The computational power of membrane systems under tight uniformity conditions

Niall Murphy (nmurphy@cs.nuim.ie)*
Department of Computer Science, National University of Ireland Maynooth, Ireland

Damien Woods (woods@caltech.edu)†
California Institute of Technology, Pasadena, CA 91125, USA

Abstract. We apply techniques from complexity theory to a model of biological cellular membranes known as membrane systems or P-systems. Like Boolean circuits, membrane systems are defined as uniform families of computational devices. To date, polynomial time uniformity has been the accepted uniformity notion for membrane systems. Here, we introduce the idea of using $\mathbf{AC}^0$-uniformity and investigate the computational power of membrane systems under these tighter conditions. It turns out that the computational power of some systems is lowered from **P** to **NL** when using $\mathbf{AC}^0$-semi-uniformity, so we argue that this is a more reasonable uniformity notion for these systems as well as others. Interestingly, other **P**-semi-uniform systems that are known to be lower-bounded by **P** are shown to retain their **P** lower-bound under the new tighter semi-uniformity condition. Similarly, a number of membrane systems that are known to solve **PSPACE**-complete problems retain their computational power under tighter uniformity conditions.

Keywords: membrane systems; P-systems; computational complexity; NL; uniformity; semi-uniformity

1. Introduction

Membrane systems [12] are a model of computation inspired by living cells. In this paper we explore the computational power of cell dissolution (reminiscent of *apoptosis*) by investigating a variant of the model called active membranes [11], originally developed to study the computational power of cell division (as a simple abstraction of, say, *binary fission* in cells). We focus on how uniformity conditions (or precomputation) affect the computational power of the model. An instance of the model consists of a number of (possibly nested) membranes, or compartments, which themselves contain objects. During a computation, objects evolve to become other objects or pass through membranes by the application of rules. In the active membrane model it is also possible for a membrane

* Supported by the Irish Research Council for Science, Engineering and Technology.
† Supported by Junta de Andalucía grant TIC-581 (Spain) and National Science Foundation Grant 0832824, the Molecular Programming Project (USA).

© 2011 *Kluwer Academic Publishers. Printed in the Netherlands.*

to completely dissolve and for a membrane to divide into two child membranes.

The active membrane model can be regarded as a model of parallel computation, however it has a number of features that make it somewhat unusual when compared to other parallel models. For example rule selection is nondeterministic, confluence plays an important role, membranes contain multisets of objects and there are many parameters to the mode. In order to clearly see the power of the model we analyse it from the computational complexity point of view, the goal being to characterise the model in terms of the set of problems that it can solve in a reasonable amount of time. One can also interpret our results as classifying the computational complexity of simulating biological phenomena that are modelled by the membrane systems under consideration.

Another, more specific, motivation is the so-called P-conjecture [13] which states that recogniser membranes systems with division rules (active membranes), but without charges, characterise **P**. On the one hand, it was shown that this conjecture does not hold for systems with non-elementary division as **PSPACE** upper [15] and lower [1] bounds were found for this variant (non-elementary division is where a membrane containing multiple membranes and objects may be copied in a single timestep). On the other hand, the P-conjecture was thought to hold for all active membrane systems without dissolution rules, when Gutiérrez-Naranjo et al. [5] gave a **P** upper-bound. The corresponding **P** lower-bound (trivially) came from the fact that the model is defined to be **P**-uniform, which we now explain.

Like Boolean circuits, membrane systems can be defined as families of finite devices. In order to prevent such a family from being too powerful, we define the family to have an associated algorithm that maps each problem instance size to a family member. This algorithm effectively ensures that family members are algorithmically related, and prevents the family definition from hiding resources that are difficult to precompute. A closely related notion is semi-uniformity, where we map each problem instance directly to a membrane system using a suitably restricted algorithm. Since much of the work on the complexity of membrane systems has been concerned with whether or not polynomial time membrane systems exist for solving intractable problems, polynomial time uniformity, or **P**-uniformity, has been commonly used.

However, the aforementioned **P** lower-bound highlights a problem with using **P**-uniformity, as it does not tell us whether this membrane model itself has (in some sense) the ability to solve all of **P** in polynomial time, or if the uniformity condition is providing the power. In fact, in Section 3 we show that when we use restricted, and more reasonable, uniformity conditions the model does not have the ability to solve all

problems in **P** (assuming **P** $\neq$ **NL**). Essentially we prove that semi-uniform and uniform families of polynomial time active membrane systems, without dissolution rules, solve no more than those problems in **NL**. This is despite the fact that these systems run for polynomial time (and can even create exponentially many objects and membranes). This result is illustrated by the bottom four nodes in Figure 1.

In Section 4 we also give a corresponding **NL** lower-bound for $\mathbf{AC}^0$-semi-uniform families of systems without dissolution indicating that the upper-bound is tight (although we slightly relax the definition of recogniser in order to simplify the construction[1]). Therefore, in the semi-uniform case, we have a characterisation of **NL** which is illustrated by the bottom left two nodes in Figure 1.

So far, we have mentioned four models which characterise **P** under **P**-uniformity but are instead upper-bounded by **NL** under $\mathbf{AC}^0$-uniformity (or **L**-uniformity). However, we claim that if a lower-bound is given by a membrane system construction and not by exploiting a powerful uniformity/semi-uniformity condition, then the power of the model should be unaffected by the change to a less-powerful uniformity/semi-uniformity condition. Interestingly, in Section 5 we show that another **P**-uniform membrane system model (with dissolution but no division) known [17] to characterise **P** actually retains this **P** characterisation when restricted to be $\mathbf{AC}^0$-semi-uniform (or **L**-semi-uniform). To show this, we give an $\mathbf{AC}^0$-semi-uniform family of membrane systems with dissolution rules that solves a **P**-complete problem. This is illustrated by the top front left node in Figure 1.

Finally, in Section 6, we show that the aforementioned **PSPACE** characterisations (top back two nodes in Figure 1) remain unchanged under tighter uniformity conditions.

2. Membrane Systems

In this section we define active membrane systems and some complexity classes. These definitions are based on those from Păun [11, 12], Pérez-Jiménez et al. [14], and Sosík and Rodríguez-Patón [15]. We also introduce the notion of $\mathbf{AC}^0$-uniformity and $\mathbf{AC}^0$-semi-uniformity for membrane systems. The set of all multisets over a set A is denoted $\mathrm{MS}(A)$.

[1] See [8] for a construction that works for the standard definition of recogniser.

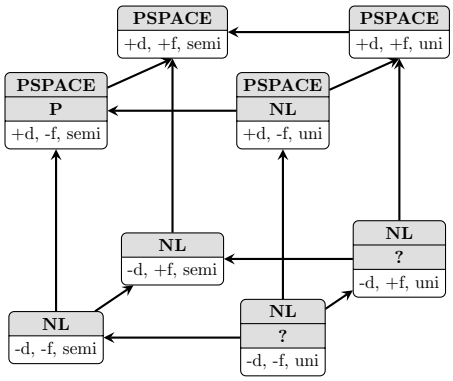

Figure 1. A diagram showing the currently known upper and lower-bounds on variants of chargeless active membranes systems with uniformity conditions computable in **L** or stricter. The lower part of each node indicates the properties that system: the parameter "-d" indicates type (d) rules are prohibited while "-f" means type (f) rules are prohibited, "uni" and "semi" indicate uniform and semi-uniform families respectively. The top part of a split node represents the best known upper-bound, and the lower part the best known lower-bound. A node with a single complexity class represents a characterisation. Arrows represent inclusions.

2.1. Active membrane systems

Active membrane systems are a class of 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).

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 (or the set of object types);*
2. $\mu = (V_\mu, E_\mu, r)$ *is a rooted tree representing the membrane structure where* $V_\mu \subsetneq \mathbb{N}$ *is finite,* $E_\mu \subsetneq V_\mu \times V_\mu$*, and root* $r \in V_\mu$*;*
3. $M : V_\mu \to \mathrm{MS}(O)$ *maps membranes to their object multisets;*
4. H *is the finite set of membrane labels;*
5. $L : V_\mu \to H$ *is an injective mapping of membranes to 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) $a\,[\,]_h \rightarrow [\,b\,]_h$ *(communication in),*

(c) $[\,a\,]_h \rightarrow [\,]_h\ b$ *(communication out),*

(d) $[\,a\,]_h \rightarrow b$ *(membrane dissolution),*

(e) $[\,a\,]_h \rightarrow [\,b\,]_h\ [\,c\,]_h$, *(elementary membrane division),*

(f) $[\,[\,]_{h1}\,[\,]_{h2}\,]_h \rightarrow [\,[\,]_{h1}\,]_h\,[\,[\,]_{h2}\,]_h$, *(strong non-elementary membrane division).*

The vertices V_μ of the membrane structure tree μ are the individual membranes of the system. The ultimate container of all membranes in the system (the root vertex r in μ) is called the skin and has label $0 \in H$ (when defining rules we let $0 = \text{skn}$). A *configuration* $\mathcal{C}$ of a membrane system is a tuple (μ, M, L) whose elements are defined in Definition 1 (with the exception that L may be surjective). A *permissible encoding* of a membrane system $\langle\Pi\rangle$, or of a configuration $\langle\mathcal{C}\rangle$, encodes all multisets in a unary manner. For example, a multiset should be encoded in the format $[\,a, a, a, b, b\,]$, rather than in the shorter form a^3b^2, in order to ensure that at most a polynomial number of objects are initially encoded in a system. We also permit the use of a blank symbol (denoted $\sqcup$) which may be inserted at any point in the encoding.

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*. That is, each timestep a multiset of applicable rules is non-deterministically chosen such that any further rules added to the set cannot be applied in that timestep.

- If a membrane is divided by a rule of type (e) or (f) and there are objects in this membrane which evolve via rules of type (a), then we assume that first the evolution rules are applied, and then the division rule. This process takes only one step.

- The rules with label h are used with membranes with label h. In each timestep, a membrane can be the subject of only one rule of types (b)–(f).

A *computation* of a membrane system is a maximal 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. $\texttt{yes}, \texttt{no} \in O$*,*
3. *the object* `yes` *or object* `no` *(but not both) appear in the multiset of the membrane with label* 0 *(or "*skn*"),*
4. *and this happens only in the halting configuration.*

2.2. Complexity classes

We introduce the notion of $\mathbf{AC}^0$-semi-uniformity and $\mathbf{AC}^0$-uniformity to membrane systems. Throughout this paper, $\mathbf{AC}^0$ is the set of problems solved by **DLOGTIME**-uniform, polynomial sized (in input length n), constant depth, circuits with AND, OR and NOT gates, and unbounded fan-in [3]. **FP**, **FL**, and $\mathbf{FAC}^0$ 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 **DLOGTIME**-uniform polynomial-sized Boolean circuits with unbounded fan-in and constant depth. Previous work on the computational complexity of membrane systems used (Turing machine) polynomial time uniformity [14]. (A notable exception is the logspace semi-uniform membrane system family by Obtułowicz [9].)

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 X if for each $x \in \Sigma^*$ there is some $\Pi \in \mathbf{\Pi}$ that decides if $x \in X$. We let $|x| = n$ denote the length of a problem instance $x \in \Sigma^*$.

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* $\mathbf{E}$ *and* $\mathbf{F}$ *be classes of functions. The class of* problems solved by an $(\mathbf{E}, \mathbf{F})$-uniform family of membrane systems *of type* $\mathcal{R}$ *in time* t*, denoted* $(\mathbf{E}, \mathbf{F})$–$\mathbf{MC}_{\mathcal{R}}(t)$*, contains all problems* X *such that:*

- *There exists an* $\mathbf{F}$-uniform *family of membrane systems,* $\mathbf{\Pi} = \{\Pi_1, \Pi_2, \ldots\}$ *of type* $\mathcal{R}$*: that is, there exists a function* $f \in \mathbf{F}$*,* $f : \{1\}^* \to \mathbf{\Pi}$ *such that* $f(1^n) = \Pi_n$*.*
- *There exists an input encoding function* $e \in \mathbf{E}$*,* $e : X \cup \overline{X} \to \mathrm{MS}(I)$ *such that* $e(x)$ *is the input multiset, which is placed in a specific input membrane of* $\Pi_{|x|}$*, 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* $x \in X$*, each computation of the system* $\Pi_{|x|}$ *on input multiset* $e(x)$ *is accepting.*

We now define semi-uniform families of membrane systems where a single function (rather than two) is used to construct the family. For each instance $x \in X \cup \overline{X}$ we have a (possibly unique) membrane system which does not need a separately constructed input, a clear departure from the spirit of circuit uniformity.

DEFINITION 4. *Let* $\mathbf{H}$ *be a class of functions. The class of* problems solved by a $(\mathbf{H})$-semi-uniform family of membrane systems *of type* $\mathcal{R}$ *in time* t*, denoted* $(\mathbf{H})$–$\mathbf{MC}^*_{\mathcal{R}}(t)$*, contains all problems* X *such that:*

– *There exists a* $\mathbf{H}$-semi-uniform *family* $\mathbf{\Pi} = \{\Pi_{x_1}, \Pi_{x_2}, \ldots\}$ *of membrane systems of type* $\mathcal{R}$*: that is, there exists a function* $h \in \mathbf{H}$*,* $h : X \cup \overline{X} \to \mathbf{\Pi}$ *such that* $h(x_i) = \Pi_{x_i}$*.*

– $\mathbf{\Pi}$ *is t-efficient:* Π_x *always halts in at most* $t(|x|)$ *steps.*

– *The family* $\mathbf{\Pi}$ *is* sound *with respect to* (X, h)*; that is, for each* $x \in \Sigma^*$*, if there exists an accepting computation of the system* Π_x *then* $x \in X$*.*

– *The family* $\mathbf{\Pi}$ *is* complete *with respect to* (X, h)*; that is, for each* $x \in X$ *every computation of the system* Π_x *is accepting.*

We define the set of languages decided by uniform families of polynomial time membrane systems to be

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

and the set of languages decided by semi-uniform families of polynomial time membrane systems to be

$$(\mathbf{H})\text{–}\mathbf{PMC}^*_{\mathcal{R}} = \bigcup_{k \in \mathbb{N}} (\mathbf{H})\text{–}\mathbf{MC}^*_{\mathcal{R}}(n^k).$$

When the symbols $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{H}$ are replaced by complexity class names such as $\mathbf{AC}^0$, $\mathbf{L}$ or $\mathbf{P}$ it means that the uniformity conditions under

consideration are in the function versions of these classes. For example, if we let $\mathbf{E} = \mathbf{F} = \mathbf{AC}^0$ then we mean that the functions $e \in \mathbf{E}$ and $f \in \mathbf{F}$ are computable in uniform $\mathbf{FAC}^0$ and we say that we have an $\mathbf{AC}^0$-uniform family.

Let $\mathcal{AM}^0_{-d}$ denote the class of membrane systems that obey Definition 2, and Definition 1 but without dissolution rules (type (d)). Then $(\mathbf{AC}^0, \mathbf{AC}^0)$–$\mathbf{PMC}_{\mathcal{AM}^0_{-d}}$ (respectively, $(\mathbf{AC}^0)$–$\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$) denotes the class of problems solvable by $\mathbf{AC}^0$-uniform (respectively, $\mathbf{AC}^0$-semi-uniform) families of polynomial time active membrane systems without charges and with no dissolution rules.

A family of membrane systems is said to be *confluent* if it is both sound and complete with respect to (X, e, f) in the uniform case and to (X, h) in the semi-uniform case. That is, each membrane system Π in a confluent family starts from a fixed initial configuration (from either $h(x)$, or $f(1^{|x|})$ and $e(x)$). Then, the system Π non-deterministically chooses one from a number of valid computations (configuration sequences). All of these valid computations give the same result: either always accepting (if $x \in X$) or else always rejecting (if $x \notin X$). All membrane system families in this paper are confluent.

$\mathbf{FAC}^0$ is usually defined using uniform Boolean circuits, however, it can be cumbersome to use uniform circuits to define uniformity conditions on membrane systems. $\mathbf{FAC}^0$ is also characterised by a number of models that are easier to analyse such as the constant time Concurrent Random Access Machine (constant time CRAM) [2, 7]. We often use a CRAM algorithm to demonstrate that families of membrane systems are $\mathbf{AC}^0$-uniform. We give a brief definition, see Immerman [7] for details.

DEFINITION 5. (CRAM [7]). *A CRAM is a concurrent-read concurrent-write parallel model of computation. A CRAM has a polynomial number of processors, each with a unique processor number, that share a common global memory. Processors run programs that act on words in memory and have instructions to add, subtract, branch, and shift by a polynomial number of bits.*

3. NL upper-bound on active membranes without dissolution rules

Previously the upper-bound on all active membrane systems without dissolution was $\mathbf{P}$ [5], that is $(\mathbf{P})$–$\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}} \subseteq \mathbf{P}$. Since membrane systems are usually $\mathbf{P}$-uniform, this $\mathbf{P}$ upper-bound was considered a

P characterisation. However, having a lower-bound of the same power as the uniformity condition is somewhat unsatisfactory, as it tells us little about the computing power of the actual membrane system itself. This is because the algorithm that encodes the input (the function e in the uniform case, or h in the semi-uniform case) takes an instance of the problem as input. If the input encoder is sufficiently powerful then it may simply solve the problem and output a `yes` or `no` object directly. To get an accurate idea of the power of a model it is advisable to restrict the uniformity conditions to be as weak as possible [3].

In this section we show that when we restrict the (semi-)uniformity conditions to be computable in $\mathbf{FAC}^0$, or even **FL**, we tighten the known upper-bound from **P** to **NL**. The proof of the **P** upper-bound in [5] involves the construction of a *dependency graph* representing all possible computation paths of a membrane system on an input. The dependency graph for a membrane system Π is a directed graph $\mathcal{G}_\Pi = (\mathcal{V}, \mathcal{E})$. Each vertex v in the graph is a pair $v = (a, h) \in O \times H$, where O is the set of objects and H is the set of membrane labels. An edge connects vertex u to vertex v if there is an evolution rule such that the left hand side of the rule has an object-membrane pair matching u and the right has an object-membrane pair matching v. We formally define this as follows (where parent(i) is the parent membrane of i in the membrane structure μ of Π).

DEFINITION 6. *Let* Π *be a recogniser active membrane system without charges and without dissolution rules* ($\mathcal{AM}^0_{-d}$)*. Let* R *be the set of rules associated with* Π*. The* dependency graph associated with Π *is the directed graph* $\mathcal{G}_\Pi = (\mathcal{V}, \mathcal{E})$ *defined as follows:*

$$\begin{aligned}
&\mathcal{V} = O \times H, \mathcal{E} = \{((a,h),(b,h')) \mid \\
&\quad ([\,a \rightarrow u\,]_h \in R, b \in u, u \in \mathrm{MS}(O), h = h') \vee \\
&\quad (a[\,]_{h'} \rightarrow [\,b\,]_{h'} \in R, i \in V_\mu, h = L(\mathrm{parent}(i)), h' = L(i)) \vee \\
&\quad ([\,a\,]_h \rightarrow [\,]_h b \in R, i \in V_\mu, h' = L(\mathrm{parent}(i)), h = L(i)) \vee \\
&\quad ([\,a\,]_h \rightarrow [\,c\,]_h[\,d\,]_h \in R, h = h', b \in \{c, d\})\}.
\end{aligned}$$

Let $\mathcal{I} \subseteq O \times H$ where $\mathcal{I} = \{(x, h) \mid x \in M(i), h = L(i), i \in V_\mu\}$, that is, vertices representing objects in the initial configuration. In the previous definition, the vertices $(\texttt{yes}, \mathrm{skn})$ and $(\texttt{no}, \mathrm{skn})$ respectively represent the objects `yes` and `no` in the output membrane. If there is a path from a vertex representing the input to the vertex $(\texttt{yes}, \mathrm{skn})$ then it is clear that this system is an accepting one. It is worth noting that, unlike upper-bound proofs for a number of other computational models, the dependency graph does not model entire configuration sequences, but rather models only certain aspects of configurations.

For the **P** upper-bound proof [5], the dependency graph was constructed in polynomial time, we now show that it can be constructed in $\mathbf{FAC}^0$.

LEMMA 1. *Given an encoding of a membrane system* $\langle\Pi\rangle$*, its dependency graph* $\mathcal{G}_\Pi$ *is constructable in* $\mathbf{FAC}^0$*.*

Proof. We provide a constant time CRAM algorithm that when given an encoding of a membrane system constructs the encoding of the dependency graph for that system. By definition every membrane in the system has a unique label at the initial timestep.

The CRAM algorithm assumes that the structure μ of the membrane system is encoded as an adjacency matrix $M_{|H|\times|H|}$. It also assumes that the set of rules R are arranged in a matrix with $|R|$ rows (one for each rule) and $m+5$ columns where m is the size of the largest multiset in a rule of type (a) in R (one row for each multiset symbol). Each rule is prefixed with its type ((a), (b), etc.), stored in column 1. From the set of rules R and the membrane structure μ, the CRAM algorithm computes an adjacency matrix $\mathcal{G}_{n\times n}$, where $n = |O||H|$, that encodes the dependency graph. The CRAM uses binary strings to encode matrix indices, in the usual way. The objects $a, b \in \{1, 2, \ldots, |O|\}$ are written in binary, as is $h \in \{1, 2, \ldots, |H|\}$. We define $\langle a, h\rangle = \text{shift}(a, \lceil\log_2(|H|+1)\rceil) + h$, where $\lceil\log_2(|H|+1)\rceil$ is computed via masking of the binary string that encodes $|H|$.

The CRAM has a processor for each entry in the rules matrix. The first column of processors check the type of the rule stored in their row, then each processor in the row looks at its designated part of the rule. If the rule is of type:

(a) One processor reads the triggering object a, another reads the membrane label h. There are m processors that each read an object in the multiset u (or blank). In the next timestep, each processor that read an object type $b \in u$ combines this with $a, h, |H|$ and then sets entry $g_{\langle a,h\rangle,\langle b,h\rangle} = 1$ in $\mathcal{G}$.

(b) One processor reads the triggering object a, another reads the membrane label h, and a third the resulting object b. The processor that read the label h writes it to a global register. Another $|H|$ processors check, in parallel, each entry of row h of M for a 1, the processor that finds $m_{h,j} = 1$ has j as part of its processor ID, and writes j to a global register. In the next timestep the processor that read object type b sets $g_{\langle a,j\rangle\langle b,h\rangle} = 1$ in the adjacency matrix of $\mathcal{G}$.

(c) One processor reads the triggering object a, another the membrane label h, and a third the resulting object b. The parent j of membrane h is found using a similar method as for type (b) rules. In the

next timestep the processor that read object b sets $g_{\langle a,h\rangle\langle b,j\rangle} = 1$ in the adjacency matrix of $\mathcal{G}$.

(e) One processor reads the triggering object a, another the membrane label h, another the resulting object b, and a fourth reads the resulting object c. In the next timestep the processors that read objects b and c respectively set $g_{\langle a,h\rangle\langle b,h\rangle} = 1$ and $g_{\langle a,h\rangle\langle c,h\rangle} = 1$ in the adjacency matrix of $\mathcal{G}$.

(f) The algorithm ignores the rule.

Thus a dependency graph is constructable from a membrane system in constant time by a CRAM, and hence in $\mathbf{FAC}^0$. □

In the previous **P** upper-bound result [5] a polynomial time algorithm was given to find a path from the $(\texttt{yes}, \mathrm{skn})$ vertex back to a vertex in $\mathcal{I}$. We now observe that this problem is reducible to STCON, the canonical **NL**-complete problem.

DEFINITION 7. (s-t connectivity (STCON)).
Instance: *A directed acyclic graph with vertices V, edges E, and vertices $s, t \in V$.*
Problem: *Given $G = (V, E, s, t)$, is there a directed path from s to t?*

THEOREM 1. $(\mathbf{AC}^0)$–$\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}} \subseteq \mathbf{NL}$

Proof. Given a membrane system Π of the type in the statement, we use Lemma 1 to convert it (in $\mathbf{FAC}^0$) to a dependency graph $\mathcal{G}$. This graph has the property that there is a path from one of the vertices in $\mathcal{I}$ to $(\texttt{yes}, \mathrm{skn})$ iff the system Π accepts. A constant time CRAM adds extra nodes s, t to $\mathcal{G}$ and new edges from s to each vertex in $\mathcal{I}$ and from $(\texttt{yes}, \mathrm{skn})$ to t. This yields an instance of STCON that has the property that there is a path from s to t iff the system Π accepts. □

This holds for both $\mathbf{AC}^0$ and **L**-uniformity, as well as for both uniform and semi-uniform families of membrane systems without dissolution.

4. NL lower-bound for semi-uniform active membranes without dissolution

In this section we give a membrane system that solves STCON in a semi-uniform manner. The algorithm works by representing edges in the problem instance graph as object evolution rules. For example, edges $(s, b), (s, c), (s, d)$ are represented as the rule $[\, s \rightarrow b, c, d \,]$. There is only

one membrane, the skin, which serves as the output membrane. The system is initialised with an s object (the start vertex) which is then acted upon by the rules. In this manner the presence of an object in a configuration indicates that the system is currently at this vertex while following (or simulating) each different path through the graph in parallel. If the t object is ever evolved the system evolves a `yes` object.

THEOREM 2. $\mathbf{NL} \subseteq (\mathbf{AC}^0)\text{–}\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$

Proof. An instance of the problem STCON is a tuple (V, E, s, t). We define a function $h(x)$, computable in $\mathbf{FAC}^0$, that maps an instance x of STCON to the membrane system Π_x.

$$\begin{aligned}\Pi_x = (&O = \{\texttt{yes}, \texttt{no}\} \cup \{c_i \mid 0 \leq i \leq |V| + 1\} \cup V,\\ &(\{0\}, \emptyset, 0),\\ &M = \{(0, \{s, c_{|V|+1}\}),\\ &H = \{\text{skn}\},\\ &L = \{(0, \text{skn})\},\\ &R = R_E \cup R_Y \cup R_C \cup R_D).\end{aligned}$$

The initial configuration contains only a single membrane which contains the object representing the start vertex s.

The evolution rules come directly from the edges of the input graph. If vertex u has edges to vertices $v_1, \ldots, v_i$ then we encode them as a single type (a) rule:

$$R_E = \{[\, u \rightarrow U\,]_{\text{skn}} \mid u \in V, U = \{v \in V \mid (u, v) \in E\}\}$$

Since the rules directly represent the edges in the graph, the object s produces the object t in at most $|V| - 1$ timesteps iff t is reachable from s. The object t then becomes the `yes` object indicating that a path from s to t exists and the computation is an accepting one.

$$R_Y = \{[\, t \rightarrow \texttt{yes}\,]_{\text{skn}}\}$$

The rules provided so far are sufficient for the family to accept instances of STCON, however a family of recogniser membrane systems should also reject non-instances of a problem. Since $\mathbf{NL} = \mathbf{coNL}$ [6, 16] these systems can also recognise **coNL**-complete problems. To simplify our construction of a system that both accepts and rejects inputs we generalise the definition of a recogniser membrane system in a way that does not change the class $\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$ under $\mathbf{AC}^0$ reductions [8]. We relax condition 3 in Definition 2 so that a single computation may produce both a `yes` and a `no` object (we consider the first one produced to be the answer).

Proceeding with this generalised definition we add a counter that counts down in parallel with the above steps.

$$R_C = \{[\, c_i \;\rightarrow\; c_{i-1}\,]_{\text{skn}} \mid i \in \{1, 2, \ldots, |V| + 1\}\}$$

Since the longest acyclic path in the graph is $|V|-1$ nodes long, any `yes` object evolvable by the system appears before timestep $|V|$. In timestep $|V|+1$ the counter evolves a `no` object, if no `yes` objects were produced before this time, then this `no` objects signifies a rejecting computation.

$$R_D = \{[\, c_0 \;\rightarrow\; \texttt{no}\,]_{\text{skn}}\}$$

The function h writes out members of this family and is easily computable in **FL**. Moreover, each member of the family is constructable by a constant time CRAM and so in $\mathbf{FAC}^0$, we give the details of a CRAM algorithm to convert a group of edges leaving the same node, for example $(s, b), (s, c), (s, d)$, into a single rule $[\; s \;\rightarrow\; b, c, d \;]$.

The CRAM takes an encoding of the graph G as a binary adjacency matrix and outputs the set of rules for the membrane system. The output registers of the CRAM are initially all blank (set to the symbol "$\sqcup$") and hold up to $|O|$ different rules in rows each using $|O|+5$ columns. One processor reads each element of the $n \times n$ adjacency matrix and checks if its element $g_{a,b} = 1$. If it is 1 then it writes out "$[\, a \;\rightarrow$" in the first 3 columns of the ath row, the object b at the bth column, and "$]_{\text{skn}}$" in the last column. (If multiple processors try to write the same information to the same register, it does not matter which succeeds.) For example, the edges $\{(c, e), (a, d), (c, d), (a, b), (a, f), (c, f)\}$ become the rules

$$[\, a \;\rightarrow\; b\sqcup d\sqcup f\,]_{\text{skn}}\,, \quad \sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\,, \quad [\, c \;\rightarrow\; \sqcup\sqcup def\,]_{\text{skn}}.$$

□

Note that in the previous proof we encode the edges of the graph as rules, rather than objects. Therefore our algorithm is *semi-uniform* as we require a different membrane system for each unique problem instance. In the membrane computing framework, for *uniform* membrane systems, inputs must be specified (encoded) as objects. It is also interesting to note that our solution uses only type (a) object evolution rules.

By combining Theorems 1 and 2 we get an **NL** characterisation for semi-uniform families.

COROLLARY 1. $\mathbf{NL} = (\mathbf{AC}^0)\text{–}\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$

5. P lower-bound for semi-uniform families of active membrane systems with dissolving rules

So far we have seen that by tightening the uniformity condition from $\mathbf{P}$ to $\mathbf{AC}^0$ we lower the power of some models from $\mathbf{P}$ down to $\mathbf{NL}$. In this section we show that this does not happen for all models that solve any problem in $\mathbf{P}$. More precisely we give an $\mathbf{AC}^0$-semi-uniform family of polynomial time membrane systems with dissolution rules that solves AGAP, the $\mathbf{P}$-complete [4] analogue of STCON.

PROBLEM 1. (Alternating Graph Accessibility Problem (AGAP) [4]).
Instance: *A directed acyclic graph with vertices V, edges E, a set $A \subseteq V$ of universal vertices ($V \backslash A$ are existential vertices), and $s, t \in V$.*
Problem: *Given $G = (V, E, A, s, t)$, does* $\mathrm{apath}(s, t)$ *hold?*

The function $\mathrm{apath}(u, v)$ holds iff
- $u = v$ or
- u is existential (that is $u \in V \backslash A$) and there exists $w \in V$ with $(u, w) \in E$ and $\mathrm{apath}(w, v)$ holds, or
- u is universal (that is $u \in A$) and for all $w \in V$ with $(u, w) \in E$, $\mathrm{apath}(w, v)$ holds.

THEOREM 3. $\mathbf{P} \subseteq (\mathbf{AC}^0)\text{–}\mathbf{PMC}^*_{\mathcal{AM}^0_{+d}}$

The remainder of this section consists of an $\mathbf{AC}^0$-semi-uniform family of membrane systems $\mathbf{\Pi}_{\mathrm{AGAP}}$ such that for each $G = (V, E, A, s, t)$ there exists a membrane system $\Pi_G \in \mathbf{\Pi}_{\mathrm{AGAP}}$ that accepts iff $G \in \mathrm{AGAP}$. To simplify the proof, we assume that the graph G has an extra existential vertex $\sigma \in V \backslash A$ and an extra edge $(\sigma, s) \in E$, and that s has no other incoming edges. It is not difficult to see that AGAP remains $\mathbf{P}$-complete under this assumption.

Let $m = |V|$. We also assume a total ordering $\leq$ on V when defining the membrane structure E_μ. We let $\sigma = V[0]$ be the first element in the ordering, $s = V[1]$ be the second element, and $V[m-1]$ be the last element. (Note that many rules begin numbering from 1 not 0). Let $V_{\leq w} = \{V[0], V[1], \ldots, w\}$ where $w \in V$, and let $V_{>w} = V \setminus V_{\leq w}$.

$$
\begin{aligned}
\Pi_G = (\, O = & \{\langle u_i, v_{i+1}\rangle \mid u, v \in V, 0 \le i < m\} \cup \\
& \{\langle u_i, v_{i+1}\rangle' \mid u, v \in V, 0 \le i < m\} \cup \\
& \{\langle u_i, v_{i+1}\rangle^{\mathrm{N}} \mid u, v \in V, 0 \le i < m\} \cup \\
& \{\langle u_i, v_{i+1}\rangle^{\mathrm{Y}} \mid u, v \in V, 0 \le i < m\} \cup \\
& \{c_i \mid 0 \le i < m\} \ \cup \{d_i \mid 0 \le i \le 2\} \cup \\
& \{u_i^{\mathrm{Y}}, u_i^{\mathrm{Y}W}, u_i^{\mathrm{N}}, u_i^{\mathrm{N}W} \mid u \in V, 0 \le i < m\}, \\
(V_\mu = & \{u_i, u_i^{\mathrm{e}} \mid u \in V \backslash \{\sigma\}, 1 \le i < m\} \cup \{\mathrm{in}, \mathrm{skn}, \mathrm{fin}\}, \\
E_\mu = & \ \{(V[u]_i^{\mathrm{e}}, V[u]_i) \mid 1 \le u < m, 1 \le i < m\} \cup \\
& \{(V[u-1]_i, V[u]_i^{\mathrm{e}}) \mid 1 \le u < m, 1 \le i < m\} \cup \\
& \{(V[m-1]_{i-1}, V[1]_i^{\mathrm{e}}) \mid 2 \le i < m\} \cup \\
& \{(\mathrm{fin}, V[1]_1^{\mathrm{e}}), (\mathrm{skn}, \mathrm{fin}), (V[m-1]_{m-1}, \mathrm{in})\}, \\
& \mathrm{skn}), \\
M = & \{(\mathrm{in}, \{\langle \sigma_0, s_1\rangle', \mathrm{c}_{m-1}\})\}, \\
H = & \{u_i, u_i^{\mathrm{e}} \mid u \in V \backslash \{\sigma\}, 1 \le i < m\} \cup \{\mathrm{in}, \mathrm{skn}, \mathrm{fin}\}, \\
L = & \ \{(u_i^{\mathrm{e}}, u_i^{\mathrm{e}}) \mid u \in V \backslash \{\sigma\}, 1 \le i < m\} \cup \\
& \{(u_i, u_i) \mid u \in V \backslash \{\sigma\}, 1 \le i < m\} \cup \\
& \{(\mathrm{in}, \mathrm{in}), (\mathrm{skn}, \mathrm{skn}), (\mathrm{fin}, \mathrm{fin})\}, \\
R = & R_F \cup R_{SB} \cup R_{S\forall} \cup R_{SC} \cup R_{S\exists} \cup R_{SE})
\end{aligned}
$$

where

$$
\begin{aligned}
R_F = \{[\, \langle u_i, v_{i+1}\rangle' \ \to \ \langle u_i, v_{i+1}\rangle, \{\langle v_{i+1}, w_{i+2}\rangle' \mid (v, w) \in E\}\,]_{\mathrm{in}} & \quad \text{(F1)} \\
\text{for all } (u, v) \in E \text{ and } 0 \le i \le m-2\} \cup & \\
\{[\, \mathrm{c}_j \ \to \ \mathrm{c}_{j-1}\,]_{\mathrm{in}} \mid 1 \le j \le m\} \cup & \quad \text{(F2)} \\
\{[\, \mathrm{c}_0\,]_{\mathrm{in}} \ \to \ d_2\} & \quad \text{(F3)}
\end{aligned}
$$

$$
\begin{aligned}
R_{SB} = \{[\, \langle u_{i-1}, t_i\rangle\,]_{t_i} \ \to \ u_{i-1}^{\mathrm{Y}} \mid u \in V \backslash \{t\}, 1 \le i < m\} \cup & \quad \text{(S4)} \\
\{[\, \langle u_{i-1}, v_i\rangle^{\mathrm{N}} \ \to \ u_{i-1}^{\mathrm{N}}\,]_{v_i^{\mathrm{e}}} \mid u, v \in V \backslash \{t\}, 1 < i < m\} \cup & \quad \text{(S5)} \\
\{[\, \langle u_{i-1}, v_i\rangle^{\mathrm{Y}} \ \to \ u_{i-1}^{\mathrm{Y}}\,]_{v_i^{\mathrm{e}}} \mid u, v \in V \backslash \{t\}, 1 < i < m\} \cup & \quad \text{(S6)} \\
\{[\, \langle u_{i-1}, v_i\rangle^{\mathrm{Y}} \ \to \ \langle u_{i-1}, v_i\rangle^{\mathrm{N}}\,]_{v_i} \mid u, v \in V \backslash \{t\}, 1 < i < m\} & \quad \text{(S7)}
\end{aligned}
$$

$$R_{S\forall} = \{[\,\langle u_{i-1}, v_i\rangle \to \langle u_{i-1}, v_i\rangle^{\mathrm{N}}\,]_{v_i} \text{ such that } u \in V\backslash\{t\}, v \in A\backslash\{\sigma, t\}, 1 < i < m\} \cup \tag{S8}$$

$$\{[\langle u_{i-1}, v_i\rangle^{\mathrm{N}} \to \langle u_{i-1}, v_i\rangle^{\mathrm{Y}}]_{v_i} \text{ such that } u \in V\backslash\{t\}, v \in A\backslash\{\sigma, t\}, 1 < i < m\} \cup \tag{S9}$$

$$\{[\,v_i^{\mathrm{Y}} \to v_i^{\mathrm{YW}}\,]_{v_i} \mid v \in A\backslash\{\sigma, t\}, 1 \le i < m\} \cup \tag{S10}$$

$$\{[\,v_i^{\mathrm{N}}\,]_{v_i} \to \lambda \mid v \in A\backslash\{\sigma, t\}, 1 \le i < m\} \cup \tag{S11}$$

$$\{[\,v_i^{\mathrm{YW}}\,]_{v_i} \to \lambda \mid v \in A\backslash\{\sigma, t\}, 1 \le i < m\} \tag{S12}$$

$$R_{SC} = \{[\,d_k \to d_{k-1}\,]_{v_i} \mid k \in \{1, 2\}, v \in V\backslash\{\sigma\}, 1 \le i < m\} \cup \tag{S13}$$

$$\{[\,d_0\,]_{v_i} \to d_0 \mid v \in V\backslash\{\sigma\}, 1 \le i < m\} \cup \tag{S14}$$

$$\{[\,d_k\,]_{v_i^{\mathrm{e}}} \to d_2 \mid k \in \{0, 1\}, v \in V\backslash\{\sigma\}, 1 \le i < m\} \tag{S15}$$

$$R_{S\exists} = \{[\langle u_{i-1}, v_i\rangle \to \langle u_{i-1}, v_i\rangle^{\mathrm{Y}}]_{v_i} \text{ such that } u \in V\backslash\{t\}, v \in V\backslash(A \cup \{\sigma, t\}), 1 < i < m\} \cup \tag{S16}$$

$$\{[\,v_i^{\mathrm{N}} \to v_i^{\mathrm{NW}}\,]_{v_i} \mid v \in V\backslash(A \cup \{\sigma, t\}), 1 \le i < m\} \cup \tag{S17}$$

$$\{[\,v_i^{\mathrm{Y}}\,]_{v_i} \to \lambda \mid v \in V\backslash(A \cup \{\sigma, t\}), 1 \le i < m\} \cup \tag{S18}$$

$$\{[\,v_i^{\mathrm{NW}}\,]_{v_i} \to \lambda \mid v \in V\backslash(A \cup \{\sigma, t\}), 1 \le i < m\} \cup \tag{S19}$$

$$R_{SE} = \{[\,\sigma_0^{\mathrm{N}} \to \sigma_0^{\mathrm{NW}}\,]_{\mathrm{fin}}\} \cup \tag{S20}$$

$$\{[\,\sigma_0^{\mathrm{Y}}\,]_{\mathrm{fin}} \to \texttt{yes}\} \cup \tag{S21}$$

$$\{[\,\sigma_0^{\mathrm{NW}}\,]_{\mathrm{fin}} \to \texttt{no}\} \tag{S22}$$

LEMMA 2. *The function* $h : \Sigma^* \to \mathbf{\Pi}_{\mathrm{AGAP}}$ *is in* $\mathbf{FAC}^0$

Proof sketch. The input word G is interpreted as an instance of AGAP encoded as: a binary adjacency matrix of edges E, a binary $|V|$-vector that encodes the universal vertices A, and vertices s and t.

The system $\Pi_G \in \mathbf{\Pi}_{\mathrm{AGAP}}$ has $\mathcal{O}(|V|^3)$ distinct object types, $\mathcal{O}(|V|^2)$ membranes and $\mathcal{O}(|V|^3)$ different rules.

We describe how the Rules (F1) are computed by a CRAM in constant time (using similar techniques as used in the proof of Theorem 2). The rules are specified for values of i from 0 to $|V| - 1$, for each i a group of processors work in parallel as follows. For each entry $e_{u,v} = 1$ in the edge adjacency matrix of G, a processor writes the first part of the rule "$[\,\langle u_i, v_{i+1}\rangle' \to \langle u_i, v_{i+1}\rangle$," and the closing part "$]_{\mathrm{in}}$" (where i is

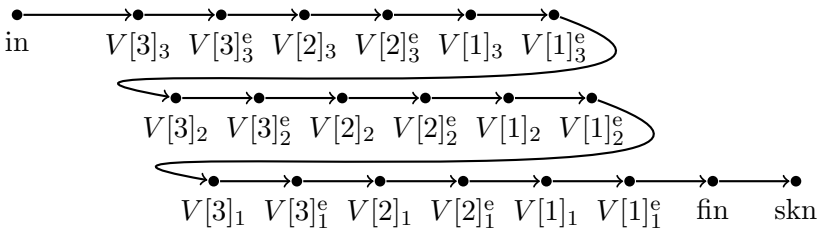

Figure 2. The membrane structure of the semi-uniform family to solve AGAP, in this case there are $m = |V| = 4$ vertices. The direction of the arrows ($\rightarrow$) indicates the movement of the objects through the structure by dissolving membranes. The root of the membrane structure is the "skin".

the particular number for this group of processors). Then, in parallel, $|V|$ processors check row v of the adjacency matrix, if a processor reads a 1 in location $e_{v,w}$ then it writes out $\langle v_{i+1}, w_{i+2}\rangle'$ to the w part of its output registers (see the proof of Theorem 2).

The other rules are computed (in parallel, for all i) from the number of vertices $|V|$, but with an additional check to see if the vertex is in the set A or equal to σ or t. Likewise the membrane structure is easily computable using a constant time CRAM.

The rules of each member of the membrane system family are specific for each problem instance so the solution is semi-uniform. The function h maps instances of $G = (V, E, A, s, t)$ to the membrane system Π_G using only constant time on a CRAM and so h is in $\mathbf{FAC}^0$. $\square$

5.1. Proof of correctness

The membrane systems of the family $\mathbf{\Pi}_{\mathrm{AGAP}}$ operate in two distinct phases, the output of the first phase acts as input to the second phase. The first phase records all paths through the graph and operates in a single membrane. The second phase uses these paths to evaluate apath(s, t).

We define the distance function $D_G : V \times V \rightarrow \mathcal{P}(\mathbb{N})$, where $D_G(\sigma, u)$ is the set of lengths of all directed paths from σ to u in the directed graph G.

5.1.1. *First phase*

The first phase of a system Π_G's computation produces objects which represent every (and only those) edge(s) in the graph G reachable from σ, as well as the distance(s) of each vertex from σ on all paths from σ.

This phase of the computation takes place in a single membrane labeled "in", the most deeply nested membrane (see Figure 2). This

phase begins with an object representing the edge (σ, s) and a counter object c_m. The counter c_m is decremented by Rules (F2) and dissolves the "in" membrane when it reaches c_0 via Rule (F3). While the counter is decrementing, Rules (F1) are acting on the objects representing edges, "following" all possible paths through the graph in parallel. Note that if there are different paths through the graph involving the same vertices, objects representing the same edges with different distances are generated. When the counter reaches c_0 it dissolves the membrane "in". This marks the end of the first phase of the algorithm and moves the computation to the second phase which is described in Section 5.1.2.

LEMMA 3. *Given a membrane system* Π_G*, where* $G = (V, E, A, s, t)$*, then in* $\leq |V| - 1$ *timesteps Rules* (F1) *produce the set of objects* $\{\langle u_i, v_{i+1}\rangle \mid (u, v) \in E$ *and* $i \in D_G(\sigma, u)\}$.

Proof. We prove by induction on i. The base case is $i = 0$. We show that, after the first timestep, Rules (F1) have evolved (a) the object $\langle \sigma_0, s_1\rangle$ and (b) the set of objects $\{\langle s_1, v_2\rangle' \mid \{(\sigma, s), (s, v)\} \subseteq E, 2 \in D_G(\sigma, v)\}$. The membrane "in" in the initial configuration of Π_G contains the object $\langle \sigma_0, s_1\rangle'$. Rules (F1) include:

$$\left[\langle \sigma_0, s_1\rangle' \rightarrow \langle \sigma_0, s_1\rangle, \left\{\langle s_1, v_2\rangle' \mid \forall v \text{ s.t. } \{(\sigma, s), (s, v)\} \subseteq E\right\}\right]_{\text{in}}$$

After applying these rules, if there are no edges leaving s in the graph G then only the object $\langle \sigma_0, s_1\rangle$ is created. Otherwise for each $(s, v) \in E$ a new edge object $\langle s_1, v_2\rangle'$ is evolved. The subscript on s is 1 since $D_G(\sigma, s) = \{1\}$ and the subscript(s) on all v is 2 since $2 \in D_G(\sigma, v)$.

For the induction hypothesis we assume that at the end of timestep i, Rules (F1) have evolved (a) the set of objects $\{\langle u_j, v_{j+1}\rangle \mid (u, v) \in E, 0 \leq j \leq i, j \in D_G(\sigma, u)\}$ and (b) the set of objects $\{\langle v_{i+1}, w_{i+2}\rangle' \mid \forall v \text{ s.t.}\{(u, v), (v, w)\} \subseteq E, i + 1 \in D_G(\sigma, v)\}$. Assuming these are the contents of "in" immediately before timestep $i + 1$, we prove that immediately after timestep $i + 1$ Rules (F1) evolve (a) the set of objects $\{\langle u_j, v_{j+1}\rangle \mid (u, v) \in E, 0 \leq j \leq i + 1, j \in D_G(\sigma, u)\}$ and (b) the set of objects $\{\langle v_{i+2}, w_{i+3}\rangle' \mid \forall v \text{ s.t.}\{(u, v), (v, w)\} \subseteq E, i + 2 \in D_G(\sigma, v)\}$. Rules (F1) include:

$$\left[\langle u_{i+1}, v_{i+2}\rangle' \rightarrow \langle u_{i+1}, v_{i+2}\rangle, \left\{\langle v_{i+2}, w_{i+3}\rangle' \mid \forall v \text{ s.t. } \{(u, v), (v, w)\} \subseteq E\right\}\right]_{\text{in}}$$

If there are no edges leaving v then, when the rules are applied, only objects $\langle u_{i+1}, v_{i+2}\rangle$ are created. Otherwise for each $(v, w) \in E$ a new primed object $\langle v_{i+2}, w_{i+3}\rangle'$ is evolved. The subscripts on vertices v and w come from the fact that $i + 2 \in D_G(\sigma, v)$ and $i + 3 \in D_G(\sigma, w)$. G is acyclic, hence the longest path length from σ is of length $< |V|$.

After timestep $|V| - 1$ the rules have produced the set of objects $\{\langle u_j, v_{j+1}\rangle \mid (u,v) \in E, 0 \leq j < |V|, j \in D_G(\sigma, u)\}$. $\square$

5.1.2. *Second phase*
The second phase of the computation begins when the parent of the membrane "in" contains the objects produced by the first phase.

Roughly speaking, this phase guides the computation along two nested loops by sequentially dissolving the membrane structure from the most deeply nested membrane to the skin membrane (see Figure 2). The outer loop iterates over all path lengths i from $|V| - 1$ down to 1, while the inner loop iterates over all vertices $v \in V \backslash \{\sigma\}$. Lemma 4 proves that each each iteration of the inner loop works correctly. Lemma 5 proves that the membrane structure guides the computation so that the correct solution is always produced.

DEFINITION 8. *Let $G = (V, E, A, s, t)$ be an* AGAP *instance, and let $\Pi_G \in \mathbf{\Pi}_{\mathrm{AGAP}}$. We say that vertex w is* evaluated *by the current configuration of Π_G for some $j \in \{1, 2, \ldots, |V| - 1\}$ if*

1. *there is a path of length j from vertex σ to vertex w in G (i.e. $j \in D_G(\sigma, w)$) and*

   (i) $w = t$, then the rules produce an object u^{Y}_{j-1} for all $(u, t) \in E$,

   (ii) $w \neq t$ and if $\mathrm{apath}(w, t)$ *holds, then the rules produce an object u^{Y}_{j-1} for all $(u, w) \in E$,*

   (iii) $w \neq t$ and if $\mathrm{apath}(w, t)$ *does* not *hold, then the rules produce an object u^{N}_{j-1} for all $(u, w) \in E$,*

2. *otherwise, no w^{Y}_j nor w^{N}_j objects are produced.*

Furthermore, the evaluation happens in the membrane labeled w_j and its parent w^{e}_j (its "evaluation" membrane) in the current configuration of Π_G. (When evaluating w at distance j some waste objects of the form $w^{\mathrm{Y}W}_j$ or $w^{\mathrm{N}W}_j$ may also be produced.)

DEFINITION 9. *We define* $\mathcal{C}_G(v,i)$ *to be the set*

$$\{\langle u_{j-1}, w_j\rangle \mid (u,w) \in E, u \neq \sigma, j \in D_G(\sigma,w), j < i\} \cup$$
$$\{\langle u_{i-1}, w_i\rangle \mid (u,w) \in E, w \in V_{\leq v}, u \neq \sigma, i \in D_G(\sigma,w)\} \cup$$
$$\{w_i^{\mathrm{Y}} \mid w \in V_{\leq v}, w \neq \sigma, i \in D_G(\sigma,w), (w,y) \in E, \mathrm{apath}(y,t) \textit{ holds}\} \cup$$
$$\{w_i^{\mathrm{N}} \mid w \in V_{\leq v}, w \neq \sigma, i \in D_G(\sigma,w), (w,y) \in E, \mathrm{apath}(y,t) \textit{ does not hold}\} \cup$$
$$\{w_{i-1}^{\mathrm{Y}} \mid w \in V_{>v}, i-1 \in D_G(\sigma,w), (w,y) \in E, \mathrm{apath}(y,t) \textit{ holds}\} \cup$$
$$\{w_{i-1}^{\mathrm{N}} \mid w \in V_{>v}, i-1 \in D_G(\sigma,w), (w,y) \in E, \mathrm{apath}(y,t) \textit{ does not hold}\} \cup$$
$$W, \textit{ where } W \subseteq \{y_i^{\mathrm{Y}W} \mid y \in V_{>v}\} \cup \{w_j^{\mathrm{Y}W} \mid w \in V, j > i\} \cup$$
$$\{y_i^{\mathrm{N}W} \mid y \in V_{>v}\} \cup \{w_j^{\mathrm{N}W} \mid w \in V, j > i\}$$

In our proofs, $\mathcal{C}_G(v,i)$ encodes G at an intermediary point in the evaluation of $\mathrm{apath}(\sigma,t)$. More precisely, given $\mathcal{C}_G(v,i)$, the following vertices have been evaluated: (a) all vertices $w \in V$ such that there is some $j > i$ where $j \in D_G(\sigma,w)$ and (b) all vertices $w \in V_{\geq v}$ such that $i \in D_G(\sigma,w)$. The set W is a subset of the possible "waste" objects which are generated by Π_G (proof of Lemma 4) as a side product.

LEMMA 4. *Let membrane system* $\Pi_G \in \mathbf{\Pi}_{\mathrm{AGAP}}$ *be in a configuration where the membrane with label* v_i *(*$v \in V\backslash\{\sigma\}$*,* $i \in D_G(\sigma,v)$*) contains the set of objects* $\mathcal{C}_G(v,i) \cup \{d_2\}$*, and nothing else. Then, in* ≤ 4 *timesteps,* Π_G *evaluates* v *at* i *according to Definition 8.* Π_G *does this by replacing each* $\langle u_{i-1}, v_i\rangle$ *object with the object* u_{i-1}^{Y} *if* $\mathrm{apath}(v,t)$ *holds and* u_{i-1}^{N} *if* $\mathrm{apath}(v,t)$ *does not hold. The membrane* v_i *and its parent "evaluation" membrane* v_i^{e} *are dissolved in the process.*

Proof. The proof is by *analysis* of the rules of Π_G. Let $v \in V\backslash\{\sigma\}$.

Case 1: v is universal ($v \in A$). In this case, the system uses the fact that if there exists $(v,y) \in E$ such that $\mathrm{apath}(y,t)$ does not hold, then $\mathrm{apath}(v,t)$ does not hold. The rules operate in ≤ 4 timesteps as follows.

Start: In membrane with label v_i where, $v \in A$.

if $v = t$ and there is an object $\langle u_{i-1}, t_i\rangle$ to trigger Rule (S4) **then** the membrane t_i is dissolved in timestep 1 evolving object u_{i-1}^{Y}. The counter decrements from d_2 to d_1 by Rule (S13). In timestep 2 the object d_1 dissolves the "evaluation" membrane v_i^{e} via Rule (S15). **End**: $\mathrm{apath}(t,t)$ *where* $i \in D_G(\sigma,t)$ *has been positively evaluated in 2 timesteps, the object* u_{i-1}^{Y} *is produced.*

else if $v = t$ and there are no objects $\langle u_{i-1}, t_i\rangle$ for Rule (S4) **then** any t_i^{N} or $t_i^{\mathrm{Y}W}$ objects dissolve t_i within 2 timesteps. Otherwise, in timestep 2 when the decrementing counter reaches d_0 by Rule (S13), it dissolves the membrane t_i via Rule (S14). In the next timestep the counter dissolves "evaluation" membrane v_i^{e} via Rule (S15).

End: *$i \notin D_G(\sigma, t)$ so* apath(t, t) *has not been evaluated, no new objects are produced and the computation continues.*

else if $v \neq t$ **then**

In timestep 1, any edge objects $\langle u_{i-1}, v_i \rangle$ evolve to $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ via Rule (S8). Any v_i^{Y} objects wait by evolving to v_i^{YW} via Rule (S10). The counter object decrements from d_2 to d_1 by Rule (S13).

if there is a v_i^{N} object to dissolve v_i via Rule (S11) **then**

apath(v, t) does not hold in the graph. The objects are now in the "evaluation" membrane v_i^{e} where any $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ objects evolve to u_{i-1}^{N} objects via Rule (S5), in timestep 2. The counter d_1 dissolves the "evaluation" membrane v_i^{e} via Rule (S15).

End: apath(v, t) *on paths of length $i \in D_G(\sigma, v)$ has been negatively evaluated in 2 timesteps, a u_{i-1}^{N} object is evolved for each $(u, v) \in E$.*

else if there is no v_i^{N} object to dissolve v_i via Rule (S11) **then**

in timestep 2 the objects remain in membrane v_i where any $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ objects evolve to $\langle u_{i-1}, v_i \rangle^{\mathrm{Y}}$ via Rule (S9).

The counter object decrements from d_1 to d_0 by Rule (S13).

if there is a v_i^{YW} object to dissolve v_i via Rule (S12) **then**

apath(v, t) holds in the graph. In timestep 3 the objects are now in the "evaluation" membrane v_i^{e} where $\langle u_{i-1}, v_i \rangle^{\mathrm{Y}}$ objects become u_{i-1}^{Y} objects via Rule (S6). The counter d_0 dissolves the in the "evaluation" membrane v_i^{e} via Rule (S15).

End: apath(v, t) *where $i \in D_G(\sigma, v)$ has been positively evaluated in 3 timesteps, a u_{i-1}^{Y} object is produced for every $(u, v) \in E$.*

else if there was no v_i^{YW} to dissolve v_i via Rule (S12) **then**

no edges leave v in the input graph so apath(v, t) does not hold. In timestep 3 any $\langle u_{i-1}, v_i \rangle^{\mathrm{Y}}$ objects evolve to $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ objects via (S7). The counter d_0 dissolves the membrane v_i via Rule (S14) and the objects move into the "evaluation" membrane v_i^{e}.

if each $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ object evolves to u_{i-1}^{N} via Rule (S5) **then**

in timestep 4 d_0 dissolves the "evaluation" membrane v_i^{e} via Rule (S15).

End: apath(v, t) *has been negatively evaluated in 4 timesteps since v is a sink node in G, a u_{i-1}^{N} object is produced for each $(u, v) \in E$.*

else if there were no $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ objects for Rule (S5) **then**

in timestep 4 d_0 dissolves the "evaluation" membrane v_i^{e} via Rule (S15).

End: $i \notin D_G(\sigma, v)$ *so* apath(v,t) *has not been evaluated, no new objects are produced and the computation continues.*

end if

end if

end if

end if

Case 2: v is existential ($v \in V \backslash (A \cup \{\sigma\})$). In this case, if there exists $(v, y) \in E$ where apath(y,t) holds, then apath(v,t) holds. Here the algorithm describing the action of the rules is very similar to the one above. The changes are to swap Rules (S8), (S10), (S11), (S12), with the existential Rules (S16), (S17), (S18), (S19), and in the sentences mentioning these rules, swap the roles of the objects tagged with "Y" and "N". (Note that in the existential case, no equivalent of Rule (S9) is needed, the edge objects are already in the form $\langle u_{i-1}, v_i \rangle^{\mathrm{N}}$ in timestep 2 by Rule (S7).)

Thus, assuming that membrane v_i ($v \in V \backslash \{\sigma\}$, $i \in D_G(\sigma, v)$) is the most deeply nested membrane in μ and contains the objects d_2 and $\mathcal{C}_G(v, i)$ then the rules of Π_G (in at most 4 timesteps) replace each $\langle u_{i-1}, v_i \rangle$ with a u_{i-1}^{Y} object if apath(v,t) holds and u_{i-1}^{N} if apath(v,t) does not hold, in the process the membrane v_i and its parent, the "evaluation" membrane v_i^{e}, are dissolved. Thus Π_G evaluates vertex v at distance i, according to Definition 8. □

LEMMA 5. *If membrane system* Π_G *contains the objects produced by the first phase (Lemma 3) in the parent of the membrane labeled "in" then within* $4(m-1)^2$ *timesteps* Π_G *the "fin" membrane contains the object* σ_0^{Y} *if* apath(s,t) *holds and the object* σ_0^{N} *if* apath(s,t) *does not hold.*

Proof. We prove by induction on the set of membranes, ordered by the membrane structure μ of Π_G, beginning at $V[m-1]_{m-1}$ and ending at $s_1 = V[1]_1$ (see Figure 2).

The base case is given by the membrane p of label $V[m-1]_{m-1}$. This membrane is the parent of "in" and, immediately after "in" is dissolved, p contains the set of objects $\mathcal{C}_G(V[m-1], m-1)$ which were generated by the first phase (see Lemma 3). Then, as per Lemma 4, Π_G replaces the set of objects of the form $\{\langle u_{m-2}, V[m-1]_{m-1} \rangle \mid (u, V[m-1]) \in E\}$ in $\mathcal{C}_G(V[m-1], m-1)$ with either an equal number of u_{m-2}^{Y} objects, or else with an equal number of u_{m-2}^{N} objects in ≤ 4 timesteps (the $V[m-1]_i^{\mathrm{Y}}$ and $V[m-1]_i^{\mathrm{N}}$ objects are removed or become waste objects). This modification of $\mathcal{C}_G(V[m-1], m-1)$ gives the set $\mathcal{C}_G(V[m-2], m-1)$ (see Definition 9). While the rules of Π_G are modifying the set of objects, they also dissolve membrane p (of label $V[m-1]_{m-1}$) and its parent

$p-1$ (of label $V[m-1]^{\mathrm{e}}_{m-1}$), placing the set $\mathcal{C}_G(V[m-2], m-1)$ in membrane $p-2$, which has label $V[m-2]_{m-1}$. This completes the base case.

For the induction hypothesis, assume that the set of objects $\mathcal{C}_G(V[z], i)$ is in membrane p of label $V[z]_i$. As per Lemma 4, Π_G replaces the set of objects of the form $\{\langle u_{i-1}, V[z]_i\rangle \mid (u, V[z]) \in E\}$ in the set $\mathcal{C}_G(V[z], i)$ with either an equal number of u^{Y}_{i-1} objects, or else with an equal number of u^{N}_{i-1} objects in ≤ 4 timesteps (the $V[z]^{\mathrm{Y}}_i$ and $V[z]^{\mathrm{N}}_i$ objects are removed or become waste objects). Then, we are in one of two cases.

Case 1: If $z > 1$ then this modification of $\mathcal{C}_G(V[z], i)$ gives the set $\mathcal{C}_G(V[z-1], i)$ (see Definition 9). While the rules of Π_G are modifying the set of objects, they also dissolve membrane p (with label $V[z]_i$), and its parent $p-1$ (with label $V[z]^{\mathrm{e}}_i$), placing the set $\mathcal{C}_G(V[z-1], i)$ in membrane $p-2$, which has label $V[z-1]_i$.

Case 2: If $z = 1$ then this modification of $\mathcal{C}_G(V[1], i)$ gives the set $\mathcal{C}_G(V[m-1], i-1)$ (see Definition 9). This is because the set $\mathcal{C}_G(V[1], i)$ must have moved up the membrane structure μ through each membrane $V[y]_i$ for $y \in \{m-1, m-2, \ldots, 1\}$ to arrive in membrane $V[1]_i$. In each membrane $V[y]_i$ the rules of Π_G operate via Lemma 4 to remove all objects in the set $\{\langle u_{i-1}, V[y]_i\rangle \mid (u, V[y]) \in E\}$ and replace them with either u^{Y}_{i-1} objects or u^{N}_{i-1} objects (the $V[y]^{\mathrm{Y}}_i$ and $V[y]^{\mathrm{N}}_i$ objects are also removed or become waste objects). This encodes that all vertices at distance i have now been evaluated, and the object set is ready for evaluation at distance $i-1$. While the rules of Π_G are modifying the set of objects, they also dissolve membrane p (with label $V[1]_i$), and its parent $p-1$ (with label $V[1]^{\mathrm{e}}_i$), placing the set $\mathcal{C}_G(V[m-1], i-1)$ in membrane $p-2$, which has label $V[m-1]_{i-1}$ (see Figure 2). This completes the inductive argument.

By the above inductive argument, system Π_G eventually reaches a configuration where the membrane of label $s_1 = V[1]_1$ contains the set of objects $\mathcal{C}_G(s, 1) = \{s^{\mathrm{Y}}_1 \mid (s, u) \in E$ where $\mathrm{apath}(u, t)$ holds$\} \cup \{s^{\mathrm{N}}_1 \mid (s, u) \in E$ where $\mathrm{apath}(u, t)$ does not hold$\} \cup \{\langle \sigma_0, s_1\rangle\}$. The rules of Π_G operate on the set $\mathcal{C}_G(s, 1)$ as in Lemma 4. The object $\langle \sigma_0, s_1\rangle$ is replaced with an object σ^{Y}_0 if $\mathrm{apath}(s, t)$ holds or is replaced by an object σ^{N}_0 if $\mathrm{apath}(s, t)$ does not hold. While the rules of Π_G are modifying the set of objects (Lemma 4), they also dissolve the membrane with label s_1, and its parent with label s^{e}_1, placing the object σ^{Y}_0 or σ^{N}_0 into membrane with label "fin".

At this point in the computation after $4(n-1)^2$ timesteps the rules of Π_G have used the objects produced in the first phase of the computation to evolve the object σ^{Y}_0 if $\mathrm{apath}(s, t)$ holds and the object σ^{N}_0 if $\mathrm{apath}(s, t)$ does not hold. □

Proof of Theorem 3. Lemma 5 shows that the object σ_0^{Y} is produced if apath(s,t) holds or else the object σ_0^{N} is produced if apath(s,t) does not hold.

All that remains is to show that Π_G correctly outputs either `yes` in the presence of σ_0^{Y}, or `no` in the absence of σ_0^{Y}. Rules (S20)–(S22) achieve this in ≤ 2 timesteps in the following straightforward manner. After Lemma 5, any σ_0^{Y} or σ_0^{N} objects are contained in the membrane of label "fin". All σ_0^{N} objects evolve to $\sigma_0^{\mathrm{N}W}$ objects in step 1. In the same step, the existence of a σ_0^{Y} object dissolves the membrane of label "fin", and places a `yes` in the output membrane. However, if there are no σ_0^{Y} objects, then, in step 2, a $\sigma_0^{\mathrm{N}W}$ object dissolves "fin", placing `no` in the output membrane. In either case, no more rules are applicable and the computation halts.

This completes the proof that the family $\mathbf{\Pi}_{\mathrm{AGAP}}$ recognises the **P**-complete language AGAP, in polynomial time. □

COROLLARY 2. $\mathbf{AC}^0$ *semi-uniform families of active membrane systems without charges* using only evolution and dissolution rules *characterise* **P**.

Proof. This follows from Theorem 3 and the known **P** upper-bound on active membrane systems without division [17]. □

6. $\mathbf{AC}^0$-uniformity and known PSPACE results

P-uniform families of active membrane systems (without charges) using non-elementary division (rules of type (f)) are known to characterise **PSPACE** [1, 15]. Clearly, the **PSPACE** upper-bound [15] is unaffected if we restrict to $\mathbf{AC}^0$-uniformity.

The lower-bound [1] is given by a **P**-uniform family of membrane systems that recognise instances of QSAT [10] in polynomial time. We sketch how this result can be modified so that it holds for Definition 3 using **L**-uniformity and, with a suitable restriction on the problem encoding, $\mathbf{AC}^0$-uniformity.

We use a restriction of QSAT, which we show is **PSPACE**-complete, where the number of variables n is even[2], and equal to the number of clauses m (i.e. $n = m = 2i, i \in \mathbb{N}$). Given an instance of QSAT in conjunctive normal form (CNF) we reduce to an instance where $n = m = 2i$ as follows. If $n < m$ we add $m - n$ "junk" variables $\{x_{n+1}, \ldots, x_m\}$

[2] The original solution requires an even number of variables. Most reasonable encodings represent variables such that it is possible check that there is an even number of them in $\mathbf{AC}^0$ (e.g. in unary).

to the instance. These new variables are not listed in the clauses and so do not affect the existence of a satisfying solution. If the total number of variables is now odd then we add another variable x_{m+1} and the tautological clause $(x_1 \vee \neg x_1)$ to φ. If $n > m$ we add $n - m$ copies of the tautological clause $(x_1 \vee \neg x_1)$ to φ, and ensure the number of variables and clauses are even as before. The input encoding in [1] (analogous to the e function in Definition 3) maps the variables in the clauses of the input instance to objects and is straightforward to compute in $\mathbf{FAC}^0$.

In Definition 3 the function $f : \{1\}^* \to \mathbf{\Pi}$ maps the encoded input instance length $1^{|x|}$, $x \in \Sigma^*$, to a membrane system in $\mathbf{\Pi}$. However, the construction in [1] makes use of both the number of variables and clauses to compute this mapping. The function f must now compute the number of variables (which is now equal to the number of clauses) from the length of its input, an instance of QSAT. The difficulty of this task is heavily dependent on the chosen encoding scheme, it can be accomplished in **FL** for most reasonable encodings and $\mathbf{FAC}^0$ for some.

Thus **L**-uniform families of $\mathcal{AM}^0$ systems using strong non-elementary division can solve at most **PSPACE**-complete problems (note that $\mathbf{L} \subsetneq \mathbf{PSPACE}$). If a suitable problem encoding scheme is used, the same result holds for $\mathbf{AC}^0$-uniform families of the same type.

7. Discussion

We have introduced $\mathbf{AC}^0$-uniformity to membrane systems, and active membrane systems in particular. This has allowed us to show an **NL** characterisation of semi-uniform systems without dissolution, an improvement over the previous **P** upper-bound. We also showed that a class of systems that was previously known to characterise **P** [17] retains its **P** lower-bound with $\mathbf{AC}^0$-semi-uniformity. Also a previous **PSPACE** [1, 15] characterisation also remains unchanged under the tighter uniformity conditions. This leads us to conclude that tight uniformity conditions, such as $\mathbf{AC}^0$-uniformity, allow us to more accurately study the computing power of our model.

We note that the system in Section 4 that solves the **NL**-complete problem STCON uses only evolution rules and that the system in Section 5 that solves the **P**-complete problem AGAP uses only dissolving and evolution rules. So in this setting, assuming $\mathbf{NL} \subsetneq \mathbf{P}$, dissolution rules significantly increase the computational power.

Acknowledgements

We would like to thank Mario J. Pérez-Jiménez and Agustín Riscos-Núñez and the other members of the Research Group on Natural Computing at the University of Seville for interesting discussions and for hosting Niall Murphy while later versions of this article were written. We would also like to thank Antonio E. Porreca for stimulating discussions about uniformity and the anonymous reviewers for their rigour in checking Section 5.

References

1. Alhazov, A. and M. J. Pérez-Jiménez: 2007, 'Uniform Solution to QSAT Using Polarizationless Active Membranes'. In: J. Durand-Lose and M. Margenstern (eds.): *Machines, Computations and Universality (MCU)*, Vol. 4664 of *LNCS*. Orléans, France, pp. 122–133.
2. Allender, E. and V. Gore: 1993, 'On strong separations from AC^0'. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **13**, 21–37.
3. Barrington, D. A. M., N. Immerman, and H. Straubing: 1990, 'On Uniformity within NC^1'. *Journal of Computer and System Sciences* **41**(3), 274–306.
4. Greenlaw, R., H. J. Hoover, and W. L. Ruzzo: 1995, *Limits to parallel computation:P-completeness Theory*. Oxford University Press.
5. Gutiérrez-Naranjo, M. A., M. J. Pérez-Jiménez, A. Riscos-Núñez, and F. J. Romero-Campero: 2006, 'Computational Efficiency of Dissolution Rules in Membrane Systems'. *International Journal of Computer Mathematics* **83**(7), 593–611.
6. Immerman, N.: 1988, 'Nondeterministic Space is Closed Under Complementation'. *SIAM Journal of Computing* **17**(5), 935–938.
7. Immerman, N.: 1989, 'Expressibility and parallel complexity'. *SIAM Journal on Computing* **18**(3), 625–638.
8. Murphy, N.: 2010, 'Uniformity conditions for membrane systems: Uncovering complexity below P'. Ph.D. thesis, National University of Ireland Maynooth.
9. Obtułowicz, A.: 2001, 'Note on some recursive family of P Systems with active membranes'. `http://ppage.psystems.eu/index.php/Papers`.
10. Papadimitriou, C. H.: 1993, *Computational Complexity*. Addison Wesley.
11. Păun, G.: 2001, 'P Systems with Active Membranes: Attacking NP-Complete Problems'. *Journal of Automata, Languages and Combinatorics* **6**(1), 75–90.
12. Păun, G.: 2002, *Membrane Computing. An Introduction*. Berlin: Springer-Verlag.
13. Păun, G.: 2005, 'Further twenty six open problems in membrane computing'. In: *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain), January 31st - February 4th*. pp. 249–262.
14. Pérez-Jiménez, M. J., A. Romero-Jiménez, and F. Sancho-Caparrini: 2003, 'Complexity classes in models of cellular computing with membranes'. *Natural Computing* **2**(3), 265–285.
15. Sosík, P. and A. Rodríguez-Patón: 2007, 'Membrane computing and complexity theory: A characterization of PSPACE'. *Journal of Computer and System Sciences* **73**(1), 137–152.

16. Szelepcsényi, R.: 1987, 'The method of forcing for nondeterministic automata'. *Bulletin of the EATCS* **33**, 96–99.
17. Zandron, C., C. Ferretti, and G. Mauri: 2000, 'Solving NP-Complete Problems Using P Systems with Active Membranes'. In: I. Antoniou, C. Calude, and M. Dinneen (eds.): *UMC '00: Proceedings of the Second International Conference on Unconventional Models of Computation*. London, UK, pp. 289–301.