Active Membrane Systems Without Charges and Using Only Symmetric Elementary Division Characterise P

Niall Murphy[1] and Damien Woods[2]

[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`

Summary. 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 P. This result characterises the power of a class of membrane systems that fall under the so called P conjecture for membrane systems.

1 Introduction

The **P**-conjecture states that recogniser membranes systems with division rules (active membranes [5]), but without charges, characterise **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 **PSPACE** [2, 8]. In this setting, using dissolution rules allows us to jump from **P** to **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 **P**.

Using our restriction insists that the two membranes that result from an elementary division rule must be identical. This models the usual 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 **P**. All recogniser membrane systems with division rules are upper bounded by **PSPACE** [8]. In this paper we show that systems with symmetric elementary division and without charges are upper bounded by **P**. From an algorithmic point of view, this result 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 [5, 6], and Sosík and Rodríguez-Patón [8].

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 $\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$ 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. $w_1, \ldots, w_m$ *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 form 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 can non-deterministically choose from various sequences of configurations, but all of them must lead to the same result.

2.2 Complexity classes

Complexity classes have been defined for membrane systems [7]. Consider a decision problem X, i.e. a set of instances $\{x_1, x_2, \ldots\}$ over some finite alphabet such that to each x_1 there is an unique answer "yes" or "no". We consider a *family* of membrane systems to solve each decision problem so that 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 (Uniform families of membrane systems). *Let $\mathcal{D}$ be a class of membrane systems and let $f : \mathbb{N} \to \mathbb{N}$ be a total function. The class of problems solved by uniform families of membrane systems of type $\mathcal{D}$ in time f, denoted by $\mathbf{MC}_{\mathcal{D}}(f)$, contains all problems X such that:*

- *There exists a* uniform *family of membrane systems, $\Pi_X = (\Pi_X(1); \Pi_X(2); \ldots)$ 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 an 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.*

Semi-uniform families of membrane systems $\Pi_X = (\Pi_X(x_1); \Pi_X(x_2); \ldots)$ 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 $\mathbf{MC}^S_{\mathcal{D}}(f)$. Obviously, $\mathbf{MC}_{\mathcal{D}}(f) \subseteq \mathbf{MC}^S_{\mathcal{D}}(f)$ for a given class $\mathcal{D}$ and a complexity [3] function f.

We denote by

$$\mathbf{PMC}_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} \mathbf{MC}_{\mathcal{D}}(O(n^k)),\ \mathbf{PMC}^S_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} \mathbf{MC}^S_{\mathcal{D}}(O(n^k))$$

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 $\mathcal{AM}^0_{-a}$ (respectively, $\mathcal{AM}^0_{+a}$) the class of all recogniser membrane systems with active membranes without charges and without asymmetric division (respectively, with asymmetric division). We denote by $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$ 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 poly(n) be the set of polynomial (complexity) functions of n.

3 An upper bound on $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$

In this section we give an upper bound of $\mathbf{P}$ on the membrane class $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$. 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 and an informal description of our contribution.

Definition 3 (Equivilance 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 part 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 $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$ 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 transfered to the parent (already an equivalence class).
- Type (e_s) rules do not increase the number of equivalence classes, the number of membranes in the existing equivalence classes simply increases.

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 (in binary) of that 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 1. $\mathbf{PMC}^{S}_{\mathcal{EAM}^0_{-a}} \subseteq \mathbf{P}$

The proof is in the remainder of this section.

3.1 Structure of RAM Registers

Our RAM uses a number of binary registers that is a polynomial (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 1).

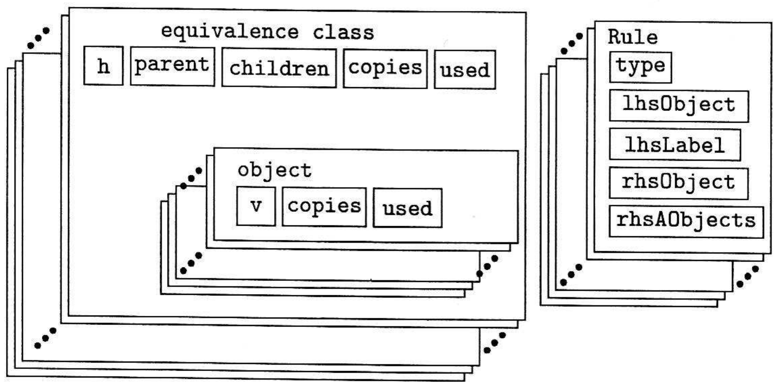

Fig. 1. A representation of our polynomial sized registers as a data structure.

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 `copies` register 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 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 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 `used` $\leq$ `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 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 `children` register references all of the child equivalence classes of e_k at depth one. Its size is bounded by poly(n) via Theorem 2.

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 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 using polynomially many equivalence classes.

In Section 3.2 we prove Theorem 2. 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 2 asserts that after a polynomial number of timesteps, there is at least one computation path where the number of equivalence classes of a $\mathbf{PMC}^{S}_{\mathcal{EAM}^0_{-a}}$ 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 2. *Given an initial configuration of a* $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ *system* Π *with* $|E_0|$ *equivalence classes and* $|V|$ *distinct object types, then there is a computation path such that at time* $t \in poly(n)$ *the number of equivalence classes is* $|E_t| = O(|E_0| + t|E_0||V|)$ *which is 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. □

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* $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ *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. □

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 get an increase of 1 in the number of equivalence classes. The following lemma gives an additive polynomial upper bound on this increase.

Lemma 2. *Given a configuration C_i of a $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$ 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, let $\mathbb{V}$ be the total number of objects in the membrane defined by e_j, let $E' \subseteq E$ 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 number of membranes in E'). Furthermore we assume that E' is ordered by number of membranes, i.e. we let $E' = (e_1, e_2, \ldots, e_{|E'|})$ 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 $V' = (v_1, v_2, \ldots, v_{|V'|})$ where $|v_k|$ is the multiplicity of objects of type v_k and $\forall k$, $|v_k| \leq |v_{k+1}|$. We divide the proof into two cases.

Case 1: $\mathbb{V} < \mathbb{M}$. Table 1 gives the proof for this case. The "Range" column gives a series of ranges for $\mathbb{V}$ with respect to the numbers of membranes in the elements of E'. The "Total EC" gives the total number of equivalence classes if we place symbols into membranes according to the orderings given on E' and V'. The "Increase EC" column gives the increase in the equivalence classes after one timestep (i.e. $|E'|$ subtracted from the "Total EC" value). Although we omit the tedious details, it is not difficult to show that by using the above ordering and going through the various sub-cases, the worst case for the total number of equivalence classes after one timestep is $|E'| + |V'|$.

Case 2: $\mathbb{V} \geq \mathbb{M}$. Case 2 is proved using Table 2. Again we omit the details, however it is not difficult to show that by using the above ordering and going through the various sub cases, the worst case for the total number of equivalence classes after one timestep is $|E'| + |V'|$.

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 $\leq |E_0||V'| \leq |E_0||V|$. □

Lemma 3. *Given a configuration C_i of a $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$ 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. □

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 $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$ 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. □

Lemma 5. *Given a configuration C_i of a $\mathbf{PMC}^S_{\mathcal{EAM}^0_{-a}}$ 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. $\square$

Range	Total EC	Increase EC
$1 \le \mathbb{V} < \lvert e_1\rvert$	$\lvert E'\rvert + \lvert V'\rvert$	$\lvert V'\rvert$
$\lvert e_1\rvert \le \mathbb{V} < \lvert e_1\rvert + \lvert e_2\rvert$	$\lvert E'\rvert + \lvert V'\rvert - 1$	$\lvert V'\rvert - 1$
	$\lvert E'\rvert + \lvert V'\rvert$	$\lvert V'\rvert$
$\lvert e_1\rvert + \lvert e_2\rvert \le \mathbb{V} < \lvert e_1\rvert + \lvert e_2\rvert + \lvert e_3\rvert$	$\lvert E'\rvert + \lvert V'\rvert - 2$	$\lvert V'\rvert - 2$
	$\lvert E'\rvert + \lvert V'\rvert - 1$	$\lvert V'\rvert - 1$
	$\lvert E'\rvert + \lvert V'\rvert$	$\lvert V'\rvert$
$\vdots$	$\vdots$	$\vdots$
$\sum_{\ell=1}^{\lvert E'\rvert-1} \lvert e_\ell\rvert \le \mathbb{V} < \sum_{\ell=1}^{\lvert E'\rvert} \lvert e_\ell\rvert$	$\lvert E'\rvert + \lvert V'\rvert - (\lvert E'\rvert - 1) + 1$	$\lvert V'\rvert - (\lvert E'\rvert - 1) + 1$
	$\lvert E'\rvert + \lvert V'\rvert$	$\lvert V'\rvert$

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.

Range	Sub-case	Total EC	Increase EC
$0 < \mathbb{M} \le \lvert v_1\rvert$	-	$\lvert E'\rvert$	0
$\lvert v_1\rvert < \mathbb{M} \le \lvert v_1\rvert + \lvert v_2\rvert$	$\exists\ m$ s.t. $1 \le m < \lvert E'\rvert$, $\sum_{\ell 1}^{m} \lvert e_\ell\rvert = \lvert v_1\rvert$, $\lvert v_2\rvert \ge \mathbb{M} - \lvert v_1\rvert$	$\lvert E'\rvert$	0
	$\nexists\ m$ s.t. $1 \le m < \lvert E'\rvert$, $\sum_{\ell 1}^{m} \lvert e_\ell\rvert = \lvert v_1\rvert$, $\lvert v_2\rvert \ge \mathbb{M} - \lvert v_1\rvert$	$\lvert E'\rvert + 1$	1
$\lvert v_1\rvert + \lvert v_2\rvert < \mathbb{M} \le \lvert v_1\rvert + \lvert v_2\rvert + \lvert v_3\rvert$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\sum_{\ell=1}^{\lvert V'\rvert-1} \lvert v_\ell\rvert < \mathbb{M} \le \sum_{\ell=1}^{\lvert V'\rvert} \lvert v_\ell\rvert$	$\vdots$	$\lvert E'\rvert + \lvert V'\rvert - 1$	$\lvert V'\rvert - 1$

Table 2. Increase in the number of equivalence classes (EC) when $\mathbb{V} \ge \mathbb{M}$. This table is used in the proof of Lemma 2.

3.3 RAM Algorithm

Here we outline a RAM algorithm that simulates the computation of any membrane system of the class $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ in polynomial time (in input length n). The algorithm operates on any initial configuration and successively applies the evolution rules of the membrane system. It orders all lists of objects, rules and equivalence classes which results in a deterministic computation path that never uses more than polynomial space. 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 at most a polynomial number of equivalence classes.

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 $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ are constructed by polynomial time deterministic Turing machines they are of polynomial size. Also, since all instances of $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ 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 $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ (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 which 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 $\mathbf{PMC}^{S}_{\mathcal{EAM}^{0}_{-a}}$ run in polynomial time, this loop iterates a polynomial number of times.

At each iteration the algorithm iterates through all equivalence classes and applies all applicable rules to it. The worst case time complexity for each function is given. The total time complexity for running the simulation for time t is $O(t|R||E|^2|V|)$.

Algorithm 1: The is the main body of the membrane simulation algorithm. The systems rules are sorted and then applied to the membrane system at each timestep until the system accepts or rejects its input.

```
                    Input: a configuration of $\mathbf{PMC}^{S}_{\mathcal{EAM}^0_{-a}}$
                    Output: The deciding configuration of the system
                    Initialise registers with input system;
                    sortedRules ← sort(rules);
$O(t)$              repeat
                        /* evolve the membrane system one step                    */
$O(|E|)$                forall equivalence_class in membraneSystem do
$O(|R||E||V|)$              ApplyRules(equivalence_class);
                        end
                    until yes or no object is in skin membrane ;
```

Function `ApplyRules(`*equivalence_class*`)` Applies all applicable rules for an equivalence class for one timestep

```
                    Input: equivalence_class
                    Output: equivalence_class after one timestep of computation
                    b_rules ← $\emptyset$;
                    b_ecs ← $\emptyset$;
                    b_objs ← $\emptyset$;
$O(|R|)$            forall rule in sortedRules do
                        if rule.label matches equivalence_class.label and rule is not type (b) then
$O(|V|)$                    forall object in sortedObjects do
                                if not all copies of object have been used then
                                    if rule is type (a) then
$O(|V|)$                                Apply_a_rule(equivalence_class, object, rule);
                                    else if rule is type (c) then
$O(1)$                                  Apply_c_rule(equivalence_class, object, rule);
                                    else if rule is type (d) then
$O(|V|)$                                Apply_d_rule(equivalence_class, object, rule);
                                    else if rule is type ($e_s$) then
$O(1)$                                  Apply_e_rule(equivalence_class, object, rule);
                                    end
                                end
                            end
                        end
                        if rule is type (b) then
$O(|E|)$                    forall child_ec in equivalence_class do
                                if child_ec.label = rule.lhsLabel and object.used $\geq 1$ then
                                    append child_ec to b_ecs ;
                                    append object to b_objs ;
$O(|V||E|)$                         Apply_b_rule(b_ecs, b_objs, rule)
                                end
                            end
                        end
                    end
$O(|V| \times |E|)$ reset all used counters to 0;
```

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 $\times$ the resultingObject.multiplicity ;
      used number of resultingObject in the equivalence_class $+=$ the multiplicity of
      resultingObject $\times$ object.multiplicity $-$ object.used ;
  end
  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 $\times$ equivalence_class.multiplicity to
      move_object.multiplicity in equivalence_class.parent ;
      add move_object.used $\times$ equivalence_class.multiplicity to move_object.used in
      equivalence_class.parent ;
      move_object.multiplicity $\leftarrow 0$;
      move_object.used $\leftarrow 0$;
  end
  equivalence_class.parent.children $\leftarrow$ equivalence_class.parent.children $\cup$
  equivalence_class.children ;
  equivalence_class.multiplicity $\leftarrow 0$;
  equivalence_class $\leftarrow \emptyset$;
```

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 ;
                  end
                  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 ;
                  end
              end
          end
```

4 Conclusion

We have given a **P** upper bound on the computational power of one of a number of membrane systems that fall under the so-called **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. 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 naïvely 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 **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. Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell.* Garland Science, New York, fourth edition, 2002.
2. Artiom Alhazov and Mario de Jesús Pérez-Jiménez. Uniform solution to QSAT using polarizationless active membranes. In Miguel Angel Gutiérrez-Naranjo, Gheorghe Paun, and Agustín Riscos-NúñezandFrancisco José Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February3,2006. Volume I*, pages 29–40. Fénix Editora, 2006.
3. José Luis Balcázar, Josep Diaz, and Joaquim Gabarró. *Structural complexity I.* Springer-Verlag New York, Inc., New York, NY, USA, 2nd edition, 1988.
4. Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, and Francisco J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics*, 83(7):593–611, 2006.
5. Gheorghe Păun. P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001. and CDMTCS TR 102, Univ. of Auckland, 1999 (www.cs. auckland.ac.nz/CDMTCS).
6. Gheorghe Păun. *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
7. Mario J. Pérez-Jiménez, Alvaro Romero-Jiménez, and Fernando Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–285, August 2003.
8. Petr. Sosík and Alfonso Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73(1):137–152, 2007.