

On acceptance conditions for membrane systems: characterisations of \mathbf{L} and \mathbf{NL}

Niall Murphy^{a,1}, Damien Woods^{b,2}

^a*Dept. of Computer Science, National University of Ireland, Maynooth, Ireland*

^b*Dept. of Computer Science & Artificial Intelligence, University of Seville, Spain*

Abstract

In this paper we investigate the affect of various acceptance conditions on recogniser membrane systems without dissolution. We demonstrate that two particular acceptance conditions (one easier to program, the other easier to prove correctness) both characterise the same complexity class, \mathbf{NL} . We also find that by restricting the acceptance conditions we obtain a characterisation of \mathbf{L} . We obtain these results by investigating the connectivity properties of dependency graphs that model membrane system computations.

Key words: P-System, recogniser, active membrane system, dependency graph, \mathbf{L} , \mathbf{NL}

1. Introduction

In the membrane systems (also known as P-systems [1]) computational complexity community it is common practice to explore the power of systems by allowing and prohibiting different developmental rules. This technique has yielded several interesting results such as the role of membrane dissolution in recognising \mathbf{PSPACE} -complete problems [2] and the role of membrane division in recognising problems outside of \mathbf{P} [3].

In this paper we do not vary the rules permitted in membrane systems but instead we vary the acceptance conditions and observe the change (or lack of change) this makes to the computing power of the system. Our main technique is to analyse the structure, and connectivity, of dependency graphs [2] that are induced by acceptance conditions. Our approach builds on previous work on dependency graphs [2, 4] to give a number of new techniques and results. Our techniques and results should be of interest to those who wish to characterise complexity classes, those studying acceptance conditions for membrane systems, and those characterising the power of membrane systems.

Email address: nmurphy@cs.nuim.ie (Niall Murphy)

¹Funded by the Irish Research Council for Science, Engineering and Technology

²Funded by Junta de Andalucia grant TIC-581

This research was motivated by the realisation that in prior work [5] we were using a seemingly more general halting condition than is used by the membrane community. Previously, we showed that \mathbf{AC}^0 -uniform families³ of active membrane systems without dissolution, and using the acceptance conditions specified in Section 3.1, characterise \mathbf{NL} [5]. However, most researchers use a more restricted acceptance condition (see Section 3.2). We show here that this more restricted definition also characterises \mathbf{NL} . This means that the two definitions are equivalent in terms of computing power for $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$ systems. The choice of which definition to use is now mostly a matter of personal taste as we have shown that the two are equivalent under \mathbf{AC}^0 reductions, i.e. there is a (very efficient) compiler to translate one definition to another.

In Section 3.3 we show that active membrane systems without dissolution, and using a restriction on the standard acceptance definition, characterise \mathbf{L} . This demonstrates that not all (minor) restrictions on halting definitions yield systems that characterise \mathbf{NL} .

We note here that the three definitions that we consider in Section 3 all characterise \mathbf{P} if they are generalised to use \mathbf{P} -uniformity. The \mathbf{P} lower bound of this characterisation is a trivial corollary of the fact that such membrane systems can easily embed polynomial time deterministic Turing machines, and is not related to the differences in their definitions.

2. Preliminaries

In this section we define membrane systems and some complexity classes. These definitions are based on those from Păun [1, 6], Sosík and Rodríguez-Patón [7], Gutiérrez-Naranjo et al. [2], and Pérez-Jiménez et al. [8]. Previous works on complexity and membrane systems spoke of solving a problem in a “uniform way”, that is, in a manner reminiscent of how families of circuits solve a problem. Sosík and Rodríguez-Patón defined uniformity for membrane systems in a similar manner to circuit uniformity, this allows us to refer to uniform families of membrane systems.

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 tuple $\Pi = (O, H, \mu, w_1, \dots, w_m, R)$ where,

1. $m \geq 1$ is the initial number of membranes;

³All membrane systems in this paper are \mathbf{AC}^0 -uniform and run for polynomial time.

2. O is the alphabet of objects;
3. H is the finite set of labels for the membranes;
4. μ is a membrane structure in the form of a tree, consisting of m membranes (nodes), labelled with elements of H . The parent of all membranes (the root node) is called the “environment” and has label $env \in H$;
5. w_1, \dots, w_m are strings over O , describing the multisets of objects placed in the m regions of μ .
6. R is a finite set of developmental rules, of the following forms:
 - (a) $[a \rightarrow u]_h$, for $h \in H, a \in O, u \in O^*$
 - (b) $a[]_h \rightarrow [b]_h$, for $h \in H, a, b \in O$
 - (c) $[a]_h \rightarrow []_h b$, for $h \in H, a, b \in O$
 - (d) $[a]_h \rightarrow b$, for $h \in H, a, b \in O$
 - (e) $[a]_h \rightarrow [b]_h [c]_h$, for $h \in H, a, b, c \in O$.
 - (f) $[a []_{h_1} []_{h_2} []_{h_3}]_{h_0} \rightarrow [b []_{h_1} []_{h_3}]_{h_0} [c []_{h_2} []_{h_3}]_{h_0}$,
for $h_0, h_1, h_2, h_3 \in H, a, b, c \in O$.

These rules are applied according to the following principles:

- All the rules are applied in a maximally parallel manner. That is, in one step, one object of a membrane is used by at most one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form, must evolve.
- If at the same time a membrane labelled with h is divided by a rule of type (e) or (f) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. This process takes only one step.
- The rules associated with membranes labelled with h are used for membranes with that label. At one step, a membrane can be the subject of only one rule of types (b)–(f).
- Rules of type (f) are division rules for non-elementary membranes. These rules allow us duplicate an entire branch of the membrane structure in the following manner. If the membrane (label h_0) to which the non-elementary division rule is applied contains objects and child membranes then copies of those membranes and all of their contents (including their own child membranes) are found in both resulting copies of h_0 .

2.2. Recogniser membrane systems

In this paper one of our goals is to unify and clarify definitions for language recognising variants of membrane systems. To achieve this, we consider three different notions of acceptance for recogniser systems, one in each of Sections 3.1 to 3.3. Each of these three definitions is a restriction on the general (and purposely vague) Definition 2 below.

We recall from [2] that a computation of the system is a sequence of configurations such that each configuration (except the initial one) is obtained from the previous one by a transition. A computation that reaches a configuration where no more rules can be applied to the existing objects and membranes is called a halting computation.

Definition 2. A *recognizer membrane system* is a membrane system with external output (that is, the results of halting computations are encoded in the environment) such that:

1. the working alphabet contains two distinguished elements **yes** and **no**;
2. if C is a computation of the system, then it is either an accepting or a rejecting computation.

This definition is vague since we have not defined accepting and rejecting computations. In Section 3 we show the set of problems that a membrane system accepts when using various notions of accepting (or rejecting) computations.

2.3. Complexity classes

Consider a decision problem X , i.e. a set of instances $X = \{x_1, x_2, \dots\}$ over some finite alphabet such that to each x_i there is a unique answer “yes” or “no”. We say that a *family* of membrane systems solves a decision problem if each instance of the problem is solved by some family member. We denote by $|x| = n$ the length of any instance $x \in X$. Throughout this paper, \mathbf{AC}^0 circuits are **DLOGTIME**-uniform, polynomial sized (in input length n), constant depth, circuits with AND, OR and NOT gates, and unbounded fanin [9]. The complexity class **L** (**NL**) is the set of problems solved by (non-)deterministic Turing machines using only $O(\log n)$ space, where n is the length of the input instance.

Definition 3. Let \mathcal{D} be a class of membrane systems and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a total function. The class of *problems solved by \mathbf{AC}^0 -uniform families of membrane systems* of type \mathcal{D} in time f , denoted $(\mathbf{AC}^0)\text{-MC}_{\mathcal{D}}(f)$, contains all problems X such that:

- There exists an \mathbf{AC}^0 -uniform family of membrane systems, $\Pi_X = (\Pi_X(1), \Pi_X(2), \dots)$ of type \mathcal{D} : that is, there exists an \mathbf{AC}^0 circuit family such that on unary input 1^n the n^{th} member of the circuit family constructs $\Pi_X(n)$.
- There exists an \mathbf{AC}^0 circuit family such that on input $x \in X$, of length $|x| = n$, the n^{th} member of the family encodes x as a multiset of input objects placed in the distinct input membrane of $\Pi_X(n)$.
- Π_X is *sound* and *complete* with respect to problem X : $\Pi_X(n)$ starting with an encoding of input $x \in X$ of length n accepts iff the answer to x is “yes”.

- Π_X is f -efficient: $\Pi_X(n)$ always halts in at most $f(n)$ steps.

Definition 3 describes \mathbf{AC}^0 -uniform families and we generalise this to define \mathbf{AC}^0 -semi-uniform families of membrane systems $\Pi_X = (\Pi_X(x_1); \Pi_X(x_2); \dots)$ where there exists an \mathbf{AC}^0 circuit family which, on an input $x \in X$, constructs membrane system $\Pi_X(x)$. Here a single circuit family (rather than two) is used to construct the semi-uniform membrane family, and so the problem instance is encoded using objects, membranes, and rules. In this case, for each instance of $x \in X$ we have a special membrane system which does not need a separately constructed input. The resulting class of problems is denoted by $(\mathbf{AC}^0)\text{-MC}_{\mathcal{D}}^*(f)$. Obviously, $(\mathbf{AC}^0)\text{-MC}_{\mathcal{D}}(f) \subseteq (\mathbf{AC}^0)\text{-MC}_{\mathcal{D}}^*(f)$ for any given class \mathcal{D} and a valid [10] complexity function f .

We define $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{D}}$ and $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{D}}^*$ as

$$(\mathbf{AC}^0)\text{-PMC}_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} (\mathbf{AC}^0)\text{-MC}_{\mathcal{D}}(n^k),$$

and

$$(\mathbf{AC}^0)\text{-PMC}_{\mathcal{D}}^* = \bigcup_{k \in \mathbb{N}} (\mathbf{AC}^0)\text{-MC}_{\mathcal{D}}^*(n^k).$$

In other words, $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{D}}$ (and $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{D}}^*$) is the class of problems solvable by uniform (respectively semi-uniform) families of membrane systems in polynomial time. We let \mathcal{AM}^0 denote the class of membrane systems with active membranes and no charges. We let $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$ denote the class of problems solvable by \mathbf{AC}^0 -semi-uniform families of membrane systems in polynomial time with no dissolution rules. In an abuse of notation, we often let $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$ refer to the class of such membrane systems (rather than problems). For brevity we often write Π_X instead of $\Pi_X(n)$ or $\Pi_X(x)$.

Remark 4. A membrane system is *confluent* if it is both sound and complete. That is a Π_X is *confluent* if all computations of Π_X with the same input give the same result; either always accepting or else always rejecting.

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

2.4. Dependency graphs and normal forms

The *dependency graph* (first introduced by Gutiérrez-Naranjo et al. [2]) is an indispensable tool for characterising the computational complexity of membrane systems without dissolution. This technique is reminiscent of configuration graphs for Turing Machines. Similarly to a configuration graph, a dependency graph helps visualise a computation. However, it differs in its approach by representing a membrane system configuration as a set of nodes rather than as a single node in configuration space.

Looking at membrane systems without dissolution as dependency graphs allows us to employ the existing, mature corpus of techniques and complexity results for graph problems. As we show in this paper, this greatly simplifies the process of proving upper and lower bounds for such systems. A key technique we use in this paper is to transfer from a dependency graph to a new membrane system, $\Pi \rightarrow \mathcal{G}_\Pi \rightarrow \Pi_{\mathcal{G}_\Pi}$. This new system accepts iff the original membrane system accepts, since their dependency graphs are isomorphic. Also, the new system is considerably simplified as it uses only one membrane (the environment) and all rules are of type (a). This is used as a normal form for membrane systems without dissolution.

In Sections 3.1 to 3.3 we define reachability problems for dependency graphs such that if the answer to the graph reachability problem is yes, then the membrane system it represents is an accepting system. This is because the nodes of a dependency graph represent an object being in a certain membrane, and an edge between two nodes represents a developmental rule that causes that object to be in that membrane. Thus if the object **yes** arrives in the environment (the acceptance signal) of the membrane system, then there is a directed path leading from one special node (**in**) to another special node (**yes**) in the dependency graph. For more details about how a dependency graph is constructed and its proof of correctness see Gutiérrez-Naranjo et al. [2, 4].

The dependency graph for a membrane system Π is a directed graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \mathbf{in}, \mathbf{yes}, \mathbf{no})$ where $\mathbf{in} \subseteq V_{\mathcal{G}}$ represents the input multiset, and $\mathbf{yes}, \mathbf{no} \in V_{\mathcal{G}}$, represent the accepting and rejecting signals respectively. Each vertex $a \in V_{\mathcal{G}}$ is a pair $a = (o, h) \in O \times H$, where O is the set of objects in Π and H is the set of membrane labels in Π . An edge (a, b) exists iff there is a developmental rule in Π such that the left hand side of the rule has the same object-membrane pair as a and the right hand side has an object-membrane pair matching b . Since there is no membrane dissolution allowed, the parent/child relationships of membranes does not change during the computation. This allows us to determine the correct parent and child membranes for type (b) and type (c) rules.

Previously [2], the graph \mathcal{G} was constructed from Π in polynomial time. We make the observation that the graph \mathcal{G} can be constructed in \mathbf{AC}^0 . We use a common circuit technique known as “masking” whereby using AND gates and a desired pattern we filter out the bits of the input string that we are interested in. We take as input a binary string x that encodes a membrane system, Π . To make a dependency graph from a membrane system requires a constant number of parallel steps that are as follows. First, a row of circuits identifies all type (b) and (c) rules and uses the membrane structure to determine the correct parent membranes, then writes out (a binary encoding of) edges representing these rules. Next, a row of circuits writes out all edges representing type (e) and (f) rules (see [2] for more details about the representation of these rules in dependency graphs). For (a) rules it is possible to have polynomially many copies of polynomially many distinct objects on the right hand side of a rule. To write out edges for these rules in constant time we take advantage of the fact that we require at most one edge for each object-membrane pair in $O \times H$. We have a circuit for each element of $\{o_h \mid o \in O, h \in H\}$. The circuit for o_h

takes as input (an encoding of) all rules in R whose left hand side is of the form $[o]_h$. The circuit then, in a parallel manner, masks (an encoding of) the right hand side of the rule (for example $[bbcdc]_h$) with the encoding of each object in O , (in the example, masking for (encoded) b would produce (encoded) $bb000$). All encoded objects in the string are then ORed together so that if there was at least one copy of that object in the system we obtain a single instance of it. The circuit being unique for a specific left hand side $[o]_h$ now writes out an encoding of the edge (o_h, b_h) and an encoding of all other edges for objects that existed on the right hand side of this rule in parallel.

Remark 5. Of course one can take the opposite view. We observe that to convert a dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ into a new membrane system, $\Pi_{\mathcal{G}}$, we simply convert the edges of the graph into object evolution rules. The set of objects of $\Pi_{\mathcal{G}}$ is $O_{\mathcal{G}} = V_{\mathcal{G}}$. The rules of $\Pi_{\mathcal{G}}$ are $\{[v \rightarrow S(v)]_{env} \mid \forall v \in V_{\mathcal{G}}\}$ where $S(v) = \{s \in V_{\mathcal{G}} \mid (v, s) \in E_{\mathcal{G}}\}$. The nodes **in**, **yes**, **no** become the input multiset, **yes** object, and **no** object respectively. We compute this in \mathbf{AC}^0 .

This new membrane system, $\Pi_{\mathcal{G}}$, highlights some points about active membrane systems without dissolution. These give rise to significant simplifications and normal forms.

Lemma 6. *Any (\mathbf{AC}^0) - $\text{PMC}^*_{\mathcal{AM}^0_d}$, Π , with m membranes can be simulated by a (\mathbf{AC}^0) - $\text{PMC}^*_{\mathcal{AM}^0_d}$ system, Π' , that (1) has no membranes other than the environment and (2) uses only rules of type (a).*

By *simulate* we mean that the latter system accepts on input **in** iff the former does. To see that Lemma 6 holds, first notice how the dependency graph represents an (object, label) pair as a single node. Also if we convert the dependency graph \mathcal{G} into a membrane system $\Pi_{\mathcal{G}}$, (1) it uses a single membrane with label *env*, and each node is modelled by a single object. (2) Each edge in \mathcal{G} becomes a rule of type (a). Notice that the dependency graphs of Π and $\Pi_{\mathcal{G}}$ are isomorphic.

Lemma 7. *Any (\mathbf{AC}^0) - $\text{PMC}^*_{\mathcal{AM}^0_d}$ system, Π , which has, as usual, multisets of objects in each membrane can be simulated by another (\mathbf{AC}^0) - $\text{PMC}^*_{\mathcal{AM}^0_d}$ system, Π' , which has sets of objects in each membrane.*

We verify Lemma 7 by observing that in a dependency graph, \mathcal{G} , the multiset of objects is encoded as a *set* of vertices, no information is kept regarding object multiplicities. Thus when \mathcal{G} is converted into a new membrane system, $\Pi_{\mathcal{G}}$, there are no rules with a right hand side with more than one instance of each object. The resulting system $\Pi_{\mathcal{G}}$ accepts iff Π accepts since the dependency graphs of both systems are isomorphic. Thus object multiplicities do not affect whether the system accepts or rejects.

3. Three different acceptance conditions

Here we present three different acceptance conditions for membrane systems with active membranes and show what complexity class they characterise. We define each acceptance condition; define a graph reachability problem that models the computation of such a system; then prove both upper and lower bounds on the computational power of the system. Each of Definitions 8, 13, 19, is a more concrete replacement for Definition 2. Most results in this section use reductions to and from reachability problems on membrane dependency graphs. Solving these reachability problems is equivalent to simulating such a membrane system since we translate (via \mathbf{AC}^0 reductions) from a membrane system to a corresponding reachability problem, and vice-versa.

3.1. General recogniser systems characterise \mathbf{NL}

In previous works [5, 11] we used a definition of recogniser membrane systems that is more general than is typical of other work in the area (i.e. Section 3.2). In this more general definition it is possible for the membrane system to output both **yes** and **no** symbols. However, when the first of these symbols is produced we call it the accepting/rejecting step of the computation. (Note that it is forbidden for both **yes** and **no** to be produced in the same timestep.) We now define this acceptance condition and then go on to show that $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$ systems with this acceptance condition characterise \mathbf{NL} .

Definition 8. A *general recognizer membrane system*, Π , is a membrane system with external output (that is, the results of halting computations are encoded in the environment) such that:

1. the working alphabet contains two distinguished elements **yes** and **no**;
2. if C is a computation of the system, (i) then a **yes** or **no** object is released into the environment, (ii) but not in the same timestep. If **yes** is released before **no** then the computation is accepting, otherwise the computation is rejecting.

We now define the reachability problem for $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$ systems whose acceptance conditions are as in Definition 8. Solving this problem is equivalent (via a reduction) to simulating such a system.

Problem 9 (GENREC).

Instance: A dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ where $\{\text{in}, \text{yes}, \text{no}\} \subseteq V_{\mathcal{G}}$, representing the rules of a general recogniser membrane system Π as defined in Definition 8.

Problem: Is the shortest directed path from **in** to **yes** of length less than the shortest directed path from **in** to **no**?

We also define the problem \mathbf{STCON} , the canonical \mathbf{NL} -complete problem [12]. This problem is also known as \mathbf{PATH} , $\mathbf{REACHABILITY}$, and \mathbf{GAP} .

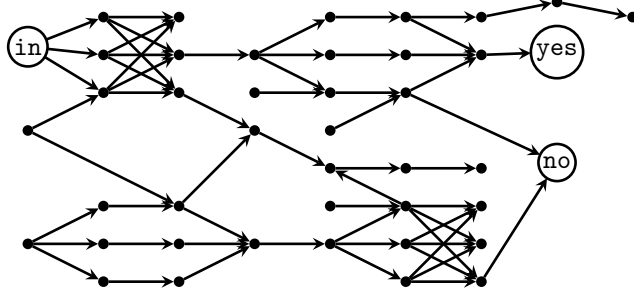


Figure 1: An example dependency graph \mathcal{G} for some unspecified *general recogniser membrane system* (Definition 8). Note that this represents a rejecting computation since the minimum directed path from **in** to **no** is of length 6, while the minimum directed path from **in** to **yes** is of length 7.

Problem 10 (STCON).

Instance: A directed acyclic graph $G = (V, E, s, t)$ where $\{s, t\} \subseteq V$.

Problem: Is there a directed path in G from s to t ?

We now provide a result which is used to show that $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d^0}^*$ systems whose acceptance conditions are as in Definition 8 characterise **NL** (this characterisation has been published elsewhere [5], we present a shorter proof here).

Theorem 11. GENREC is **NL**-complete

PROOF. First we show $\text{STCON} \leq_{\mathbf{AC}^0} \text{GENREC}$. Given an instance $G = (V, E, s, t)$ of STCON, we construct a dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \mathbf{in}, \mathbf{yes}, \mathbf{no})$ such that $V_{\mathcal{G}} = V \cup \{\mathbf{no}\}$ and $E_{\mathcal{G}} = E$. We replace all instances of s with **in**, and t with **yes**, in \mathcal{G} . Clearly there is a path from **in** to **yes** iff there is a path from s to t in G . We also add a directed path of length $|V| + 1$ from **in** to **no** in \mathcal{G} . This ensures that if there is not a path from s to t in G , then **no** is reached after all other paths have terminated. This reduction is computed in \mathbf{AC}^0 .

We now prove the correctness of the above reduction. Since GENREC is defined in terms of the general recogniser membrane systems (Definition 8), we often appeal to Definition 8 in the proof. Recall that, via Remark 5, we can translate \mathcal{G} to a membrane system $\Pi_{\mathcal{G}}$ in \mathbf{AC}^0 .

- By adding a path of length $|V| + 1$ from **in** to **no** we are guaranteeing that object **no** is not produced by the membrane system $\Pi_{\mathcal{G}}$ at the same time as *any* other object, this satisfies point 2(ii) of Definition 8.
- If there is a path from s to t in G (and **yes** is evolved in $\Pi_{\mathcal{G}}$) the reduction ensures that a path from **in** to **yes** exists in \mathcal{G} . Also in either case a path from **in** to **no** is created by the reduction that ensures the correct output from $\Pi_{\mathcal{G}}$. Thus we satisfy point 2(i) of Definition 8.

We now show that $\text{GENREC} \in \mathbf{NL}$. Let M be a non-deterministic Turing machine with two variables x and y . Finding the shortest path between two nodes is well known to be computable in \mathbf{NL} via $\leq n$ iterations of a STCON algorithm. Set x to be the shortest path from **in** to **yes**. Set y to be the shortest path from **in** to **no**. If $x < y$, M accepts, otherwise M rejects. Thus M uses a non-deterministic algorithm and two binary counters to solve GENREC and so the problem is in \mathbf{NL} . \square

Theorem 12. \mathbf{NL} is characterised by $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d}^*$ using the general acceptance conditions from Definition 8.

The proof is omitted, but can be obtained by using standard techniques along with Remark 5, Theorem 11, and Definition 8.

3.2. Standard recogniser membrane systems characterise \mathbf{NL}

In this section we discuss the “standard” definition for recogniser membrane systems, i.e. the definition that most researchers use when proving results about recogniser membrane systems. On a given input, these systems produce either a **yes** object or a **no** object, but not both. Also it is assumed that this occurs in the last timestep of the computation where no other rules are applicable.

By showing an \mathbf{NL} characterisation for such systems, we are showing that this definition has equal power to the more general definition discussed above in Section 3.1. Furthermore, we have provided a “compiler,” via reductions, to translate a system that uses the general definition into a system that uses the standard definition. This is significant since the general definition is often easier to program, while it is often easier to prove certain properties (such as correctness) for the standard definition. We begin with a definition of standard recogniser membrane systems from Gutiérrez-Naranjo et al. [2].

Definition 13 ([2]). A *recognizer membrane system*, Π , is a membrane system with external output (that is, the results of halting computations are encoded in the environment) such that:

1. the working alphabet contains two distinguished elements **yes** and **no**;
2. all computations halt; and
3. if C is a computation of the system, then (i) either object **yes** or object **no** (but not both) must have been released into the environment, and (ii) only in the last step of the computation. If **yes** is released then the computation is accepting, otherwise the computation is rejecting.

Remark 14. Definition 13 affects the dependency graph of such systems so that we can define the following subsets of the objects O .

$$O_{\text{yes}} = \{o \mid o \in O \text{ and } o \text{ eventually evolves } \text{yes}\},$$

$$O_{\text{no}} = \{o \mid o \in O \text{ and } o \text{ eventually evolves } \text{no}\}, \text{ and } O_{\text{other}} = O \setminus (O_{\text{yes}} \cup O_{\text{no}}).$$

Lemma 15. $O_{\text{yes}} \cap O_{\text{no}} = \emptyset$.

PROOF. Assume that object $o \in O_{\text{yes}} \cap O_{\text{no}}$, this implies that both a **yes** and a **no** object are produced by the confluent system on a given input which contradicts point 3(i) of Definition 13. \square

These observations are illustrated in Figure 2.

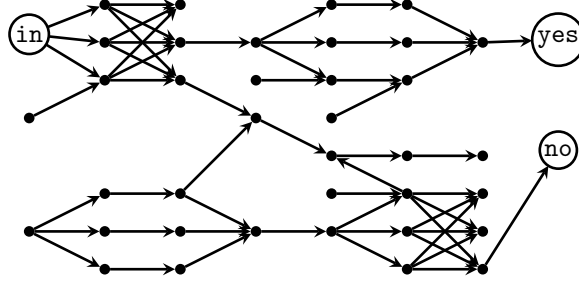


Figure 2: An example dependency graph \mathcal{G} for some unspecified standard recogniser membrane system (Definition 13). Note that via Lemma 15 there are no directed paths from O_{yes} to O_{no} , they are *weakly connected*.

We now define the reachability problem for $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d}^*$ systems whose acceptance conditions are as in Definition 13. We remind the reader that these systems are confluent via Definition 3 and Remark 4.

Problem 16 (STDREC).

Instance: A dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ where $\{\text{in}, \text{yes}, \text{no}\} \subseteq V_{\mathcal{G}}$, representing the rules of a $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d}^*$ recogniser membrane system Π as defined in Definition 13.

Problem: Is there a directed path from **in** to **yes**?

We now provide the main result needed to show that standard $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d}^*$ characterises **NL**.

Theorem 17. *STDREC is NL-complete.*

PROOF. First we show $\text{STCON} \leq_{\mathbf{AC}^0} \text{STDREC}$. Given an instance $G = (V, E, s, t)$ of STCON, we construct a dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ such that $V_{\mathcal{G}} = V \cup \{\text{yes}, \text{no}\}$ and $E_{\mathcal{G}} = E$. We replace s with **in** in \mathcal{G} . We add a directed path of $|V| + 1$ edges leading from t to **yes** to ensure that all other computations have halted before **yes** is evolved. Clearly there is a path from **in** to **yes** in \mathcal{G} iff there is a path from s to t in graph G .

So far, \mathcal{G} we have shown that $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d}^*$ recogniser membrane systems, as in Definition 13, *accept* words in STCON. However, the construction does not explicitly say how to reject words that are not in the language, which is a requirement of Definition 13. We extend the proof as follows. Let $\overline{\text{STCON}}$ be the complementary problem to STCON, i.e. given an acyclic graph G' is

there no directed path from s' to t' ? $\overline{\text{STCON}}$ is **coNL**-complete (via the same reduction that is used to show the **NL**-completeness of **STCON**), and so is also **NL**-complete (since $\text{NL} = \text{coNL}$ [13, 14]). Now we define a third **NL**-complete problem **STCON**– $\overline{\text{STCON}}$; the set of graphs with two disjoint components G, G' that are related in the following sense: s eventually yields t in G iff s' does not eventually yield t' in G' . Now we reduce this graph to a dependency graph \mathcal{G} in a similar manner as the above reduction. That is, we place an edge from **in** to s and from **in** to s' . We add a directed path of $|V| + 1$ edges leading from t to **yes**, and another directed path of $|V| + 1$ edges leading from t' to **no**. Then the induced membrane system $\Pi_{\mathcal{G}}$ correctly decides **STCON**– $\overline{\text{STCON}}$ since it answers **yes** iff s leads to t , otherwise it answers **no**. This reduction is computed in \mathbf{AC}^0 .

We now prove the correctness of the above reduction. Recall that, via Remark 5, we translate \mathcal{G} to a membrane system $\Pi_{\mathcal{G}}$ in \mathbf{AC}^0 .

- Since an instance of **STCON**– $\overline{\text{STCON}}$ is an acyclic graph we trivially satisfy point 2 of Definition 13.
- In the induced membrane system $\Pi_{\mathcal{G}}$ the node **in** can only lead to one of **yes** or **no**, but not both, since the embedded **STCON** and $\overline{\text{STCON}}$ problems are complementary. This satisfies point 3(i) of Definition 13.
- $\Pi_{\mathcal{G}}$ outputs (either **yes** or **no**) in the last step because we add $|V| + 1$ extra edges from t and t' so that the accepting or rejecting path is the longest in the dependency graph, satisfying point 3(ii) of Definition 13.

Now we show that $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$, as in Definition 13, can recognise no more than **NL** by showing that $\text{STDREC} \leq_{\mathbf{AC}^0} \text{STCON}$. We observe that an instance of **STDREC** is a directed acyclic graph (via point 2 of Definition 13). Given an instance $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ of **STDREC**, we construct $G = (V, E, s, t)$ such that $V = V_{\mathcal{G}}$ and $E = E_{\mathcal{G}}$ and replace all instances of **in** with s and **yes** with t in \mathcal{G} . Clearly there is a path from s to t in G iff there is a path from **in** to **yes** in the dependency graph \mathcal{G} . This reduction is computed in \mathbf{AC}^0 . \square

Theorem 18. ***NL** is characterised by $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}^0_d}^*$ using the standard acceptance conditions from Definition 13.*

The proof is omitted, but can be obtained by using standard techniques along with Remark 5, Theorem 17, and Definition 13.

3.3. Restricted recogniser membrane systems characterise **L**

We now consider a restriction on the standard definition of recogniser membrane systems. Above in Section 3.2, we forbid an object that eventually yielded a **yes** from also yielding a **no** (and vice versa). Now we further restrict the system and require that *all* descendent nodes of **in** must eventually yield **yes**, or all must eventually yield **no**. Notice that this restriction forbids objects that

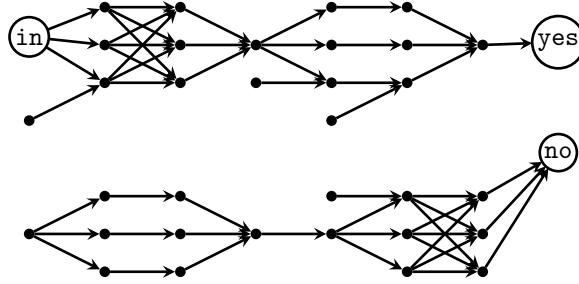


Figure 3: An example dependency graph \mathcal{G} for some unspecified restricted recogniser membrane system (Definition 19).

do not contribute to the final answer (accept or reject) and forbids rules of the form $[a \rightarrow \lambda]$ where λ is the empty word.

Definition 19. A *restricted recogniser membrane system*, Π , is a membrane system with external output (that is, the results of halting computations are encoded in the environment) such that:

1. the working alphabet contains two distinguished elements **yes** and **no**;
2. all computations halt;
3. if C is a computation of the system, then (i) either object **yes** or object **no** (but not both) must have been released into the environment, and (ii) only in the last step of the computation. If **yes** is released then the computation is accepting, otherwise the computation is rejecting.
4. each object $o \in O$ must, via a sequence of zero or more developmental rules, lead to **yes**, or else lead to **no**, but not both.

The definition has the following effect on the dependency graph.

Remark 20. Since every object eventually yields exactly one **yes**, or exactly one **no**, the graph \mathcal{G} consists of exactly two disjoint components.

We now define a graph reachability problem for $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d^0}^*$ systems whose acceptance conditions are as in Definition 19.

Problem 21 (RSTREC).

Instance: A dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ where $\{\text{in}, \text{yes}, \text{no}\} \subseteq V_{\mathcal{G}}$, representing the rules of an $(\mathbf{AC}^0)\text{-PMC}_{\mathcal{AM}_d^0}^*$ recogniser membrane system Π as defined in Definition 19.

Problem: Is there a directed path from **in** to **yes**?

We define the **L**-complete problem DIRECTED FOREST ACCESSIBILITY (DFA) [15].

Problem 22 (DFA [15]).

Instance: An acyclic directed graph $G = (V, E, s, t)$ where $\{s, t\} \subseteq V$ and each node is of out-degree 0 or 1.

Property: Is there a directed path from s to t ?

Theorem 23. RSTREC is \mathbf{L} -complete

PROOF. First we show $\text{DFA} \leq_{\mathbf{AC}^0} \text{RSTREC}$. Given an instance $G = (V, E, s, t)$ of DFA, we construct a dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ such that $V_{\mathcal{G}} = V \cup \{\text{no}\}$ and $E_{\mathcal{G}} = E \setminus \{(t, v) \mid v \in V\}$. We also replace s with **in**, and add a directed path of length $|V| + 1$ from t to **yes** in \mathcal{G} . Clearly there is a path from **in** to **yes** in \mathcal{G} iff there is a path from s to t in graph G . Note that since we removed the edge (if it exists) leaving t , every computation halts (in the induced membrane system $\Pi_{\mathcal{G}}$) upon evolving **yes**. We also add an edge from all nodes, except **yes**, of out-degree 0 to **no**. There is now a path from **in** to **no** iff there is no path from s to t in G because all paths that do not lead to **yes** now lead to **no**. This reduction is computed in \mathbf{AC}^0 .

We now prove the correctness of the above reduction. Recall that, via Remark 5, we translate \mathcal{G} to a membrane system $\Pi_{\mathcal{G}}$ in \mathbf{AC}^0 .

- Since G (as a forest) is acyclic, our reduction ensures \mathcal{G} , and hence any computation of $\Pi_{\mathcal{G}}$, is acyclic also, satisfying point 2 of Definition 19.
- Our reduction ensures that exactly 2 nodes in \mathcal{G} have out-degree 0, the (sink) nodes **yes** and **no**, this implies that the only objects that have no applicable rules in $\Pi_{\mathcal{G}}$ are **yes** and **no**. This satisfies points 1 and 3(ii) of Definition 19.
- Since every node in G has out-degree 0 or 1, then every node in \mathcal{G} has out-degree 0 or 1 (and every object in $\Pi_{\mathcal{G}}$ has 0 or 1 applicable developmental rules). Combined with the previous point, this implies that all nodes in \mathcal{G} are on a path to either **yes** or **no**, and that all objects in $\Pi_{\mathcal{G}}$ eventually yield either **yes** or **no**, satisfying points 4 and 3(i) of Definition 19.

Now we show RSTREC is contained in \mathbf{L} by outlining a deterministic logspace Turing machine M that decides RSTREC. The input tape of M encodes an instance $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \text{in}, \text{yes}, \text{no})$ of RSTREC. Starting with the input node **in**, M stores this node in a variable called x on its work tape. If x is neither **yes** nor **no** then M searches the set of edges $E_{\mathcal{G}}$ on its input tape, upon finding an edge (x, v) , the machine sets x to be v (overwriting the previous value). The computation carries on in this fashion until either x equals **no** causing M to reject, or **yes**, in which case M accepts.

The algorithm correctly decides RSTREC because each node in the data-structure has out-degree 0 or 1 and we simply trace along a path until we reach a sink. If the sink is **yes**, we accept, otherwise we reject. Since only one node is stored on M 's work tape at any time, M uses $O(\log n)$ space (where n is the input length). Thus $\text{RSTREC} \in \mathbf{L}$. \square

Theorem 24. \mathbf{L} is characterised by $(\mathbf{AC}^0)\text{-PMC}^*_{\mathcal{AM}^0_d}$ using the restricted acceptance conditions from Definition 19.

The proof is omitted, but can be obtained by using standard techniques along with Remark 5, Theorem 23, and Definition 19.

4. Conclusions

In this paper we have shown how the acceptance conditions of membrane systems affect the computational complexity of the system. We have presented an analysis of three different acceptance conditions and proved that they each characterise one of two logspace complexity classes, \mathbf{NL} or \mathbf{L} .

In our previous work [5] we used Definition 8 as our acceptance condition. Systems using this definition are relatively easy to program (construct a membrane system to solve a problem) because one is not concerned with ensuring the system halts or that only **yes** or only **no** is output. However Definition 13 is the more common definition that is used when discussing active membrane systems as it is easier to prove correctness for these systems. The results in Sections 3.1 and 3.2 reveal that when working with $(\mathbf{AC}^0)\text{-PMC}^*_{\mathcal{AM}^0_d}$ systems, both of Definitions 8 and 13 characterise \mathbf{NL} . Our result gives an \mathbf{AC}^0 computable compiler to turn a system obeying one definition into a system obeying the other definition. This makes the choice of either definition a matter of taste and convenience.

We also have given the first complexity class defined by membrane systems that characterises \mathbf{L} .

It is interesting to note that the rules of $(\mathbf{AC}^0)\text{-PMC}^*_{\mathcal{AM}^0_d}$ systems allow for the generation of an exponential amount of objects and membranes. However these systems decide only those problems that a (non-)deterministic Turing machine uses logarithmic space to decide.

Here we looked at a number of acceptance conditions for active membrane systems and then characterised the computational complexity classes of the systems. However, it is also possible to go in the other direction, that is, to choose a complexity class and then try to engineer an acceptance condition in order to characterise the class. This technique may give rise to interesting new characterisations. Furthermore, we would hope that it may even be useful to help solve some open questions on the power of certain classes of membrane systems.

We intend to extend this research to see what effect, if any, acceptance conditions have on the complexity of *uniform* active membrane systems. The techniques may also prove useful for exploring other classes of membrane systems such as tissue P-systems.

Acknowledgements

We would like to thank Mario J. Pérez-Jiménez and Agustín Riscos-Núñez for interesting discussion and clarification on the standard definition of recog-

niser membrane systems. We would also like to thank Petr Sosík for his comments on an earlier draft of this paper.

References

- [1] G. Păun, *Membrane Computing*, Springer-Verlag, Berlin, 2002.
- [2] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez, F. J. Romero-Campero, Computational efficiency of dissolution rules in membrane systems, *International Journal of Computer Mathematics* 83 (7) (2006) 593–611.
- [3] C. Zandron, C. Ferretti, G. Mauri, Solving NP-complete problems using P Systems with active membranes, *Proceedings of the Second International Conference on Unconventional Models of Computation* (2001) 289–301.
- [4] M. Gutiérrez-Naranjo, M. Pérez-Jiménez, A. Riscos-Núñez, F. Romero-Campero, On the power of dissolution in P systems with active membranes, in: *6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*, Vol. 3850, 2006, pp. 224–240.
- [5] N. Murphy, D. Woods, A characterisation of NL using membrane systems without charges and dissolution, *Unconventional Computing, 7th International Conference, UC 2008, LNCS 5204* (2008) 164–176.
- [6] G. Păun, P Systems with active membranes: Attacking NP-Complete problems, *Journal of Automata, Languages and Combinatorics* 6 (1) (2001) 75–90.
- [7] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: A characterization of PSPACE, *Journal of Computer and System Sciences* 73 (1) (2007) 137–152.
- [8] M. J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity classes in models of cellular computing with membranes, *Natural Computing* 2 (3) (2003) 265–285.
- [9] D. A. M. Barrington, N. Immerman, H. Straubing, On uniformity within NC^1 , *Journal of Computer and System Sciences* 41 (3) (1990) 274–306.
- [10] J. L. Balcázar, J. Diaz, J. Gabarró, *Structural complexity I*, 2nd Edition, Springer-Verlag New York, Inc., New York, NY, USA, 1988.
- [11] N. Murphy, D. Woods, Active membrane systems without charges and using only symmetric elementary division characterise P, *8th International Workshop on Membrane Computing, LNCS 4860* (2007) 367–384.
- [12] N. D. Jones, Space-bounded reducibility among combinatorial problems, *Journal of Computer and Systems Sciences* 11 (1) (1975) 68–85.

- [13] N. Immerman, Nondeterministic space is closed under complementation, *SIAM Journal of Computing* 17 (5) (1988) 935–938.
- [14] R. Szelepcsényi, The method of forcing for nondeterministic automata, *Bulletin of the EATCS* 33 (1987) 96–99.
- [15] S. A. Cook, P. McKenzie, Problems complete for deterministic logarithmic space, *Journal of Algorithms* 8 (5) (1987) 385–394.