First steps towards linking membrane depth and the Polynomial Hierarchy

Antonio E. Porreca[1] and Niall Murphy[2]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca, Italy
porreca@disco.unimib.it
[2] Department of Computer Science
National University of Ireland Maynooth, Ireland
nmurphy@cs.nuim.ie

Abstract. In this paper we take the first steps in studying possible connections between non-elementary division with limited membrane depth and the levels of the Polynomial Hierarchy. We present a uniform family with a membrane structure of depth $d+1$ that solves a problem complete for level d of the Polynomial Hierarchy.

1 Introduction

Active membrane systems without charges are an extremely interesting group of models to study from the computational complexity point of view. Forbidding the use of a single rule type yields dramatic differences in computing power of these models. For example, it is known that systems with strong non-elementary division characterise PSPACE [1,14], but when dissolution is forbidden these systems can solve at most problems in NL in the AC^0-semi-uniform case [7], and at most AC^0 in the AC^0-uniform case [8]. Since $\mathsf{AC}^0 \subsetneq \mathsf{NL} \subsetneq \mathsf{PSPACE}$ it seems these rules somehow capture different aspects of computation.

In this report we present our first step towards a better understanding of the difference between P and PSPACE in terms of membrane systems. We suspect that the depth of a membrane system combined with non-elementary division is the key to this difference. Non-elementary division an operation where a membrane divides and all child membranes (and their child membranes etc.) get copied. There are two varieties of non-elementary division, "strong" which is triggered by membranes, and "weak" which is triggered by objects. (The labels "weak" and "strong" have nothing to do with the power of these rules.) Elementary division is where division is only permitted on membranes that do not have child membranes, and can be thought of as non-elementary division on structure of depth of 0.

- Systems with *strong* non-elementary division and polynomial membrane depth are known to characterise PSPACE [1,14].
- Systems with *weak* non-elementary division and polynomial depth can solve at least all of $\mathsf{NP} \cup \mathsf{coNP}$ [3].

– Systems with elementary division (non-elementary division on depth 0) are believed to characterise P (see [6,16] for some partial results, this is an open problem known as the P-conjecture [5,12]).

This has lead us to an intriguing hypothesis: that by using non-elementary division rules and by limiting the depth of the membrane structure we can characterise each level of the polynomial hierarchy from P to PSPACE. If this hypothesis is correct it will help us understand how membrane division contributes in the jump from P to PSPACE and will help resolve the P-conjecture.

The idea that increasing the depth of the membrane structure also increases the computing power of the systems is also consistent with another recent result. Porreca et al. [13] show that (if no time limit is imposed) increasing the depth of active membrane systems using only communication and strong non-elementary division rules permits the systems to solve exponentially harder problems.

This report presents our first steps to proving a link between non-elementary division for a specific membrane depth and the polynomial hierarchy. We show that logspace uniform families of membrane system with a structure of depth $d+1$ can solve problems complete for the d^{th} level of the polynomial hierarchy. In other words, adding a further level of depth gives us the power of an oracle for the previous level of the hierarchy.

In future work we hope to find a corresponding upper-bound where a Turing machine with d alternations can simulate a membrane system with non-elementary division and depth $d+1$.

2 Definitions for membrane systems

In this section we define membrane systems and some complexity classes, these definitions are based on those from [4,11,9,10,14]. The set of all multisets over a set A is denoted $\mathrm{MS}(A)$.

2.1 Active membrane systems

Active membrane systems are a class of membrane systems with membrane division rules. In this paper we use division rules that can act on elementary membranes, which are membranes that do not contain other membranes (i.e. leaves in the membrane structure), or non-elementary membranes, membranes that do contain other membranes.

Definition 1. *An active membrane system without charges is a 6-tuple* $\Pi = (O, \mu, M, H, L, R)$ *where,*

1. O *is the alphabet of objects;*
2. $\mu = (V_\mu, E_\mu)$ *is a tree representing the membrane structure, where* $V_\mu \subseteq \mathbb{N}$ *and* $E_\mu \subsetneq V_\mu \times V_\mu$*;*
3. $M : V_\mu \to \mathrm{MS}(O)$ *maps membranes to their multisets;*
4. H *is the finite set of membrane labels;*

5. $L : V_\mu \to H$ *maps membranes to their labels;*
6. R *is a finite set of developmental rules of the following types (where* $a, b, c \in O$ *and* $u \in \mathrm{MS}(O)$*,* $h \in H$*):*
   (a) $[\, a \to u\,]_h$ *(object evolution),*
   (b) $a[\,]_h \to [\, b]_h$ *(communication in),*
   (c) $[\, a]_h \to [\,]_h\, b$ *(communication out),*
   (d) $[\, a]_h \to b$ *(membrane dissolution),*
   ($\mathrm{e_w}$) $[\, a]_h \to [\, b]_h\, [\, c]_h$*, (weak non-elementary membrane division).*

The vertices V_μ of the membrane structure tree μ are the individual membranes of the system. The parent of all membranes in the system (the root vertex in μ) is called the "skin" and has label $0 \in H$. A *configuration* $\mathcal{C}$ of a membrane system is a tuple (μ, M, L) whose elements are defined in Definition 1. A *permissible encoding* of a membrane system $\langle \Pi \rangle$, or a configuration $\langle \mathcal{C} \rangle$, encodes all multisets in a unary manner. For example, a multiset must be specified in the format $[\, a, a, a, b, b\,]$, rather than a^3b^2, in order to ensure that at most a polynomial number of objects are initially encoded in a system.

The rules in the set R are applied to a configuration according to the following principles:

- All the rules are applied in a *maximally parallel manner*. In each timestep, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do so.
- If a membrane labelled h is divided by a rule of type (e) and there are objects in this membrane which evolve via rules of type (a), then we assume that first the evolution (a) rules are used, and then the division (e) rules. This process takes only one step.
- The rules associated with membranes labelled with h are used for membranes with that label. In each timestep, a membrane can be the subject of only one rule of types (b)–($\mathrm{e_w}$).

A *computation* of a membrane system is a sequence of configurations such that each configuration (except the initial one) is obtained from the previous one by a transition (one-step maximally parallel application of the rules). Membrane systems are non-deterministic, therefore on a given input there are multiple possible computations. A computation that reaches a configuration where no more rules are applicable is called a *halting computation.*

Definition 2. *A* recogniser membrane system *is a membrane system* Π *such that:*

1. *all computations halt,*
2. $\mathtt{yes}, \mathtt{no} \in O$*,*
3. *the object* `yes` *or object* `no` *(but not both) appear in the multiset of the membrane with label* 0 *(the skin),*
4. *and this happens only in the halting configuration.*

2.2 Complexity classes

A problem is a set $X = \{x_1, x_2, \ldots\} \subseteq \Sigma^*$ and its complement is $\overline{X} = \Sigma^* - X$ where Σ is some finite alphabet. We say that a *family* $\mathbf{\Pi}$ of membrane systems recognises a problem if for each $x \in \Sigma^*$ there is some $\Pi \in \mathbf{\Pi}$ that decides if $x \in X$. We denote by $|x| = n$ the length of any instance $x \in \Sigma^*$. Throughout this paper, AC^0 circuits are $\mathsf{DLOGTIME}$-uniform, polynomial sized (in input length n), constant depth, circuits with AND, OR and NOT gates, and unbounded fan-in [2]. FP, FL, and FAC^0 are the classes of functions that are respectively computable by deterministic Turing Machines in polynomial time, by deterministic Turing machines using logarithmic space, and by $\mathsf{DLOGTIME}$-uniform polynomial-sized alternating circuits with unbounded fan-in and constant depth.

Definition 3. *Let $\mathcal{R}$ be a class of recogniser membrane systems and let $t : \mathbb{N} \to \mathbb{N}$ be a total function. Let E and F be classes of functions. The class of* problems solved by a (E, F)-uniform family of membrane systems *of type $\mathcal{R}$ in time t, denoted (E, F)–$\mathsf{MC}_{\mathcal{R}}(t)$, contains all problems X such that:*

- *There exists an* F-uniform *family of membrane systems, $\mathbf{\Pi} = \{\Pi_1, \Pi_2, \ldots\}$ of type $\mathcal{R}$: that is, there exists a function $f \in \mathsf{F}$, $f : \{1\}^* \to \mathbf{\Pi}$ such that $f(1^n) = \Pi_n$, where $|x| = n$.*
- *There exists an input encoding function $e \in \mathsf{E}$, $e : X \cup \overline{X} \to \mathrm{MS}(I)$ such that $e(x)$ is the input multiset, which is placed in a specific input membrane of Π_n, where $|x| = n$ and $I \subsetneq O$ is the set of input objects.*
- *$\mathbf{\Pi}$ is t-efficient: Π_n always halts in at most $t(n)$ steps.*
- *The family $\mathbf{\Pi}$ is* sound *with respect to (X, e, f); that is if there is an accepting computation of the system $\Pi_{|x|}$ on input multiset $e(x)$ then $x \in X$.*
- *The family $\mathbf{\Pi}$ is* complete *with respect to (X, e, f); that is, for each input $x \in X$, then every computation of the system $\Pi_{|x|}$ on input multiset $e(x)$ is accepting.*

We define the set of languages decided by a uniform family of membrane systems in polynomial time to be

$$(\mathsf{E}, \mathsf{F})\text{–}\mathsf{PMC}_{\mathcal{R}} = \bigcup_{k \in \mathbb{N}} (\mathsf{E}, \mathsf{F})\text{–}\mathsf{MC}_{\mathcal{R}}(n^k)$$

When the symbols E and F are replaced by complexity class names such as AC^0, L or P it means that the uniformity conditions under consideration are in the function versions of these classes. For example, if we let $\mathsf{E} = \mathsf{F} = \mathsf{AC}^0$ then we mean that the functions $e \in \mathsf{E}$ and $f \in \mathsf{F}$ are computable in uniform FAC^0 and we say we have an AC^0-uniform family.

Let $\mathcal{AM}^0_{+wne}$ denote the class of membrane systems that obey Definition 2, and Definition 1. Thus $(\mathsf{AC}^0, \mathsf{L})$–$\mathsf{PMC}_{\mathcal{AM}^0_{+wne}}$ denotes the class of problems solvable by L-uniform families of active membrane systems without charges in polynomial time with weak non-elementary division rules where the input is encoded using a function in FAC^0.

Remark 4. A membrane system is said to be *confluent* if it is both sound and complete. That is, a membrane system Π is *confluent* if all computations of Π with the same input x (properly encoded) give the same result; either always "accepts" or else always "rejects".

In a confluent membrane system, given a fixed initial configuration, the system non-deterministically chooses one from a number of valid computations (configuration sequences), but all of these computations must lead to the same result, either all accepting or all rejecting.

3 Polynomial Hierarchy

A well know extension for models of computation is to augment them with an "oracle", that is, the ability to solve certain decision problems in a single timestep. An oracle machine is a machine with access to a special oracle tape that is used to make queries of the form "is $q \in L$" for some language L. By the notation M^{C} we mean the set of problems solved by machines characterising the complexity class M having access to an oracle for a language L in the complexity class C. For instance, P^{NP} is the class of problems solved by deterministic Turing machines working in polynomial time and using an oracle for a problem in NP.

Definition 5 (The Polynomial Hierarchy). *The first level of the hierarchy is* $\Delta_0\mathsf{P} = \Sigma_0\mathsf{P} = \Pi_0\mathsf{P} = \mathsf{P}$. *Then each level of the hierarchy is defined for all* $i \geq 0$,

$$\begin{aligned}\Delta_{i+1}\mathsf{P} &= \mathsf{P}^{\Sigma_i\mathsf{P}}\\ \Sigma_{i+1}\mathsf{P} &= \mathsf{NP}^{\Sigma_i\mathsf{P}}\\ \Pi_{i+1}\mathsf{P} &= \mathsf{coNP}^{\Sigma_i\mathsf{P}}\end{aligned}$$

We define the cumulative polynomial hierarchy to be the class $\mathsf{PH} = \cup_{i\geq 0}\Sigma_i\mathsf{P}$.

Note that $\Sigma_1\mathsf{P} = \mathsf{NP}$ and $\Pi_1\mathsf{P} = \mathsf{coNP}$. The hierarchy possesses the following inclusion structure:

$$\Sigma_i\mathsf{P} \cup \Pi_i\mathsf{P} \subseteq \Delta_{i+1}\mathsf{P} \subseteq \Sigma_{i+1}\mathsf{P} \cap \Pi_{i+1}\mathsf{P}, \text{ for all } i \geq 0.$$

Each level of the polynomial hierarchy has its own complete problems.

Problem 6 (Boolean Satisfiability with i Quantifiers (QSAT_d)). Given a Boolean formula φ, and a partitioning of the variables of φ into d sets $X_1, \ldots, X_d$. Is there a partial truth assignment for the variables in X_1 such that for all the partial truth assignment for the variables in X_2 such that there is a partial truth assignments for the variables in X_3, and so on up to X_d, such that φ is satisfied by the overall truth assignment?

Lemma 7. QSAT_d *is complete for the class* $\Sigma_d\mathsf{P}$ *[15].*

We have defined QSAT_i so that the odd quantifiers are existential. Without loss of generality we can assume that the expression φ is always in conjunctive normal form with three literals in each clause (3CNF). We refer to this restriction of QSAT_d as $\Sigma_d\mathsf{SAT}$ for short. If the odd numbered sets of variables are universal and φ in disjunctive normal form with 3 variables in each clause (3DNF) we refer to it as $\Pi_d\mathsf{SAT}$.

4 Description of a uniform family to solve $\Sigma_d\mathsf{SAT}$

In this section we provide some details of a uniform family of active membrane systems with a membrane structure $d+1$ levels deep which decides instances of $\Sigma_d\mathsf{SAT}$. The uniform family implements the following straightforward quantifier elimination algorithm to establish the validity of quantified Boolean formulas. We first describe how the algorithm works on QSAT, then show how it is affected by considering the restriction $\Sigma_d\mathsf{SAT}$. The algorithm works by reducing the problem to the evaluation of quantifier-free and variable-free expressions. This method is based on the following simple observations:

$$\begin{aligned} \forall x\psi(x) &\iff \psi(0) \wedge \psi(1) \\ \exists x\psi(x) &\iff \psi(0) \vee \psi(1). \end{aligned}$$

By applying these equivalences recursively to an instance of $\Sigma_d\mathsf{SAT}$, the quantifiers can be eliminated one by one. We then evaluate the final fully expanded expression to obtain the result. This evaluation can be computed in polynomial time with respect to the size of the expression; note however, that the expression to evaluate is exponentially larger than the input, since eliminating a quantifier doubles its size.

This quantifier elimination algorithm is needlessly inefficient when executed sequentially: since QSAT is in PSPACE, this problem can be solved in exponentially less space. However, the algorithm can be made to run in polynomial time if an exponential number of processors are available. The two sub-formulas resulting from the elimination of a quantifier can be evaluated *independently*, and their truth values conjuncted or disjuncted (according to the specific quantifier) only in the last step. This is equivalent to evaluating the formula under every possible assignment to the variables, then feeding the results into an exponentially-sized Boolean circuit C which forms a complete binary tree (the same form as the recursion tree of the quantifier elimination algorithm, or equivalently, as the parse tree of the resulting Boolean expression) where the nodes of depth i are $\wedge$-gates (resp., $\vee$-gates) if variable x_{i+1} is universally (resp., existentially) quantified. Notice that the depth of this circuit is linear with respect to the number of variables.

When the number of alternations of quantifiers is bounded by a constant d (as in the problems $\Sigma_d\mathsf{SAT}$ and $\Pi_d\mathsf{SAT}$), and if unbounded fan-in gates are available, the circuit used to combine the results of the evaluation of the formula can be reduced to constant depth d. Indeed, a sequence of k consecutive quantifiers can

be eliminated simultaneously, as long as they are all universal or all existential, and the values of the 2^k resulting sub-formulas fed into a single $\wedge$- or $\vee$-gate, thus increasing the depth of the circuit just by one. In symbols:

$$\forall x_1 \cdots \forall x_k \varphi(x_1, \ldots, x_k) \iff \bigwedge_{(x_1,\ldots,x_k)\in\{0,1\}^k} \varphi(x_1, \ldots, x_k)$$
$$\exists x_1 \cdots \exists x_k \varphi(x_1, \ldots, x_k) \iff \bigvee_{(x_1,\ldots,x_k)\in\{0,1\}^k} \varphi(x_1, \ldots, x_k).$$

4.1 Encoding of Σ_dSAT instances

We specify that instances of Σ_dSAT are encoded as follows.

We encode which variables are bound by which quantifiers in a binary matrix Q with m rows and m columns. Each column represents one of the m variables of the formula. There are a maximum of m rows since at most $d \leq m$ quantifiers are possible for each instance. The elements of Q are defined as follows:

$$q_{i,j} = \begin{cases} 1 & \text{variable } x_j \text{ is bound by the } i^{\text{th}} \text{ quantifier} \\ 0 & \text{otherwise} \end{cases}$$

To encode the Boolean formula φ we use P a $2m \times 2m \times 2m$ three dimensional binary matrix. Each element of the matrix represents the three variables in a clause in the problem instance. If the element $q_{i,j,k} = 1$ then the variables $x_{i \bmod m}$, $x_{j \bmod m}$ $x_{k \bmod m}$ exist in the clause. If $i < m$ then the variable x_1 is unnegated in the clause while if $i > m$ then the variable x_i appears negated.

The total length of the binary string (we flatten the matrices to strings) to encode an instance of Σ_dSAT with m variables is thus $m^2 + 2m^3$ bits.

4.2 Evaluating quantified Boolean formulas

We now describe a logspace uniform family of active membrane systems without charges to recognise problem Σ_dSAT and where d is odd. (The arguments for even d and for the problem Π_dSAT are similar.) The family encoding function f takes as input the number $1^{m^2+2m^3}$ which is the length of the input instance encoded in unary, from this it calculates the value m which is used to construct the family member Π_n.

We present a high level sketch of the membrane structure and rules of Π_n to convince the reader of the systems existence.

The membrane structure μ of Π_n is represented (in bracket language) as follows:

$$\overbrace{[[\cdots[[}^{d \text{ membranes}} \quad \underbrace{[\,]_{c\langle 1,1,1\rangle}[\,]_{c\langle 1,1,2\rangle}\cdots[\,]_{c\langle 2m,2m,2m\rangle}}_{2m^3 \text{ membranes}}]_{d+1} \quad]_d]_{d-1}\cdots]_2]_1$$

The input membrane is $d+1$ and contains the objects produced by the e function from Definition 3. This function takes a potential instances of $\Sigma_d\mathsf{SAT}$ as input, instances are encoded as binary strings using the scheme described in Section 4.1. For each element $p_{i,j,k} = 0$ of the matrix P an object $\overline{c}_{\langle i,j,k\rangle}$ is created, these represent the clauses *not* used in the instance. For each element $q_{i,j} = 1$ of the matrix Q the object $x_{i,j,0}$ such that $1 \leq i \leq m$ and variable x_i is bound by the j-th quantifier in the input formula. The third subscript of $x_{i,j,0}$ is a time counter, which is incremented during each computation steps by evolution rules such as $[x_{i,j,t} \to x_{i,j,t+1}]_{d+1}$, unless a different behaviour is explicitly described below for some values of t. It is easy to imagine how a uniform constant depth circuit can map the encoding described in Section 4.1 to these objects, so we claim the object encoding function e is in FAC^0.

A timer-object $z_{i,0}$ is contained in membrane i for $1 \leq i \leq d+1$; it is also incremented via $[z_{i,t} \to z_{i,t+1}]_i$ during each step, unless explicitly stated below. All membranes c_j contain an analogous object $z_{0,0}$.

The computation of Π_n on a given input is divided into four phases.

Phase 1. Dissolution of membranes representing unused clauses. Membrane $c_{\langle i,j,k\rangle}$ represents the same clause as the element $p_{i,j,k}$ in Section 4.1. These membranes are dissolved during the first two computation steps if that clause does not occur in the input formula (i.e., if object $\overline{c}_{\langle i,j,k\rangle}$ occurs in the input multiset), according to the following rules:

$$\overline{c}_{\langle i,j,k\rangle}\ [\ ]_{c_{\langle i,j,k\rangle}} \to [\overline{c}_{\langle i,j,k\rangle}]_{c_{\langle i,j,k\rangle}} \qquad\qquad [\overline{c}_{\langle i,j,k\rangle}]_{c_{\langle i,j,k\rangle}} \to \lambda$$

The total duration of Phase 1 is exactly two computation steps.

Phase 2. Quantifier elimination In the second phase we use non-elementary membrane division in order to carry out the process of quantifier elimination, as described in the beginning of this section.

Let x_i be the variable bound by the first quantifier having the smallest subscript. The corresponding object $x_{i,1,2}$ (here the third subscript is 2 because the first phase took two steps) is first moved to membrane 2 (the one immediately below the corresponding quantifier-membrane) by using a series of communication rules:

$$[x_{i,1,2}]_{d+1} \to [\ ]_{d+1}\ x_{i,1,3} \qquad [x_{i,1,3}]_d \to [\ ]_d\ x_{i,1,4} \qquad \cdots$$
$$[x_{i,1,d-1}]_4 \to [\ ]_4\ x_{i,1,d} \qquad [x_{i,1,d}]_3 \to [\ ]_3\ x_{i,1,d+1}$$

Then, object $x_{i,1,d+1}$ divides membrane 2, *duplicating all of its substructure*, and becoming a "true" object on one side and a "false" object on the other:

$$[x_{i,1,d+1}]_2 \to [t_{i,\epsilon,d+2}]_2\ [f_{i,\epsilon,d+2}]_2$$

The second subscript is erased (i.e., replaced by ϵ) in the process, since it is not needed anymore. The object $t_{i,\epsilon,d+2}$ is now brought back to membrane $d+1$

using another series of communication rules:

$$t_{i,\epsilon,d+2}\ [\ ]_3 \to [t_{i,\epsilon,d+3}]_3 \qquad t_{i,\epsilon,d+3}\ [\ ]_3 \to [t_{i,\epsilon,d+4}]_3 \qquad \cdots$$
$$t_{i,\epsilon,2d}\ [\ ]_d \to [t_{i,\epsilon,2d+1}]_d \qquad t_{i,\epsilon,2d+1}\ [\ ]_{d+1} \to [t_{i,\epsilon,2d+2}]_{d+1}$$

and analogously for $f_{i,\epsilon,d+2}$. Notice that now we have *two* instances of membrane $d+1$: in one of them, the variable x_i is set to true, and in the other it is set to false. The timer subscript of $t_{i,\epsilon,d+2}$ and $f_{i,\epsilon,d+2}$ continues to be incremented inside membrane $d+1$.

In the subsequent steps, the objects representing the other variables bound by the first quantifier move to membrane 2 to divide and generate an assignment for their variable then move back to membrane $d+1$. This same process is then performed for all variables bound by the second quantifier, then third and so on until the d^{th}quantifier. The timers can be synchronized correctly by always assuming the longest possible path (from membrane $d+1$ to 2) which is $2d+1$ steps. The time required by Phase 2 is then $m(2d+1)$ steps. At the end of this phase each of the 2^m copies of membrane $d+1$ contains a different assignment (either a $t_{i,\epsilon,m(2d+1)+2}$ or $f_{i,\epsilon,m(2d+1)+2}$ object) to the variables $x_1, \ldots, x_m$.

Phase 3. Evaluation of the matrix of the formula. The objects representing truth assignments of the variables now must be copied so that there are enough for each clause-membrane to take in. That is, each membrane representing a clause containing the literal x_i can bring in a copy of the corresponding "true" object, and each one containing the literal $\bar{x}_i$ can bring in a copy of the corresponding "false" object. Each of the copies is subscripted by the name of one of the clauses which they satisfy, i.e.,

$$[\ t_{i,\epsilon,m(2d+1)+2} \to \{t'_{i,\epsilon,\langle j,k,l\rangle} \mid i=j \vee i=k \vee i=l\}\ ]_{d+1}$$
$$[\ f_{i,\epsilon,m(2d+1)+2} \to \{f'_{i,\epsilon,\langle j,k,l\rangle} \mid i+m=j \vee i+m=k \vee i+m=l\}\ ]_{d+1}$$

Notice that these objects do not need a timer subscript.

To evaluate a clause occurring in the input formula, membrane $c_{\langle i,j,k\rangle}$ tries to bring in one of the objects corresponding to a variable assignment that will make the clause true. (Recall that in the first phase we removed all clauses not appearing in the input instance.) For example, for the membrane $c_{\langle 1,2,6\rangle}$ which represents the clause $x_1 \vee x_2 \vee \bar{x}_3$ uses the following rules:

$$t'_{1,\epsilon,\langle i,j,k\rangle}\ [\ ]_{c_{\langle 1,2,6\rangle}} \to [t'_{d+1}]_{c_{\langle 1,2,6\rangle}} \text{ where } 1=i \vee 1=j \vee 1=k$$
$$t'_{2,\epsilon,\langle i,j,k\rangle}\ [\ ]_{c_{\langle 1,2,6\rangle}} \to [t'_{d+1}]_{c_{\langle 1,2,6\rangle}} \text{ where } 2=i \vee 2=j \vee 2=k$$
$$f'_{3,\epsilon,\langle i,j,k\rangle}\ [\ ]_{c_{\langle 1,2,6\rangle}} \to [t'_{d+1}]_{c_{\langle 1,2,6\rangle}} \text{ where } 6=i \vee 6=j \vee 6=k$$

At most three objects are sent to in each clause membrane $c_{\langle i,j,k\rangle}$ in successive steps. One of the objects t'_{d+1} (whose only subscript indicates that it is a "true" object of level $d+1$, this prevents mixing up truth values on different levels of the membrane structure) after at most three steps dissolves membrane $c_{\langle i,j,k\rangle}$ via

$$[t'_{d+1}]_{c_{\langle i,j,k\rangle}} \to t'_{d+1}$$

If t'_{d+1} has not dissolved $c_{\langle i,j,k\rangle}$ at time $m(2d+1)+7$, then we infer that the clause is not satisfied. The counter object $z_{0,m(2d+1)+7}$ (whose second subscript has been incremented each step) in each remaining clause membrane evolves into a false value and dissolves the membrane:

$$[z_{0,m(2d+1)+7}]_{c_{\langle i,j,k\rangle}} \to f'_{d+1}$$

After at most six computation steps, all the objects denoting the results of the evaluations have been sent to $d+1$.

Membrane $d+1$ now computes the conjunction of the value-objects located inside it. If a "false" object f'_{d+1} appears, then the whole conjunction has a false result which is denoted by f'_d:

$$[f'_{d+1}] \to f'_d$$

If no instance of f'_{d+1} appears inside $d+1$ at time $m(2d+1)+9$, then all clauses evaluate to true. The object $z_{d+1,m(2d+1)+9}$ (obtained by repeatedly increasing the second subscript of $z_{d+1,0}$ as described above for $z_{0,0}$) is then used to produce a true result:

$$[z_{d+1,m(2d+1)+9}] \to t'_d$$

Now each instance of membrane d contains either t'_d or f'_d, each of these objects represents the evaluation of Boolean formula on some assignment. The whole phase requires at most eight steps.

Phase 4. Computing the value of the whole formula The 2^{m-1} copies of membrane d (corresponding to the innermost quantifier of the input formula) must now combine the results coming from the (now dissolved) children membranes labelled by $d+1$. Since d is odd by hypothesis, the last quantifier is $\exists$, hence the results must be combined by disjunction. If a true object t'_d exists, then the result of the evaluation is, in turn, true:

$$[t'_d]_d \to t'_{d-1}$$

otherwise, we can dissolve d via the object $z_{d,m(2d+1)+11}$ (when its counter reaches this value), which is transformed into a false value:

$$[z_{d,m(2d+1)+11}]_d \to f'_{d-1}$$

The evaluation then proceeds to the upper (i.e., outermost) levels of the membrane structure in a completely analogous way, alternating universal quantification (corresponding to conjunction, which is performed as described at the end of Phase 3) and existential quantification. Clearly, the counters of the $z_{i,t}$ objects must be adjusted appropriately: this is easy to accomplish, since the evaluation of each quantifier requires at most two computation steps.

The only difference occurs on the last level: instead of sending out object t'_0 or f'_0 from the outermost membrane, we use the objects `yes` and `no` in order to signal the result of the whole computation.

Phase 4 is completed in $2d$ steps.

This Section describes the proof for the following theorem.

Theorem 8. $\Sigma_d\mathsf{SAT} \in (\mathsf{AC}^0, \mathsf{L})$–$\mathsf{PMC}_{\mathcal{AM}^0_{+wne}}$ *where the depth of the membrane structure is limited to* $d+1$.

Note that we do not give the membrane system family in enough detail to show that it holds for AC^0-uniformity however the system described is easily L uniform.

Corollary 9. $\mathsf{QSAT} \in (\mathsf{AC}^0, \mathsf{L})$–$\mathsf{PMC}_{\mathcal{AM}^0_{+wne}}$ *if the depth of the membrane structure is polynomial of* m*, the number of variables.*

5 Conclusions and future directions

We proved that in the setting of active membrane systems without charges and using non-elementary division rules, a membrane structure of depth $d+1$ is sufficient to decide (in polynomial time) the validity of quantified Boolean formulas with d alternations of quantifiers. An interpretation of this result is that each level of nesting of membranes provides access to an oracle. Since there is no known way to perform the same task using substantially shallower membrane structures, this seems to suggest that increasing the depth of the membrane structure actually increases the computing power of the systems.

Whether this apparent phenomenon corresponds to reality remains an open problem. Future work on this topic may involve simulating arbitrary $(d+1)$-depth families of membrane systems by devices characterising the d^{th} level of the polynomial hierarchy (such as suitable alternating Turing machines). Also, identifying the relationship between depth and membrane division on the one hand, and alternation on the other.

References

1. Artiom Alhazov and Mario J. Pérez-Jiménez. Uniform solution to QSAT using polarizationless active membranes. In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations and Universality (MCU)*, volume 4664 of *LNCS*, pages 122–133, Orléans, France, September 2007. Springer.
2. David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
3. Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustin Riscos-Núñez. A logarithmic bound for solving subset sum with P systems. In *8th International Workshop on Membrane Computing, WMC 2007*, volume 4860 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2007.
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. Giancarlo Mauri, Mario Pérez-Jiménez, and Claudio Zandron. On a Păun's conjecture in membrane systems. In José Mira and José R. Álvarez, editors, *Bio-inspired Modeling of Cognitive Tasks*, volume 4527, pages 180–192. Springer Berlin / Heidelberg, 2007.

6. Niall Murphy and Damien Woods. Active membrane systems without charges and using only symmetric elementary division characterise P. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007 Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*, volume 4860 of *LNCS*, pages 367–384. Springer, 2007.
7. Niall Murphy and Damien Woods. A characterisation of NL using membrane systems without charges and dissolution. In *UC*, pages 164–176. Springer, 2008. To appear.
8. Niall Murphy and Damien Woods. Uniformity conditions in natural computing. In *Proceedings of DNA16*, 2010. To appear.
9. Gheorghe Păun. P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
10. Gheorghe Păun. *Membrane Computing*. Springer-Verlag, Berlin, 2002.
11. 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, 2003.
12. Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Alvaro Romero–Jiménez, and Damien Woods. *Handbook of Membrane systems*, chapter 12: Complexity – Membrane Division, Membrane Creation. Oxford University Press, 2009.
13. Antonio E. Porreca, Alberto Leporati, and Claudio Zandron. On a powerful class of non-universal P systems with active membranes. Submitted, 2010.
14. 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.
15. Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
16. Damien Woods, Niall Murphy, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. Membrane dissolution and division in P. In *Unconventional Computation*, volume 5715, pages 262–276, 2009.