v_{|V'|})$ where $|v_k|$ is the multiplicity of objects of type v_k and $\forall k, |v_k| \leq |v_{k+1}|$. This ordering ensures that the same deterministic computation path is followed for different instances of the same input configuration. We now consider the two possible cases.

Case 1: $\mathbb{V} < \mathbb{M}$. Table 1 explicitly gives the proof for this case. The \mathbb{M} membranes, beginning with membranes from equivalence class e_1 , each receive one object, beginning with available objects of type v_1 . We continue, following the above orderings on V' and E' , until there are no more objects to communicate. Thus after these type (b) rules have been applied, some number from the E' has received objects, leading to the E' rows in the “Range” column of Table 1. If objects of one distinct type fill up an equivalence class exactly, that class cannot be split into further equivalence classes in that time step. The “Sub-case” column captures all possible (given our ordering on E' and V') ways that objects can fill increasing numbers of equivalence classes. The “Increase EC” column gives the increase in the equivalence classes after one timestep each sub-case. The worst case increase is caused by no equivalence class being exclusively filled up by a distinct object, this means every distinct object communicated will create a new equivalence class. The worst case increase in the total number of equivalence classes after one timestep is $|V'|$.

Case 2: $\mathbb{V} \geq \mathbb{M}$. Table 2 explicitly gives the proof for this case. The \mathbb{M} membranes, beginning with membranes from equivalence class e_1 , each receive one object, beginning with available objects of type v_1 . We continue, following the above orderings on V' and E' , until there are no more available membranes to communicate to. Thus after these type (b) rules have been applied, some number from the V' have been communicated, leading to the V' rows in the “Range” column of Table 2. If membranes from one equivalence class all receive objects of the same distinct type, that class cannot be split into further equivalence classes in that time step. The “Sub-case” column captures all possible (given our ordering on E' and V') ways that equivalence classes can be filled by increasing numbers of distinct objects. The “Increase EC” column gives the increase in the equivalence classes after one timestep each sub-case. The worst case increase is caused by no equivalence class being exclusively filled up by a distinct object, this means every distinct object communicated will create a new equivalence class. The worst case increase in the total number of equivalence classes after one timestep is $|V'| - 1$.

This procedure is iterated over all parent membranes e_j where type (b) rules are applicable, by Definition 3 the number of such parent membranes $\leq |E_0|$. For each parent it is the case that $|V'| \leq |V|$. Thus there is a computation path $C_i \vdash C_{i+1}$ where the increase in the number of equivalence classes is $\leq |E_0||V'| \leq |E_0||V|$. \square

Lemma 3. *Given a configuration C_i of a $\text{PMC}_{\mathcal{E}, \mathcal{A}, \mathcal{M}_{-a}}^S$ system with $|E|$ equivalence classes. After a single timestep, where only rules of type (c) (outgoing objects) are applied, there exists a configuration C_{i+1} such that $C_i \vdash C_{i+1}$ and C_{i+1} has $\leq |E|$ equivalence classes.*

Proof. If a type (c) rule is applicable to an object in a membrane in equivalence class e_k , then the rule is also applicable in exactly the same way to all membranes in e_k . Due to non-determinism in the choice of rules and objects it could be the case that membranes in e_k eject different symbols. However lets assume an ordering on the object types V and on the rules R . We apply the type (c) rules to objects using this ordering. Then all membranes in an equivalence class evolve identically in one (each membrane ejects the same symbol), and so no new equivalence classes are created from e_k . The single parent of all the membranes in e_k is in an equivalence class e_j which, by Definition 3, contains exactly one membrane and so no new equivalence classes are created from e_j .

Thus there is a computation path $C_i \vdash C_{i+1}$ where there is no increase in the number of equivalence classes. \square

Interestingly, dissolution is the easiest rule to handle using our approach. The following lemma actually proves something stronger than the other lemmata: dissolution *never* leads to an increase in the number of equivalence classes.

Lemma 4. *Given a configuration C_i of a $\text{PMC}_{\mathcal{E}, \mathcal{A}, \mathcal{M}_{-a}}^S$ system with $|E|$ equivalence classes. After a single timestep, where only rules of type (d) (membrane dissolution) are applied then for all C_{i+1} , such that $C_i \vdash C_{i+1}$, C_{i+1} has $\leq |E|$ equivalence classes.*

Proof. If there is at least one type (d) rule that is applicable to an object and a membrane in equivalence class e_k , then there is at least one rule that is also applicable to all membranes in e_k . Unlike previous proofs, we do not require an ordering on the objects and rules: all membranes in e_k dissolve and equivalence class e_k no longer exists. The single parent of all the membranes in e_k is in an equivalence class e_j which, by Definition 3, contains exactly one membrane and so no new equivalence classes are created from e_j .

Thus for all C_{i+1} , where $C_i \vdash C_{i+1}$, there is no increase in the number of equivalence classes. \square

Lemma 5. *Given a configuration C_i of a $\text{PMC}_{\mathcal{E}, \mathcal{A}, \mathcal{M}_{-a}}^S$ system with $|E|$ equivalence classes. After a single timestep, where only rules of type (e_s) (symmetric membrane division) are applied, there exists a configuration C_{i+1} such that $C_i \vdash C_{i+1}$ and C_{i+1} has $\leq |E|$ equivalence classes.*

Proof. If a type (e_s) rule is applicable to an object and membrane in equivalence class e_k , then the rule is also applicable in exactly the same way to all membranes in e_k . Due to non-determinism in the choice of rules and objects it could be the case that membranes in e_k divide using and/or creating different symbols. However lets assume an ordering on the object types V and on the rules R . We apply the type (e_s) rules to objects (and membranes) using this ordering. Then all membranes in an equivalence class evolve identically in a timestep (each membrane in e_k divides using the same rule). The number of membranes in e_k doubles, but since each new membrane is identical, no new equivalence classes are created from e_k .

Thus there is a computation path $C_i \vdash C_{i+1}$ where there is no increase in the number of equivalence classes. \square

3.3 Polynomial Time RAM Algorithm

Here we outline a RAM algorithm that simulates the computation of any membrane system of the class $\mathbf{PMC}_{\mathcal{EAM}_a^0}^S$ in polynomial time (in input length n). The algorithm operates on any valid initial configuration and successively applies the evolution rules of the membrane system.

The algorithm makes explicit use of the polynomial size bounded registers described in Section 3.1. It also relies on the confluent nature of recogniser membrane systems and simulates only one of the set of valid computation paths. In particular, using the results from Section 3.2, the algorithm chooses a computation path that uses polynomial space by sorting the membranes, objects and rules of a configuration.

Our `sort` function runs in polynomial time (in input length n) and sorts lists of

- object multisets by object multiplicity, then lexicographically.
- equivalence classes by membrane multiplicity, then by label, and then by objects.
- rules by rules type, matching label, matching object, and then by output object(s).

Since instances of $\mathbf{PMC}_{\mathcal{EAM}_a^0}^S$ are constructed by polynomial time (or logspace) deterministic Turing machines they are at most polynomial size. Also, since all instances of $\mathbf{PMC}_{\mathcal{EAM}_a^0}^S$ run in polynomial time, if our algorithm simulates it with a polynomial time overhead we obtain a polynomial time upper bound.

Our algorithm begins with a configuration of $\mathbf{PMC}_{\mathcal{EAM}_a^0}^S$ (see Algorithm 1). The input configuration is encoded into the registers of the RAM in polynomial time. The rules of the system are sorted and the algorithm then enters a loop. At each iteration all available rules are applied, this simulates a single timestep of the membrane systems computation. The loop terminates when the system ejects a `yes` or `no` object, indicating that the computation has halted. Since all instances of $\mathbf{PMC}_{\mathcal{EAM}_a^0}^S$ run in polynomial time, this loop iterates a polynomial number of times. The total time complexity for running the simulation for time t is $O(t|R||E|^2|V|)$.

Table 1. Increase in the number of equivalence classes (EC) when $\mathbb{V} < \mathbb{M}$. This table is used in the proof of Lemma 2. In the Graphic column, a \square represents an equivalence class and a \circ represents a single membrane in an equivalence class.

Range	Sub-cases	Increase EC	Graphic
$1 \leq \mathbb{V} < e_1 $		$ V' $	
$ e_1 \leq \mathbb{V} < e_1 + e_2 $	$e \in E', e = 1$, every EC in e is filled by objects from a $v \in V'$	$ V' - 1$	
$ e_1 + e_2 \leq \mathbb{V} < e_1 + e_2 + e_3 $	$\nexists e \in E'$, every EC in e is filled by objects from a $v \in V'$	$ V' $	
$ e_1 + e_2 + e_3 \leq \mathbb{V} < e_1 + e_2 + e_3 + e_4 $	$e \in E', e = 2$, every EC in e is filled by objects from a $v \in V'$ $e \in E', e = 1$, every EC in e is filled by objects from a $v \in V'$ $\nexists e \in E'$, every EC in e is filled by objects from a $v \in V'$	$ V' - 2$ $ V' - 1$ $ V' $	- - -
\vdots	\vdots	$ V' - 3$ $ V' - 2$ $ V' - 1$ $ V' $	- - - -
$\sum_{\ell=1}^{ E' } e_\ell \leq \mathbb{V} < \sum_{\ell=1}^{ E' } e_\ell $	$e \in E', e < E' $, every EC in e is filled by objects from a $v \in V'$	$ V' - E' $	\vdots - - - -

Table 2. Increase in the number of equivalence classes (EC) when $\mathbb{V} \geq \mathbb{M}$. This table is used in the proof of Lemma 2. In the Graphic column, a \square represents an equivalence class and a \circ represents a single membrane in an equivalence class.

Range	Sub-cases	Increase EC	Graphic
$0 < \mathbb{M} \leq v_1 $	-	0	
$ v_1 < \mathbb{M} \leq v_1 + v_2 $	$\forall e \in E', \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$	0	
$ v_1 + v_2 < \mathbb{M} \leq v_1 + v_2 + v_3 $	$\forall e \in E' - \{e_{11}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$ $\forall e \in E', \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$	1	
$ v_1 + v_2 + v_3 < \mathbb{M} \leq v_1 + v_2 + v_3 + v_4 $	$\forall e \in E' - \{e_{11}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$ $\forall e \in E' - \{e_{11}, e_{12}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$ $\forall e \in E' - \{e_{11}, e_{12}, e_{13}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$	0	-
\vdots	\vdots	\vdots	-
$ V' - 1 \sum_{\ell=1}^{ V' } v_\ell < \mathbb{M} \leq \sum_{\ell=1}^{ V' } v_\ell $	$\forall e \in E', \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$ $\forall e \in E' - \{e_{11}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$ $\forall e \in E' - \{e_{11}, e_{12}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$ $\forall e \in E' - \{e_{11}, e_{12}, e_{13}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$	0	-
\vdots	\vdots	\vdots	-
$ V' - 1 \sum_{k=1}^{ E' } \{e_{ik}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$	$\forall e \in E' - \bigcup_{k=1}^{ E' } \{e_{ik}\}, \text{ every EC in } e \text{ is filled by objects from } a \ v \in V'$	$ V' - 1$	-

Function. `Apply_a_rule(equivalence_class, object, rule)` applies a single type (*a*) rule to instances of an object in an equivalence class. Total time complexity $O(|V|)$.

Input. `equivalence_class, object, rule`

Output. `equivalence_class` after a type (*a*) rule on an object has been applied

$O(|V|)$ **forall** `resultingObject` *in* `rule.outAobjects` **do**
 multiplicity of `resultingObject` in `equivalence_class` += the multiplicity of
 matching object – the number of `object.used` × the
 `resultingObject.multiplicity` ;
 used number of `resultingObject` in the `equivalence_class` += the multiplicity
 of `resultingObject` × `object.multiplicity` – `object.used` ;
 decrement `object.multiplicity` ;
 set `object.used` = `object.multiplicity` ;

Function. `Apply_c_rule(equivalence_class, object, rule)` applies a single rule of type (*c*) to a membrane. Total time complexity $O(1)$.

Input. `equivalence_class`

Output. `equivalence_class` after a (*c*) rule have been applied

decrement `object.multiplicity` ;

increment `object.multiplicity` in `equivalence_class.parent` of the generated object;

increment `object.used` in `equivalence_class.parent` of the generated object;

increment `equivalence_class.used` ;

Function. `Apply_d_rule(equivalence_class, object, rule)`. This function applies dissolution rules to an equivalence class. It calculates the total number of each object in the equivalence class and adds it to the parent. It also copies the child membranes from the dissolving membrane and adds them to the parents child list. The total time complexity is $O(|V|)$.

Input. `equivalence_class`

Output. `equivalence_class` after (*d*) rule has been applied

decrement `object.multiplicity` ;

increment `object.multiplicity` in `equivalence_class.parent` from the rule;

increment `object.used` in `equivalence_class.parent` from the rule;

`/* move contents of the dissolved membrane to its parent */`

$O(|V|)$ **forall** `move_object` *in* `equivalence_class` *objects* **do**
 add `move_object.multiplicity` × `equivalence_class.multiplicity` to
 `move_object.multiplicity` in `equivalence_class.parent` ;
 add `move_object.used` × `equivalence_class.multiplicity` to `move_object.used` in
 `equivalence_class.parent` ;
 `move_object.multiplicity` ← 0;
 `move_object.used` ← 0;
 `equivalence_class.parent.children` ← `equivalence_class.parent.children` ∪
 `equivalence_class.children` ;
 `equivalence_class.multiplicity` ← 0;
 `equivalence_class` ← ∅;

Function. `Apply_es_rule`(*equivalence_class*, *object*, *rule*). Applies a single rule of type (e_s) to a membrane. Total time complexity $O(1)$.

Input. `equivalence_class`

Output. `equivalence_class` after (e_s) rule has been applied

decrement `object.multiplicity` ;

increment `object.multiplicity` from the rule;

increment `object.used` from the rule;

increment `equivalence_class.used` ;

`equivalence_class.multiplicity` \leftarrow `equivalence_class.multiplicity` \times 2;

Function. `Apply_b_rules`(*b_equivalence_classes*, *b_objects*, *b_rules*). Total time complexity $O(|V||E|)$.

Input. `membrane`

Output. `membrane` after (b) rules have been applied

`b_objects_sorted` \leftarrow `sort` (`b_objects`);

`b_equivalence_classes_sorted` \leftarrow `sort` (`b_equivalence_classes`);

$O(|V|)$ **forall** `object` *in* `b_objects_sorted` **do**

$O(|E|)$ **forall** `equivalence_class` *in* `b_equivalence_classes_sorted` **do**

if `object.multiplicity` $<$ `equivalence_class.multiplicity` **then**

 copy `equivalence_class` to `new_equiv_class` ;

 subtract `object.multiplicity` from `new_equiv_class.multiplicity` ;

`equivalence_class.multiplicity` \leftarrow `object.multiplicity` ;

`equivalence_class.used` \leftarrow `equivalence_class.multiplicity` ;

 increment `equivalence_class.object.multiplicity` ;

 increment `equivalence_class.object.used` ;

else if `object.multiplicity` \geq `equivalence_class.multiplicity` **then**

 increment `equivalence_class.object.multiplicity` ;

 increment `equivalence_class.object.used` ;

`equivalence_class.used` \leftarrow `equivalence_class.multiplicity` ;

 subtract `equivalence_class.multiplicity` from `object.multiplicity` ;

4 Conclusion

We have given a \mathbf{P} upper bound on the computational power of one of a number of membrane systems that fall under the so-called \mathbf{P} -conjecture. In particular we consider a variant of membrane systems that allows only symmetric division. This variant can easily generate an exponential number of membranes and objects in polynomial time. We restricted the uniformity condition to logspace, making the \mathbf{P} lower bound more meaningful. Our technique relies on being able to find computation paths that use only polynomial space in polynomial time. It seems that this technique is not naively applicable to the case of asymmetric division: it is possible to find examples where all computation paths are forced to use an exponential number of equivalence classes.

Furthermore the result seems interesting since before before now, all models without dissolution rules were upper bounded by \mathbf{P} and all those with dissolution

rules characterised **PSPACE**. This result shows that despite having dissolution rules, by using only symmetric elementary division we restrict the system so that it does not create exponential space on all computation paths in polynomial time.

Acknowledgements

Niall Murphy is supported by the Irish Research Council for Science, Engineering and Technology. Damien Woods is supported by Science Foundation Ireland grant number 04/IN3/1524. We give a special thanks to Thomas J. Naughton for interesting comments and ideas.

References

1. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: *Molecular Biology of the Cell*, 4th edn. Garland Science, New York (2002)
2. Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution to QSAT using polarization-less active membranes. In: Gutiérrez-Naranjo, M.A., Păun, G., Riscos-Núñez, A., Romero-Campero, F.J. (eds.) *Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30–February 3, 2006, vol. I, pp. 29–40. Fénix Editora (2006)
3. Balcázar, J.L., Diaz, J., Gabarró, J.: *Structural complexity I*, 2nd edn. Springer, Berlin (1988)
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* 83(7), 593–611 (2006)
5. Ladner, R.E.: The circuit value problem is log space complete for P. *SIGACT News* 7(1), 18–20 (1975)
6. Păun, G.: P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics* 6(1), 75–90 (2001)
7. Păun, G.: *Membrane Computing. An Introduction*. Springer, Berlin (2002)
8. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. *Natural Computing* 2(3), 265–285 (2003)
9. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences* 73(1), 137–152 (2007)