Uniformity is weaker than semi-uniformity for some membrane systems

Niall Murphy*[1] Damien Woods†[2]

[1]Microsoft Research Cambridge
[2]California Institute of Technology

Abstract

We investigate computing models that are presented as families of finite computing devices with a uniformity condition on the entire family. Examples of such models include Boolean circuits, membrane systems, DNA computers, chemical reaction networks and tile assembly systems, and there are many others. However, in such models there are actually two distinct kinds of uniformity condition. The first is the most common and well-understood, where each input length is mapped to a single computing device (e.g. a Boolean circuit) that computes on the finite set of inputs of that length. The second, called semi-uniformity, is where each input is mapped to a computing device for that input (e.g. a circuit with the input encoded as constants). The former notion is well-known and used in Boolean circuit complexity, while the latter notion is frequently found in literature on nature-inspired computation from the past 20 years or so.

Are these two notions distinct? For many models it has been found that these notions are in fact the same, in the sense that the choice of uniformity or semi-uniformity leads to characterisations of the same complexity classes. In other related work, we showed that these notions are actually distinct for certain classes of Boolean circuits. Here, we give analogous results for membrane systems by showing that certain classes of uniform membrane systems are strictly weaker than the analogous semi-uniform classes. This solves a known open problem in the theory of membrane systems. We then go on to present results towards characterising the power of these semi-uniform and uniform membrane models in terms of NL and languages reducible to the unary languages in NL, respectively.

*a-nimurp@microsoft.com Supported by a PICATA Young Doctors fellowship from CEI Campus Moncloa, UCM-UPM, Madrid, Spain and an Embark Fellowship from the Irish Research Council for Science, Engineering and Technology.

†woods@caltech.edu. Supported by NASA grant NNX13AJ56G, and NSF grants 0832824 & 1317694 (The Molecular Programming Project), CCF-1219274 and CCF-1162589

1 Introduction

Many of the early DNA computing algorithms [22, 28, 29, 39] involved mapping an instance of an NP-hard problem (such as Maximal Clique) to a set of DNA strands and lab protocols, and then using well-known biomolecular techniques to solve the problem. To assert generality for such an algorithm one would define a mapping from arbitrary problem instances to sets of DNA polymers and experimental protocols. In order to claim that this mapping is not doing the essential computation, it would have to be easily computable (for example, logspace computable). Circuit uniformity (introduced by Borodin [11]) provides a well-established framework where we map each input length $n \in \mathbb{N}$ to a circuit $C_n \in \mathfrak{C}$, with a suitably simple mapping. However, some of the DNA computing algorithms cited above do something different, they map an *instance* x of the problem to a computing device C_x that is unique to that input (via a suitably simple encoding function). This latter notion is called *semi-uniformity* [48, 45], and in fact quite a number of nature-inspired computational models use semi-uniformity. This raises the immediate question of whether the notions of uniformity and semi-uniformity are computationally equivalent. We investigate this question in the field of membrane computing or P-systems [48, 43]. This is a branch of natural computing which explores the power of computational models that are inspired by the structure and function of living cells.

It has been shown in a number of models that whether one chooses to use uniformity or semi-uniformity does not affect the power of the model. However, our main result shows that uniformity is computationally strictly weaker than semi-uniformity for a number of classes of membrane systems. Specifically, we prove that choosing one notion over another in this setting gives characterisations of complexity classes that are known to be distinct. The uniform versus semi-uniform question that we address has been stated as Open Problem C in [49].

Why is this result surprising? We know that the class of problems solved by a uniform family of devices is contained in the analogous semi-uniform class, since the former is a restriction of the latter. However, it is also the case that in almost all membrane system models studied to date, the classes of problems solved by semi-uniform and uniform families turned out to be equal, see, e.g., [4, 34, 55]. Specifically, if we want to solve some problem, by specifying a family of membrane systems (or some other model), it is often much easier to first use the more general notion of semi-uniformity, and then subsequently try to find a uniform solution. In almost all cases where a polynomial time semi-uniform family of membrane systems was given for some problem [3, 45, 55], at a later point a uniform version of the same result was published [2, 4, 45]. Here we prove that this improvement is not always possible.

We go on to give a number of other results that tease out the computational power of semi-uniform and uniform families of membrane systems.

Our main result proves something general about uniform and semi-uniform families of finite devices that is independent of particular models and formalisms. Our techniques can be applied to other computational models besides membrane systems and we have demonstrated this by showing similar results for Boolean circuits [38]. Indeed, a number of other models explicitly, or implicitly, use notions of uniformity and semi-uniformity. Models presented as uniform families of devices include membrane systems and Boolean circuits as noted above, as well as DNA computers [1, 9, 53, 50, 8], chemical reaction networks [17, 16, 57, 58], neural networks [42] and other models studied in computational complexity theory. Besides membrane systems, a surprising number of models, including some just mentioned, are presented as semi-uniform families of devices, including DNA computers [28, 29], chemical reaction networks [17, 57], the abstract tile assembly model [51, 54], the nubots model of active molecular self-assembly and robotics [60, 15], and an insertion-based polymer model [18, 30]. Uniform and semi-uniform families of devices are both natural ways to present a model of computation and elucidating the distinction between them seems a worthy goal.

Furthermore, although we do not formally show it, our results hold for a version of the stochastic chemical reaction network model [52] that meets our definitions for membrane systems and in particular where there are families of networks deciding languages and *unimolecular reactions only* (in the model there are discrete natural number molecular counts and all reactions are of the form $A \to \mathcal{M}$, where $\mathcal{M}$ is a mutliset of molecular species). Interestingly, these results also hold if we generalise this model to use maximally parallel synchronous reaction updates. This shows that adding the seemingly strong and unrealistic ability of maximal parallelism in this context conveys no extra power to the model (despite the fact that it does increase the power of more general, bimolecular for example, chemical reaction network models).

Our main result is of importance to work on models of computation and natural computing since it highlights that the (seemingly harmless) choice between uniformity and semi-uniformity in these models may lead to drastic changes in computational power. How drastic? Roughly speaking, we find that the semi-uniform models studied here characterise the class NL, while the analogous uniform models have power comparable to, or more formally reducible to, the unary languages in NL. Our work here and on Boolean circuits suggests that this question should be asked of other computational models.

1.1 Overview of results

Roughly speaking, a membrane system consists of a membrane-bound compartment that contains other (possibly nested) membrane-bound compartments that in turn contain *objects* that interact with each other and with membranes to carry out a computation. A family, or set, of *recogniser* membrane systems decides a language L. Families can be uniform or semi-uniform. For a uniform family there is an associated pair of functions (f, e), where f maps a binary input word x, of length n, to a membrane system Π_n that may be used to process any word of length n, and e encodes x as a multiset of input objects to Π_n (for each of the 2^n words of length $n \in \mathbb{N}$ we have a single membrane system Π_n). For a semi-uniform family, a single function h maps the input word x to a membrane system Π_x (for each word we have a membrane system). In either case, rules are applied to objects in the membrane system until it produces special object(s) indicating that x is accepted or rejected. Of course the encoding functions f, e, h should be suitably simple so that the membrane system, and not the encoding functions, are doing the interesting work. In this paper we use FAC^0 uniformity and semi-uniformity, that is, the functions f, e, h are in FAC^0, the class of functions computed by uniform constant depth polynomial size Boolean circuits; this is a class of fairly simple problems and is mostly known for what it does *not* contain.

In Section 3 we give our main result, that uniform families of active membrane systems without charges and dissolution (denoted $\mathcal{AM}^0_{-d}$) that run in polynomial time are strictly weaker than their semi-uniform counterpart. We prove this by showing that these uniform families solve no more than non-uniform-AC^0, a class that does not even contain Parity (the set of words over $\{0, 1\}$ with an odd number of 1s). The analogous semi-uniform systems can indeed solve Parity and do much else besides. In fact, for two out of three models that we consider, the semi-uniform families exactly characterise NL. This is shown in Section 4 and illustrated as Theorem 17 at the top of Figure 1.

This leaves the question: what is the exact power of uniform families of $\mathcal{AM}^0_{-d}$ systems? In previous papers, where more powerful membrane systems and complexity classes are studied, e.g. [2, 3, 4], model definitional choices were not so important. In our setting, definitional details such as the choice of uniformity condition and the particular kinds of acceptance modes allowed for such *recogniser* membrane systems lead to seemingly different results and some open questions as we now describe.

We give results for three variants on the definition of recogniser membrane system. The most powerful are *acknowledger* membrane systems, where an accepting computation should produce one or more *yes* objects, and a rejecting computation should produce zero *yes* objects. In Section 5 we give an exact characterisation of uniform families of acknowledger $\mathcal{AM}^0_{-d}$ membrane systems. It turns out that they decide exactly those languages that are FAC^0 disjunctive truth-table reducible to the unary languages in NL (called tallyNL). See Theorem 20 in Figure 1.

In Section 6 we consider *recogniser*$_{\geqslant 1}$ membrane systems: a restriction of acknowledger systems where an accepting computation produces one or more *yes* objects and zero *no* objects, and a rejecting computation produces one or more *no* objects and zero *yes* objects. We give upper and lower-bounds, in terms of classes reducible to tallyNL, for uniform families of *recogniser*$_{\geqslant 1}$ $\mathcal{AM}^0_{-d}$ systems. In Figure 1, two upper bounds are illustrated as Theorems 21 and 23, and a lower bound as Theorem 24.

The more standard, uniform *recogniser* systems, are a restriction of recogniser$_{\geqslant 1}$ membrane systems and are defined so that an accepting computation should produce a single *yes* object and zero *no* objects, and a rejecting computation should produce a single *no* object and zero *yes* objects. As noted above, our results (Figure 1, Theorem 16) show that these uniform recogniser systems are strictly weaker than semi-uniform recogniser systems in our setting. We do not give a tight characterisation for the power of uniform recogniser systems, but discuss this as an open problem in Section 7.

We note that there is a previous P characterisation for both uniform and semi-uniform families of active membrane systems without charges and dissolution [21]: the same systems as we use here, but under much more general uniformity conditions, namely *polynomial time*, or P, uniformity. In that work the authors are motivated by the relationship with classes above P and so it is sufficient in their work to use P uniformity. When using significantly tighter uniformity conditions (e.g. FAC^0), such polynomial-time encoding functions for uniform and semi-uniform families can be

$$\begin{tikzcd}
 & \overset{\text{acknowledger}}{\mathsf{FAC}^0\text{-semi-uniform-}\mathsf{PMC}^*\mathcal{AM}^0_{-d}} \arrow[r, equal, "(17)"] & \mathsf{NL} \arrow[r, equal, "(17)"] & \overset{\text{recogniser}_{\geqslant 1}}{\mathsf{FAC}^0\text{-semi-uniform-}\mathsf{PMC}^*\mathcal{AM}^0_{-d}} & \\
 & & \mathsf{FAC}^0_{\mathrm{T}}(\mathsf{tallyNL}) \arrow[u, "\subsetneq"'] & & \\
 & & \mathsf{FAC}^0_{\mathrm{tt}}(\mathsf{tallyNL}) \arrow[u] & & \\
\overset{\text{acknowledger}}{(\mathsf{FAC}^0,\mathsf{FAC}^0)\text{-uniform-}\mathsf{PMC}\mathcal{AM}^0_{-d}} \arrow[uuur, "\subsetneq", "(16)"'] \arrow[r, equal, "(20)"] & \mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL}) \arrow[ur] & & \mathsf{FAC}^0_{\mathrm{ctt}}(\mathsf{tallyNL}) \arrow[ul] & \\
 & & \star \arrow[ul, "(21)"] \arrow[ur, "(23)"'] \arrow[rr, dashed, no head] & & \overset{\text{recogniser}_{\geqslant 1}}{(\mathsf{FAC}^0,\mathsf{FAC}^0)\text{-uniform-}\mathsf{PMC}\mathcal{AM}^0_{-d}} \\
 & & \mathsf{FAC}^0_{\mathrm{m}}(\mathsf{tallyNL}) \arrow[u, "(24)"'] \arrow[uul, bend left] \arrow[uur, bend right] & & \\
 & & \mathsf{tallyNL} \arrow[u, "\subsetneq"] & &
\end{tikzcd}$$

Figure 1: Summary of results. Numerical labels refer to theorems which are proved in this paper, and symbols are used to show inclusion type, with an unlabelled arrow denoting $\subseteq$. The figure shows the relationship between NL, $\mathsf{tallyNL}$ (the set of unary languages decided in non-deterministic logspace), and a number of classes $\mathsf{FAC}^0_{\mathrm{r}}(\mathsf{tallyNL})$ of languages that are reducible to $\mathsf{tallyNL}$ by various types of FAC^0 computable reductions r. The star ($\star$) indicates the class labeled by the dashed line. See [38] for proofs of inclusions that do not have numerical labels and for more on classes reducible to $\mathsf{tallyNL}$.

seen to be stronger than the membrane systems themselves [37] (assuming $\mathsf{NL} \subsetneq \mathsf{P}$). In this paper we use FAC^0 uniformity which is weak enough to expose the underlying power of certain, suitably weak, classes of active membrane systems without charges or dissolution. A number of other varieties of membrane systems (e.g. [20, 40]) also claim P characterisations that depend on P uniformity. We leave it as a possible direction for future work to investigate these, and other, membrane systems under suitably tight notions of uniformity or semi-uniformity.

2 Definitions

For a function $f\colon \{0,1\}^* \mapsto \{0,1\}^*$ and integers $m, n \geq 1$ let $f_n\colon \{0,1\}^n \mapsto \{0,1\}^m$ be the restriction of f to domain and range consisting of strings of length n and m respectively. We consider only functions f where for each n there is an m such that all length-n strings in f's domain are mapped to length-m strings, thus $f = \bigcup_{n=0}^{\infty} f_n$. Each language $L \subseteq \{0,1\}^*$ has an associated total *characteristic function* $\chi_L\colon \{0,1\}^* \mapsto \{0,1\}$ defined by $\chi_L(w) = 1$ if $w \in L$ and 0 if $w \notin L$. We say a language L is decided by a Turing machine M if M computes the characteristic function χ_L. For a string w, we let $|w|$ denote its length.

Let NL be the class of languages accepted by non-deterministic logarithmic-space

Turing machines. Such machines have a read-only input tape, a write-only output tape and a read-write work tape whose length is a logarithmic function of input length. The class of functions computed by a deterministic logarithmic-space Turing machines (with an additional write-only output tape) is denoted FL.

Let tally be the set of all languages over the one-letter alphabet $\{1\}$. We define $\mathsf{tallyNL} = \mathsf{tally} \cap \mathsf{NL}$, i.e. the class of all tally languages and length encoded languages in NL. For more details on complexity classes and Turing machines see [41].

A circuit C_n computes a function computes a function $f_n\colon \{0,1\}^n \mapsto \{0,1\}^m$ on a fixed number n of Boolean variables. We consider functions of an arbitrary number of variables by defining (possibly infinite) families of circuits. We say a family of circuits $\mathfrak{C} = \{C_n \mid n \in \mathbb{N}\}$ computes a function $f\colon \{0,1\}^* \mapsto \{0,1\}^*$ if for all $n \in \mathbb{N}$ and for all $w \in \{0,1\}^n$ circuit C_n outputs the string $f_n(w)$. We say a family of circuits $\mathfrak{C}$ decides a language $L \subseteq \{0,1\}^*$ if for each $w \in \{0,1\}^n$ circuit $C_n \in \mathfrak{C}$ on input w computes χ_L.

In a *non-uniform* family of circuits there is no required similarity between family members. In order to specify such a requirement we use a *uniformity function* that algorithmically specifies the similarity between members of a circuit family. Roughly speaking, a *uniform circuit family* $\mathfrak{C}$ is an infinite sequence of circuits with an associated function $f\colon \{1\}^* \to \mathfrak{C}$ that generates members of the family and is computable within some resource bound. For more details on Boolean circuits see [59].

When dealing with uniformity for small complexity classes one of the preferred uniformity conditions is $\mathsf{DLOGTIME}$-uniformity [32]. Roughly speaking, a circuit is $\mathsf{DLOGTIME}$-uniform if there is a procedure that can decide if a word is in the "connection language" of the circuit family in time linear in the word length. Each word of the connection language encodes either an input gate of the circuit, an output gate of the circuit, or a wire connecting the output of one identified gate to the input of a second identified gate. Each word also encodes, in binary, the number n for this circuit. For more details on $\mathsf{DLOGTIME}$ uniformity see [5, 32].

The *depth* of a circuit is the length of the longest path from an input gate to an output gate. The *size* of a circuit is the number of wires it contains [5].

Non-uniform-AC^0 is the set of languages decidable by families of constant-depth polynomial-size (in input length n) circuits with unbounded fan-in AND and OR gates, and NOT gates with fan-in‘1. AC^0 is the set of languages decidable by constant-depth polynomial-size (in input length n) $\mathsf{DLOGTIME}$-uniform circuits with unbounded fan-in AND and OR gates, and NOT gates with fan-in 1. FAC^0 is the class of functions computable by polynomial-size constant-depth $\mathsf{DLOGTIME}$-uniform circuits with unbounded fan-in AND and OR gates, and NOT gates with fan-in 1.

2.1 Reductions

For concreteness, we explicitly define some standard types of reductions. Let $A, B \subseteq \{0,1\}^*$. Let C be a set of functions (for example FL or FAC^0), a function f is C-computable if $f \in \mathsf{C}$.

Definition 1 (Many-one reducible)**.** *Set A is many-one reducible to set B, written $A \leq_{\mathrm{m}}^{\mathsf{C}} B$, if there is a function f that is C-computable with the property that for all*

w, $w \in A$, if and only if $f(w) \in B$.

The following definition of truth table reduction comes from [10, 12], see also [27, 47]. The Boolean function σ is historically called a truth table [47].

Definition 2 (Truth-table reduction)**.** *Set A is C truth table reducible to set B, written $A \leq^{\mathsf{C}}_{\mathrm{tt}} B$, if there exists C-computable functions $\tau : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^* \times \ldots$ and $\sigma : \{0,1\}^* \to \{0,1\}$ such that $w \in A$ if and only if $\tau(w) = (a_1, \ldots, a_{\ell_w})$ such that $\sigma(\chi_B(a_1), \ldots, \chi_B(a_{\ell_w})) = 1$, where χ_B is the characteristic function of B.*

A *disjunctive* truth table reduction (dtt) is one where at least one string generated by $\tau(w)$ is in B, in other words $\sigma(\chi_B(a_1), \ldots, \chi_B(a_{\ell_w})) = \bigvee_{1 \leq i \leq \ell_w} \chi_B(a_i)$. A *conjunctive* truth table reduction (ctt) is one where all the strings generated by $\tau(w)$ are in B, in other words $\sigma(\chi_B(a_1), \ldots, \chi_B(a_{\ell_w})) = \bigwedge_{1 \leq i \leq \ell_w} \chi_B(a_i)$.

Definition 3 (Turing reducible)**.** *Set A is C Turing reducible to B, written $A \leq^{\mathsf{C}}_{\mathrm{T}} B$, if there is a Turing machine M, that is resource-bounded in the same way machines computing functions in C are, such that $w \in A$ iff M accepts w with B as its oracle.*

The following implications follow directly from these definitions, for more details see [27].

$$\begin{tikzcd}[row sep=tiny] & A \leq^{\mathsf{C}}_{\mathrm{dtt}} B \arrow[dr, Rightarrow] & & \\ A \leq^{\mathsf{C}}_{\mathrm{m}} B \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & & A \leq^{\mathsf{C}}_{\mathrm{tt}} B \arrow[r, Rightarrow] & A \leq^{\mathsf{C}}_{\mathrm{T}} B \\ & A \leq^{\mathsf{C}}_{\mathrm{ctt}} B \arrow[ur, Rightarrow] & & \end{tikzcd}$$

Let $\mathsf{FAC}^0_{\mathrm{r}}(\mathsf{C})$ be the set of all languages that are FAC^0 reducible to languages in C via a reduction of some type r $\in$ {m, dtt, ctt, tt, T}.

2.2 Configuration graphs

Definition 4 (Configuration Graph)**.** *Let $w \in \{0,1\}^*$ be the input to a halting $s(|w|)$-space bounded Turing machine M. The* configuration graph *$G_{M,w}$ of M on input w is an acyclic directed graph where for each potential configuration of M there is a vertex that encodes it and where a potential configuration consists of an input read bit, work tape contents, input tape head position and work tape head position. The graph $G_{M,w}$ has a directed edge from a vertex c to a vertex c' if the configuration encoded by c' can be reached from the configuration encoded by c in one step via M's transition function.*

A configuration graph $G_{M,w}$ has the property that there is a directed path from the vertex c_s representing the initial configuration, to the accept vertex c_a if an only if M accepts input w. Also, we consider only space bounded Turing machines that do not repeat a configuration (i.e. loop), hence we define configuration graphs to be acyclic which will be a useful property later on. We are interested in $\mathcal{O}(\log |w|)$ space bounded Turing machines, whose configuration graphs are of size (number of vertices) $\mathcal{O}(|w|^2|Q|)$. Lemma 5 follows from Theorem 3.16 in [25].

Lemma 5. *Given the binary encoding of a Turing machine M, which has state set Q and an FAC^0 computable space bound $\mathcal{O}(\log |w|)$, and given an input w, the configuration graph $G_{M,w}$, of size $\mathcal{O}(|w|^2|Q|)$, is computable in $\mathsf{DLOGTIME}$-uniform-FAC^0.*

2.3 Membrane systems

In this section we define the specific variant of membrane systems we use in this paper. We also define recognizer membrane systems, uniform families and some complexity classes. These definitions are based on those from the literature [31, 44].

In this paper the term *membrane systems* and the notation $\mathcal{AM}^0_{-d}$ refer to active membrane systems without charges and without dissolution rules [21, 44].

Let $\mathsf{MS}(O)$ represent the set of all multisets over the elements of the finite set O.

Definition 6. *A* membrane system *of type* $\mathcal{AM}^0_{-d}$ *is a tuple* $\Pi = (O, \mu, M, H, \Lambda, R)$ *where:*

- O *is the alphabet of objects (or the set of object types);*
- $\mu = (V_\mu, E_\mu, env)$ *is a rooted tree representing the membrane structure.* $V_\mu \subsetneq \mathbb{N}$ *is the finite set of membranes.* $E_\mu \subsetneq V_\mu \times V_\mu$ *such that* $(p, c) \in E_\mu$ *if the (parent) membrane* p *contains the (child) membrane* c*. The root,* $env \in V_\mu$*, of the tree is the only membrane with no parent and is called the "environment". Leaves of the tree represent "elementary membranes": i.e. membranes which contain no other membranes.*
- $M \colon V_\mu \to \mathsf{MS}(O)$ *map each membrane to an object multiset, defining the membrane's object contents;*
- $\Lambda \colon V_\mu \to H$ *is an injective mapping of membranes to* H*, the finite set of membrane labels. In this work the environment membrane always has the label "env";*
- R *is a finite set of developmental rules of the following types (where* $o, u, c \in O$ *and* $w \in \mathsf{MS}(O)$*,* $h \in H$*):*

  (a) $[\, o \to w \,]_h$ *(object rewriting), an object* o *in a membrane with label* h *is replaced by a multiset of objects* w*.*

  (b) $o\,[\,]_h \to [\, u \,]_h$ *(communication in), an object* o *in a membrane with a child membrane with label* h *is moved into the child membrane and modified to become* u*.*

  (c) $[\, o \,]_h \to [\,]_h\, u$ *(communication out), an object* o *in a membrane with label* h *is moved into the parent membrane and modified to become* u*.*

  (e) $[\, o \,]_h \to [\, u \,]_h\, [\, v \,]_h$ *(elementary membrane division), an elementary membrane with label* h *containing object* o *is duplicated, in one copy* o *is replaced by* u *while in the other copy it is replaced by* v*.*

The environment membrane cannot divide nor communicate out objects.[1]

The missing (d) rule is the dissolution rule which we do not consider in this paper. Active membrane systems may also have *non-elementary membrane division*

[1]Definitions of active membranes often include a second container membrane that cannot dissolve called the "skin" [44], we omit this from our definitions. The proofs in this paper can be easily modified to account for a skin.

rules [44]. That is, membranes with child membranes may also divide. For the kinds of membrane systems we consider in this paper the inclusion or omission of non-elementary division rules does not affect the results [21, 37].

A *configuration* $\mathcal{C}$ of a membrane system is a tuple $(\mu = (V_\mu, E_\mu, env), M, \Lambda)$ whose elements are defined in Definition 6 (with the exception that Λ may be non-injective).

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 is encoded in the format $[a, a, a, b, b]$, rather than in the shorter format a^3b^2. Likewise, the membrane structure should be encoded such that each membrane child-parent relation is written explicitly.

A configuration $\mathcal{C}_i$ *transitions* to configuration $\mathcal{C}_{i+1}$ by the application of a multiset of rules $\mathcal{R}$ from the set R. The rules are applied in a *maximally parallel manner*. That is, at each timestep, a multiset of applicable rules $\mathcal{R}$ is non-deterministically chosen such that (i) all rules in $\mathcal{R}$ are applicable, and (ii) there does not exist a multiset of applicable rules $\mathcal{R}'$ such that $\mathcal{R} \subsetneq \mathcal{R}'$. Rules are *applicable* in a timestep according to the following principles: Rules are applied to the most deeply nested membranes first. In each timestep, an object can be involved in at most one rule of any type. A membrane can be the subject of at most one rule of type (b), (c) or (e). If a membrane is divided (a rule of type (e)) and there are objects in this membrane which evolve via rules of type (a), then we assume that first the type (a) rules are applied, and then the division rule. All other rules are applied non-deterministically.

A *computation* of a membrane system is a sequence of configurations where each configuration transitions to the next. As noted above, at a given timestep the multiset of applicable rules is non-deterministically chosen: therefore on a given input there are multiple possible computations. In other words, membrane systems are non-deterministic. A computation that reaches a configuration where no more rules are applicable is called a *halting computation.*

2.3.1 Recogniser, recogniser$_{\geqslant 1}$, and acknowledger membrane systems

For the following three definitions it is the case that the set of objects O contains the special objects yes and no and that there are no rules applicable to yes or no (hence if yes or no are created, they can never be destroyed). The standard [44] definition of a recogniser membrane system is as follows.

Definition 7. *A* recogniser membrane system *is a membrane system such that all computations halt, and at the halting step (and not before) exactly one of the objects* $\mathit{yes} \in O$ *or* $\mathit{no} \in O$ *appears in the multiset of the environment membrane.*

A computation that halts with yes in the environment is referred to as an *accepting computation* while one with no in the environment is referred to as a *rejecting computation.* In this paper, and in previous work [36, 37], we also use the following more general systems:

Definition 8. *A* recogniser$_{\geqslant 1}$ membrane system *is a membrane system such that all computations halt, and either (a) one or more copies of the object* $\mathit{yes} \in O$ *or (b) one or more copies of the object* $\mathit{no} \in O$ *appear in the multiset of the environment membrane, but not both.*

As with recogniser membrane systems, a computation of a recogniser$_{\geqslant 1}$ membrane system that halts with $\texttt{yes}$ in the environment is referred to as an *accepting computation* while one with $\texttt{no}$ in the environment is referred to as a *rejecting computation.* In this paper we also use the following systems that are more general than the two above:

Definition 9. Acknowledger membrane systems *are systems such that all computations halt (and where one or more copies of the distinguished object* $\texttt{yes}$ *may or may not appear in the env membrane).*

We say that a computation of an acknowledger membrane system is an *accepting computation* if at least one $\texttt{yes}$ object is present in the *env* membrane at the final step. A computation of an acknowledger membrane system is in a *rejecting computation* if there are zero $\texttt{yes}$ objects in the *env* membrane at the final step.

2.3.2 Families of membrane systems

There are two main notions of uniformity considered in the membrane computing literature defined as follows.

Definition 10 (Semi-uniform families)**.** *A family of membrane systems systems* $\mathbf{\Pi} = \{\Pi_w \mid w \in \Sigma^*\}$ *is said to be* semi-uniform *if there is a function* $h\colon \Sigma^* \mapsto \mathbf{\Pi}$ *that maps from each input word* w *to a description (in a permissible encoding) of a membrane system* Π_w*.*

Definition 11 (Uniform families)**.** *A family of membrane systems* $\mathbf{\Pi} = \{\Pi_n \mid n \in \mathbb{N}\}$ *is said to be uniform if there are two associated functions:*

1. $f\colon 1^* \mapsto \mathbf{\Pi}$ *that maps* 1^n *(the unary representation of* n*) to the description (in a permissible encoding) of a membrane system* Π_n *with a designated input membrane;*

2. $e\colon \Sigma^* \mapsto \mathsf{MS}(O)$ *that maps a word* $w \in \Sigma^*$ *to the* input *multiset* $e(w)$ *(in a permissible encoding) where* O *is the set of objects of* $f(1^n)$*,* $n = |w|$*.*

We let $\Pi_n(e(w))$ denote the membrane system $f(1^n) = \Pi_n$ with the multiset $e(w)$ in its designated input membrane. Note that both Π_n and $\Pi_n(e(w))$ are membrane systems.

In this paper, we deal only with confluent membrane systems: in a *confluent membrane system* Π all computations of Π agree on the answer, that is, either all of Π's computations are accepting (in which case Π accepts) or else all of Π's computations are rejecting (in which case Π rejects).

A semi-uniform family, $\mathbf{\Pi}$, recognises a language $L \subseteq \Sigma^*$ confluently if for all $w \in \Sigma^*$ there exists $\Pi_w \in \mathbf{\Pi}$ such that $w \in L$ implies that Π_w accepts confluently and $w \notin L$ implies Π_w rejects confluently. A uniform family, $\mathbf{\Pi}$, with encoder e, recognises a language $L \subseteq \Sigma^*$ confluently if for all $w \in \Sigma^*$ there exists $\Pi_{|w|}(e(w))$ where $\Pi_{|w|} \in \mathbf{\Pi}$ such that $w \in L$ implies that $\Pi_{|w|}(e(w))$ accepts confluently and $w \notin L$ implies $\Pi_{|w|}(e(w))$ rejects confluently. Such a (semi-)uniform families are called a *confluent* families of recogniser, recogniser$_{\geqslant 1}$, or acknowledger membrane systems.

That is, each membrane system Π in a confluent family starts from a fixed initial configuration and then Π non-deterministically chooses one from a number of valid computations. All of these valid computations give the same result: either all accepting (if $w \in L$) or else all rejecting (if $w \notin L$).

If the functions $f(1^{|w|})$ and $e(w)$ (or respectively the single function $h(w)$) for a (semi-)uniform family are computable in time polynomial in $|w|$ on a Turing machine we say the family uses *polynomial time (semi-)uniformity.* If the uniformity functions are computable by $\mathsf{DLOGTIME}$ uniform constant depth circuits, that is, $f, e, h \in \mathsf{FAC}^0$, then the family is said to use constant depth uniformity.

In this paper we consider two classes of problems, those that can be solved by FAC^0-uniform families of confluent $\mathcal{AM}^0_{-d}$ (active membranes without charges or dissolution rules) that run in time polynomial in $|w|$, denoted $(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d}$, and FAC^0-semi-uniform families of confluent $\mathcal{AM}^0_{-d}$ systems that run in time polynomial in $|w|$, denoted FAC^0-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$.

2.4 Context-freeness in membrane systems

Lemma 12. *Let o be an object in a membrane with label h in a configuration $\mathcal{C}_i$ of a membrane system Π. Remove all other objects from $\mathcal{C}_i$ to get configuration $\mathcal{C}_i^{\emptyset}$. If there is a rule r in Π such that by applying that rule to o, h in $\mathcal{C}_i^{\emptyset}$ gives a configuration $\mathcal{C}_{i+1}^{\emptyset}$ with object o' in h', then it is the case that from configuration $\mathcal{C}_i$ there exists a configuration $\mathcal{C}_{i+1}$ reachable in a single step that contains o' in h'.*

Proof. The rule r is of the type (a), (b), (c) or (e) as described in Definition 6. It is sufficient to show that there is always at least one maximal set of rule applications for configuration $\mathcal{C}_i$ that creates o' in h' in $\mathcal{C}_{i+1}$.

Recall that an object in a configuration can be involved in at most one rule of any type. If the rule r is of type (a), it has the form $[\,o \to o'w\,]_h$ where w is a (possibly empty) string over O and it is necessarily the case that $h = h'$ (rules of type (a) are applied within a single membrane).

Let the notation $\mathcal{C}_i - \{o\}$ denote the configuration $\mathcal{C}_i$ without the instance of the object o under consideration and consider any maximal multiset R of rules that can be applied to the configuration $\mathcal{C}_i - \{o\}$. Also, consider the multiset of rule applications R unioned with the application of the rule $[\,o \to o'w\,]_h$ to our object instance o in the relevant membrane with label h in $\mathcal{C}_i$. We claim that this new multiset is a maximal multiset of rules that can be applied to $\mathcal{C}_i$. To see this notice that object instance o has a rule being applied to it, and each object can have at most one rule applied to it, and no other objects with applicable rules are without rules because R was maximal. Hence there is a maximal multiset of rule applications for $\mathcal{C}_i$ that applies r and hence when it is applied $\mathcal{C}_{i+1}$ contains o' in a membrane with label $h = h'$.

Rules of type (b), (c) and (e) involve both an object and a membrane. Consider $\mathcal{C}_i - \{o\}$ defined as above, and consider any maximal multiset R of rule applications to $\mathcal{C}_i - \{o\}$. Furthermore, if in R there is a rule that involves the membrane with label h where object instance o was, then remove that rule application from R to get $\hat{R}$. We claim that the multiset of rule applications $\hat{R}$, unioned together with the rule application "rule r applied to our object instance o contained in the membrane with label h" is a maximal multiset of rule applications for $\mathcal{C}_i$. To see this note that (i) r is now being applied to the relevant instances of o, h so no other rule can be applied to that object nor to the membrane with that label, and (ii) there are no other rules that can be applied because R was maximal. After the application of this maximal multiset of rules the new configuration $\mathcal{C}_{i+1}$ contains o' in a membrane with label h'. □

The following lemma generalises Lemma 12 from one to multiple computation steps, and applies it to the setting of systems that recognise languages. Intuitively, it states that if a sequence of rules r can be applied starting from some configuration it is not possible to prevent this from happening by adding new objects to that configuration.

Lemma 13. *Let Π be a recogniser, recogniser$_{\geqslant 1}$ or acknowledger membrane system. Let o be an object in a membrane with label h in a configuration $\mathcal{C}_i$ of Π. Remove all other objects from $\mathcal{C}_i$ to get configuration $\mathcal{C}_i^{\emptyset}$. If starting from configuration $\mathcal{C}_i^{\emptyset}$ there is a computation that halts after t steps on configuration $\mathcal{C}_{i+t}$ that contains object* `yes` *in the environment then it is the case that starting from configuration $\mathcal{C}_i$ there exists a halting computation with* `yes` *in the environment.*

Proof. By hypothesis we know that there is a sequence of t rules $r_1, r_2, \ldots, r_t$ that can be applied to $\mathcal{C}_i^{\emptyset}$ to get `yes` in the environment. We apply Lemma 12 t times, first to configuration $\mathcal{C}_i$ with $r = r_1$, then to $\mathcal{C}_{i+1}$ with $r = r_2$, and so on until we get configuration $\mathcal{C}_{i+t}$ which contains `yes` in the environment.

If Π is a recogniser system then we are done: recogniser systems produce `yes` *in the halting step.* If Π is a recogniser$_{\geqslant 1}$ or acknowledger membrane system we add the fact (from Section 2.3.1) that no rules can be applied to the object `yes`, and since there is a computation where `yes` is in the environment at configuration $\mathcal{C}_{i+t}$, then it remains there until the computation eventually halts. □

Lemma 13 shows that the kind of membrane systems studied in this paper intuitively exhibit some notion of context-freeness. Essentially, there is a sense in which an object o_s can be said to trigger a sequence of rules that eventually result in the production of object o_t on some computation, and specifically, the production of o_t can not be prevented by starting over with more objects (more context) in the system. Hence, the ideas used in the proof of Lemma 13 justify the use of the following definition in our proofs.

Definition 14 (Eventually evolves)**.** *Let $\mathcal{C}_s$ be a configuration of a membrane system Π, containing an object of type o_s in a membrane labelled h_s (along with any number of other objects and membranes). Let $\mathcal{C}_s^{\emptyset}$ denote $\mathcal{C}_s$ with all objects removed except one instance of o_s in the relevant membrane with label h. We say that o_s in h_s in configuration $\mathcal{C}_s$* eventually evolves on some computation path*, or for short* eventually evolves*, object type o_t in a membrane labelled h_t if there is a computation (sequence of configurations) starting from $\mathcal{C}_s^{\emptyset}$ where the final configuration in the computation has object type o_t in a membrane labelled h_t.*

Note that if o_s in h_s in $\mathcal{C}_s$ eventually evolves $\texttt{yes}$ in *env* this means that by Lemma 13 there is at least one computation (sequence of configurations) that leads to a configuration with $\texttt{yes}$ in *env* from $\mathcal{C}_s$. However, since membrane systems are nondeterministic, this does not necessarily happen for all computations.

3 Uniformity is strictly weaker than semi-uniformity

Theorem 16 proves that uniform families of membrane systems are strictly weaker than semi-uniform families of the same type. The result holds for all three definitions of acknowledger, recogniser$_{\geqslant 1}$ and recogniser membrane systems.

Lemma 15. $(\mathsf{FAC}^0, \mathsf{FAC}^0)$*-uniform-*$\mathsf{PMC}\mathcal{AM}^0_{-d} \subseteq$ *non-uniform-*AC^0*, for acknowledger, recogniser*$_{\geqslant 1}$ *and recogniser membrane systems.*

Proof. Let $L \in (\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d}$, and let $\mathbf{\Pi}$ be the FAC^0-uniform family of that type that decides L. That is, given $w \in \{0,1\}^*$ there is a membrane system $\Pi_{|w|} \in \mathbf{\Pi}$ that accepts $e(w)$ iff $w \in L$.

We now describe a non-uniform family of constant-depth circuits $\mathfrak{C} = \{C_n \mid n \in \mathbb{N}$ and C_n accepts $L \cap \{0,1\}^n\}$ that recognizes L. For any input $w \in \{0,1\}^*$, we claim that circuit $C_{|w|} \in \mathfrak{C}$ decides whether or not $w \in L$. The first constant number of layers of the circuit $C_{|w|}$ compute the input encoding function $e(w) \in \mathsf{FAC}^0$. This generates a polynomial (in $|w|$) number of binary words that encode elements from the polynomially sized object set O as well as their multiplicities (in unary).

The circuit $C_{|w|}$ then converts the list of encoded $e(w)$ objects into a single binary string χ of length $|\chi| = |O|$ such that for all $i \in \{1, 2, \ldots, |O|\}$, the ith bit $\chi_i = 1$ iff $o_i \in O$ is in $e(w)$. That is, χ is a characteristic sequence for $e(w)$, ignoring multiplicities.

For each i, the bit χ_i is wired into a unique AND gate a_i, giving a total of $|O|$ AND gates at this level. The second input to the AND gate a_i is from a constant gate c_i, where $c_i = 1$ if $o_i \in O$ in the input membrane eventually evolves (Definition 14) to the $\texttt{yes}$ object in the *env* membrane and $c_i = 0$ otherwise.

The next layer contains a single OR gate g such that for each i, AND gate a_i is wired to g. This OR gate is the output gate of the circuit. Also wired into the OR are $|O| \times |H|$ constant gates such that gate $c_{o,h} = 1$ if both (i) $o \in O$ is in membrane labelled $h \in H$ in the initial configuration of $\Pi_{|x|}$ and (ii) o in h eventually evolves to $\texttt{yes}$ in the *env* membrane, otherwise $c_{o,h} = 0$.

We now argue that the above construction of $C_{|w|}$ accepts $w \in L$. Recall that $\Pi_{|w|}(e(w))$ is a confluent membrane system and so if the computation is an accepting one, then all possible computation paths are accepting. For a computation to be accepting, a $\texttt{yes}$ object must appear in the *env* membrane. Therefore at least one object in the initial configuration of $\Pi_{|w|}(e(w))$ must eventually evolve to be a $\texttt{yes}$ in the *env* membrane. Also $\Pi_{|w|}(e(w))$ is confluent, therefore if at least one object in the initial configuration of $\Pi_{|w|}(e(w))$ eventually evolves $\texttt{yes}$ in the *env* membrane, the system accepts. Since the property of whether an object in some membrane eventually evolves to object $\texttt{yes}$ in the *env* membrane depends only on R and μ in $\Pi_{|w|}$, and hence in turn depends only on $|w|$ (by Lemma 13), it can be encoded (non-uniformly) in the constants c_i in circuit $C_{|w|}$.

Suppose $\Pi_{|w|}$ accepts regardless of the input $e(x)$. In this case one of the objects, say o, in the initial configuration of $\Pi_{|w|}$ will eventually evolve to *yes* in the *env* membrane. This means the relevant gate $c_{o,h}$ will be a 1-constant gate and so the output OR will evaluate to 1 and so $C_{|w|}$ accepts regardless of input.

Suppose $w \in L$, therefore at least one of the objects in in $e(w)$, when placed in the input membrane of $\Pi_{|w|}$, yields a computation that ends with a configuration with object *yes* in membrane *env*. In turn this implies that at least one of the AND gates a_i has inputs $c_i = 1$ and $\chi_i = 1$ and so evaluates to 1. Finally this causes the OR to evaluate to 1 and so $C_{|w|}$ accepts input w.

Suppose $w \notin L$, in this case none of the objects in $e(w)$ will eventually evolve to *yes* in the *env* membrane. Thus any of the a_i AND gates that have a constant $c_i = 1$ as input will have $\chi_i = 0$ and so will evaluate to 0. With all 0 inputs, the output OR evaluates to 0 and the circuit rejects.

This circuit is of polynomial size and its depth is the sum of the depths of the FAC^0 encoding function (which has depth $O(1)$, by definition), the depth of the circuit that converts $e(w)$ into χ (which is $O(1)$ using masking and comparison), and 2 for the final layer of AND gates and the single OR gate. Hence $\mathfrak{C}$ is a non-uniform-AC^0 circuit family that recognizes L. □

Theorem 16. *$(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d} \subsetneq \mathsf{FAC}^0$-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$, for acknowledger, recogniser$_{\geqslant 1}$ and recogniser membrane systems.*

Proof. ($\subseteq$) By definition, uniform families are a restriction of semi-uniform families and so $(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d} \subseteq \mathsf{FAC}^0$-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$.
($\neq$) $\mathsf{Parity} \subseteq \{0,1\}^*$ is the set of binary strings that contain an odd number of 1s. We claim that $\mathsf{Parity} \in \mathsf{FAC}^0$-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$ for recogniser systems (and hence also for acknowledger and recogniser$_{\geqslant 1}$ membrane systems). Let $w \in \{0,1\}$, $n = |w|$, and let $w = w_1, \ldots, w_n$. We will define the function $h\colon \{0,1\} \mapsto \mathbf{\Pi}$, where each $h(w) = \Pi_w$ computes $\chi_{\mathsf{Parity}}(w)$ as follows. Each Π_w has a single membrane, *env*, the set O contains $2n+2$ objects: $O = \{o_i | 1 \leq i \leq n\} \cup \{e_i | 1 \leq i \leq n\} \cup \{\textit{yes}, \textit{no}\}$. The initial configuration is the membrane *env* containing a single object o_1 in *env* if $w_1 = 1$ or object e_1 in *env* if $w_1 = 0$. The rules of Π_w are as follows: if $w_i = 1$ then $[\, o_i \to e_{i+1} \,]_{env}$, $[\, e_i \to o_{i+1} \,]_{env}$ and if $w_i = 0$ then $[\, e_i \to e_{i+1} \,]_{env}$, $[\, o_i \to o_{i+1} \,]_{env}$. There are also the rules $[\, e_n \to \textit{no} \,]_{env}$ and $[\, o_n \to \textit{yes} \,]_{env}$.

By starting with object o_1 if $w_1 = 1$, and then changing between e_i and o_i if $w_i = 1$, and not changing if $w_i = 0$ at each timestep then we ensure that the object o_i represents "the parity of the first i bits of w is odd", and e_i represents that they are even. Thus, o_n evolves to a single *yes* object if there is an odd number of 1s in w and e_n evolves to a single *no* if there is an even number of 1s in w.

To end we note that it is known [19] that $\mathsf{Parity} \notin$ non-uniform-AC^0. Lemma 15 shows that $(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d} \subseteq$ non-uniform-AC^0, for acknowledger, recogniser$_{\geqslant 1}$, and recogniser membrane systems. □

4 The computational power of semi-uniform families

In prior work [37], we have shown that semi-uniform families of recogniser$_{\geqslant 1}$ membrane systems characterise NL. We give an alternative proof here to demonstrate techniques that we will use in later sections for uniform families.

Theorem 17 ([37])**.** *FAC^0-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d} = \mathsf{NL}$, for both acknowledger and recogniser$_{\geqslant 1}$ membrane systems.*

Proof. Lemmas 18 and 19 give the proof for acknowledger and recogniser$_{\geqslant 1}$ membrane systems. □

Lemma 18 ([37])**.** *FAC^0-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d} \subseteq \mathsf{NL}$, for acknowledger, recogniser$_{\geqslant 1}$ and recogniser membrane systems.*

Proof. Let $\mathbf{\Pi}$ be a semi-uniform family of acknowledger, recogniser$_{\geqslant 1}$ or recogniser membrane systems that recognises $L \in \mathsf{FAC}^0$-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$. Let $h\colon \{0,1\}^* \mapsto \mathbf{\Pi}$ be the semi-uniformity function of $\mathbf{\Pi}$, that is, on input $x \in \{0,1\}^*$, $\Pi_x = h(x)$ accepts iff $x \in L$. We present a non-deterministic logspace Turing machine M that recognises L.

The computation of M proceeds as follows: First M, on input x, non-deterministically chooses a single object from $\mathcal{C}_1$, the initial configuration of Π_x, and stores (a string representation of) the object and its containing membrane on its work tape. Then M enters a loop where at each iteration it non-deterministically chooses one of the rules applicable to the object on its work tape. If the rule is of type (a) or (e) (Definition 6) then M replaces the current object on the work tape (the membrane remains unchanged) with a non-deterministically chosen object from the right hand side of the rule. If the rule is of type (b) or (c) then the object on the work tape is replaced by the object on the right hand side of the rule and the membrane on the work tape is replaced by the parent (type (c)) or child membrane (type (b)) of the current membrane. If during the computation the work tape is found to store the object ***yes*** in the *env* membrane then $M(x)$ halts and accepts. Otherwise, if there are no rules applicable to the object and membrane on the work tape, and it is not ***yes*** in *env*, then $M(x)$ halts and rejects.

Suppose that $x \in L$ and so $\Pi_x = h(x)$ accepts. This implies that there is one (or more) objects in the initial configuration of Π_x that will, by the application of rules to this object and its successors, become the object ***yes*** in the *env* membrane by the end of the computation of Π_x (this claim follows from the kind of rules we allow—they are essentially context free—and can be formally proven using dependency graphs [21]). Indeed, this observation holds for all three kinds of membrane systems: acknowledger, recogniser$_{\geqslant 1}$ and recogniser. By non-deterministically choosing an object in the initial configuration, and non-deterministically choosing the rules that are applied to this object and its successors we ensure that there is a computation of $M(x)$ for each possible sequence of rule applications of Π_x for each object in the initial configuration Π_x (this follows from Lemma 13). Therefore at least one computation of $M(x)$ will produce the object *yes* in the *env* membrane and so $M(x)$ accepts, by confluence. That is, if Π_x accepts then M accepts on input x.

Suppose that $x \notin L$ and so $\Pi_x = h(x)$ rejects. This implies that there is no valid computation of Π_x where an object in the initial configuration evolves to *yes* in the *env* membrane. Indeed, this observation holds for all three kinds of membrane systems: acknowledger, recogniser$_{\geqslant 1}$ and recogniser. In this case *all* computation branches of $M(x)$ will reach an object to which no further rules are applicable (that is not *yes*) and so will halt in the rejecting state. That is, if Π_x rejects then M rejects on input x.

To simulate the computation of Π_x in logspace, $M(x)$ recomputes relevant logarithmic sized pieces of $h(x) = \Pi_x$ via the classic technique for composing logspace algorithms (see Chapter 4.3 of [7]) each time it needs information about Π_x, i.e. initial configuration, rules, or membrane structure. From the statement, h is computable in FAC^0. This means that the number of unique objects and labels in Π_x are polynomial in $n = |x|$ and so each can be uniquely identified in binary with a string of length $\log n$. $M(x)$ uses a constant number of $\log n$ sized binary strings to encode the current object and membrane, as well as some counters and temporary storage needed to re-compute $h(x)$.

Therefore L is decided by a non-deterministic logspace Turing machine. $\square$

Lemma 19 ([37]). $\mathsf{NL} \subseteq \mathsf{FAC}^0$*-semi-uniform-*$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$*, for acknowledger and recogniser*$_{\geqslant 1}$ *membrane systems.*

Proof. Let $L \in \mathsf{NL}$. That is, there is a non-deterministic logspace Turing machine M with one or more accepting computation paths exactly for input words $x \in L \subseteq \{0,1\}^*$.

We show that there is an FAC^0 semi-uniform family of polynomial-time membrane systems $\mathbf{\Pi}$ that recognises L. We now describe a function $h\colon \{0,1\}^* \to \mathbf{\Pi}$, computable in FAC^0, such that if $x \in L$ then $h(x) = \Pi_x$ accepts, otherwise Π_x rejects.

Consider the configuration graph $G_{M,x}$ for M on input $x \in \{0,1\}^*$, which is FAC^0 computable from M and x (see Section 2.2 and Lemma 5). Also consider the Turing machine N_M (and its configuration graph $G_{N,x}$) that on input x accepts only if all computations of M reject on input x, that is, $x \notin L$. N_M uses the standard un-reachability algorithm [24, 56] for non-deterministic logspace.

The function $h(x)$ constructs the configuration graph $G_{M,x}$ and modifies it to produce a membrane system Π_x as follows. O, the set of unique objects of Π_x has an object encoding each vertex in the configuration graphs $G_{M,x}$ and $G_{N,x}$ as well as two extra objects, *yes* and *no*. The initial configuration of Π_x has a single membrane labelled *env* that contains two objects: c_i which encodes the initial configuration of $M(x)$; and c_j which encodes the initial configuration of $N_M(x)$. The edges of the configuration graphs $G_{M,x}$ and $G_{N,x}$ are encoded as object rewriting rules in the membrane system. If vertex u has k edges to vertices $v_1, \ldots, v_k$ then $h(x)$ encodes all k edges as a single type (a) rule: $[\, u \to v_1, \ldots, v_k \,]_{env}$. Let vertex (object) c_a encode the accepting configuration of the Turing machine M, and let $h(x)$ include the rule $[\, c_a \to \textit{yes} \,]_{env}$. Likewise for the vertex (object) c_b that encodes an accepting configuration of the Turing machine N_M, $h(x)$ includes the rule $[\, c_b \to \textit{no} \,]_{env}$.

We now argue that each member $\Pi_x = h(x)$ of the semi-uniform family $\mathbf{\Pi}$, accepts iff $x \in L$.

Suppose $x \in L$, therefore Turing machine $M(x)$ accepts. This implies that configuration graph $G_{M,x}$ has the property that there is a directed path from the vertex c_i representing the initial configuration, to the accept vertex c_a. The assumption also implies that $N_M(x)$ must reject, and so configuration graph $G_{N,x}$ does not have a directed path from the object c_j encoding its initial configuration to c_b, its accept configuration. Since $\Pi_x = h(x)$ directly encodes the configuration graphs as objects and rules then the existence of a path from c_i to c_a implies that the membrane system will produce the object *yes* during its computation. The absence of a path from c_j to c_b implies that the membrane system will not produce the object *no* during its computation. Therefore Π_x accepts if $x \in L$.

Suppose $x \notin L$, therefore no computation paths of Turing machine $M(x)$ accept. This implies that configuration graph $G_{M,x}$ has the property that there is no directed path from the vertex c_i representing the initial configuration, to the accept vertex c_a. The assumption also implies that $N_M(x)$ must accept, and so configuration graph $G_{N,x}$ has a directed path from the object c_j encoding its initial configuration to c_b, its accept configuration. Since $\Pi_x = h(x)$ directly encodes the configuration graphs as objects and rules then the existence of a path from c_j to c_b implies that the membrane system will produce the object *no* during its computation. The absence of a path from c_i to c_a implies that the membrane system will not produce the object *yes* during its computation. Therefore Π_x rejects if $x \notin L$.

Since each configuration graph is acyclic and has $p(|x|)$ nodes where p is some polynomial function, it follows that the membrane system itself is of polynomial size and halts in polynomial time. The configuration graph can be computed in FAC^0 by Lemma 5.

In conclusion, function h defines a semi-uniform family of polynomial time $\mathcal{AM}^0_{-d}$ recogniser$_{\geqslant 1}$ (and so also acknowledger) membrane systems that accept the language in $L \in \mathsf{NL}$. $\square$

Note that the above proof fails for recogniser membrane systems since if there is more than one accepting computation (or in the rejecting case, more than one rejecting computation) then multiple copies of the object *yes* (or *no*) are produced in violation of the definition of recogniser membrane systems.

5 The computational power of uniform families of acknowledger membrane systems

In this section we focus on acknowledger membrane systems (Definition 9) where the accepting condition is met by the presence of one or more *yes* object in the environment in the last step of a computation, and the absence of *yes* implies rejection. We give a characterisation of uniform families of acknowledger membrane systems:

Theorem 20. $(\mathsf{FAC}^0, \mathsf{FAC}^0)$*-uniform-*$\mathsf{PMC}\mathcal{AM}^0_{-d} = \mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL})$*, for acknowledger membrane systems.*

The proof of this result is the combination of Lemmas 21 and 22. Before giving the lemmas we first introduce the following FAC^0 computable functions that will be used in the proofs.

Pairing function We require an injective function that pairs two binary strings into one and is extremely easy (FAC^0) to compute. We use the pairing function that interleaves the bits of two binary string arguments a and b. For example, the binary strings $a = a_2a_1a_0$ and $b = b_2b_1b_0$ are paired as the interleaved string $\langle a, b\rangle = b_2a_2b_1a_1b_0a_0$. The circuits for interleaving and de-interleaving have only a single input gate layer and a single output gate layer (and so have 2 layers). The wiring between each input and output gate can be shown to be $\mathsf{DLOGTIME}$-uniform.

Binary to Unary There is a constant depth circuit family where circuit C_n takes as input some word $w \in \{0,1\}^n$ and outputs 1^x where x is the positive integer encoded in the first $\lceil \log_2 n\rceil$ bits of w [14]. It can be shown that this circuit family is $\mathsf{DLOGTIME}$ uniform and so this conversion from short binary strings to unary is in FAC^0.

Unary to Binary There is a constant depth circuit family where circuit C_n takes as input some word $w = 0^{n-x}1^x$ where $0 \le x \le n$, and outputs the binary encoding of x [14]. It can be shown that this circuit family is $\mathsf{DLOGTIME}$ uniform and so unary to binary conversion is in FAC^0.

Lemma 21. *$(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d} \subseteq \mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL})$, for acknowledger, recogniser$_{\geqslant 1}$ and recogniser membrane systems.*

Proof. Let $L \in (\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d}$. That is, there exist two functions $e, f \in \mathsf{FAC}^0$, such that e maps $x \in \{0,1\}^*$ to a multiset of membrane system objects (the input), and f maps $u \in \{1\}^*$ to a membrane system, $f(1^{|x|}) = \Pi_{|x|} \in \mathbf{\Pi}$, such that $\Pi_{|x|}$ accepts input $e(x)$ iff $x \in L$.

We claim that L is FAC^0 disjunctive reducible to a unary language T, where T is decided by a non-deterministic logspace Turing machine $\mathcal{T}$.

Let T be the set of words of the form $1^{\langle o, |x|\rangle}$ where, for all $|x| \in \mathbb{N}$ and then for all $o \in O_{|x|}$, membrane system $\Pi_{|x|} \in \mathbf{\Pi}$ accepts if object o in the input membrane eventually evolves to the object $\boldsymbol{yes}$ in the *env* membrane (where $O_{|x|}$ is the set of objects of $\Pi_{|x|}$, where both x and o are encoded in binary, $\langle\cdot,\cdot\rangle$ is the binary interleaving function defined at the start of Section 5, and 1^b denotes the unary word over $\{1\}$ of length b for a binary number b). Turing machine $\mathcal{T}$ decides words in T by first converting the input word to binary and then reversing the pairing function to find o_i and $|x|$. $\mathcal{T}$ then proceeds by simulating $\Pi_{|x|}$ in non-deterministic logspace using a similar method as described in Lemma 18, that is, by storing a constant number of objects and membranes on its work tape and recomputing $f(1^{|x|})$ as needed (the main difference is that $\mathcal{T}$ uses o_i as its starting object instead of non-deterministically choosing one). As in Lemma 18, $\mathcal{T}$ accepts if there exists a valid computation in $\Pi_{|x|}$ where o_i in the input membrane becomes $\boldsymbol{yes}$ in the *env* membrane. $\mathcal{T}$ rejects if there are no valid computations that lead to a $\boldsymbol{yes}$ object in the *env* membrane. Therefore

T is a tally language decided by a non-deterministic logspace Turing machine and so $T \in \mathsf{tallyNL}$.

We now define the function $\tau \in \mathsf{FAC}^0$, that maps from $\{0,1\}^*$ to the set of tuples of unary words, and later prove that if $x \in L$ then $\tau(x) \cap T \neq \emptyset$, otherwise if $x \notin L$ then $\tau(x) \cap T = \emptyset$. Let $\tau(x) = (u_1, \ldots, u_{q(|x|)})$, where $q(|x|)$ is the number of object types o_i in $e(x)$, and $u_i = 1^{\langle o_i, |x| \rangle}$. Note that the set of unique words in $\tau(x)$ is a bijection onto the set of objects $e(x)$ so $q(|x|)$ is polynomial in $|x|$. Since e, the pairing function, binary-unary conversions, as well as calculation of $q(|x|)$ are in FAC^0, it is not difficult to see that $\tau \in \mathsf{FAC}^0$.

We now prove that τ is a disjunctive reduction from L to T. Suppose $x \in L$, this implies that *at least one* of the objects in $e(x)$, when placed in the input membrane of $\Pi_{|x|}$ evolves to a $\boldsymbol{yes}$ object in the *env* membrane by the end of the computation of $\Pi_{|x|}$. Then, by the definition of τ, if $x \in L$ then $\exists o \in \tau(x)$ such that $o \in T$.

Let $x \notin L$, this implies *none* of the objects in $e(x)$, when placed in the input membrane of $\Pi_{|x|}$, evolve to a $\boldsymbol{yes}$ object in the *env* membrane by the end of the computation of $\Pi_{|x|}$. Then, by the definition of τ, if $x \notin L$ then $\nexists o \in \tau(x)$ such that $o \notin T$. $\square$

Lemma 22. $\mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL}) \subseteq (\mathsf{FAC}^0, \mathsf{FAC}^0)$*-uniform-*$\mathsf{PMC}\mathcal{AM}^0_{-d}$*, for acknowledger membrane systems.*

Proof. Let $L \in \mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL})$. That is, there exists a unary language $T \subseteq \{1\}^*$ that is recognised by a non-deterministic logspace Turing machine $\mathcal{T}$, and a function $r \in \mathsf{FAC}^0$ that maps $x \in \{0,1\}^*$ to a set of unary words such that $r(x) \cap T \neq \emptyset$ if $x \in L$, and $r(x) \cap T = \emptyset$ otherwise. Let $q'(|x|) = \max(\{\max(|r(w)|) \mid w \in \{0,1\}^{|x|}\})$, that is, the length of largest word produced by r on any input of length $|x|$. Note that $q'(|x|)$ is computable by f since $r \in \mathsf{FAC}^0$.

We present an FAC^0 uniform polynomial-time $\mathcal{AM}^0_{-d}$ membrane family $\mathbf{\Pi}$ that recognises L. The family is composed of two functions: the uniformity function $f \colon \{1\}^* \to \mathbf{\Pi}$; and e that maps from binary words to the multiset of unique objects in the appropriate member of $\mathbf{\Pi}$.

Each member $\Pi_{|x|} = f(1^{|x|})$ of $\mathbf{\Pi}$ has one single membrane, *env*, that is both the environment and the input membrane. On input $1^{|x|}$, the function f produces a configuration graph $G_{\mathcal{T},u}$ for machine $\mathcal{T}$ on input 1^u for each $u \in \{1, 2, \ldots, q'(|x|)\}$. (Note that this is a generalization of the technique used in the proof of Lemma 19.) Since we have unary input words we can include the input word as part of the configuration to ensure that there is a unique input configuration for each $G_{\mathcal{T},u}$.

Each of the $q'(|x|)$ configuration graphs are converted to membrane rules and objects, using the same technique (without the second Turing machine that solves un-reachability) from the proof of Lemma 19, of a single membrane system $\Pi_{|x|}$. In summary, the vertices of the configuration graphs become objects in $\Pi_{|x|}$ and the edges in the graph become type (a) rules. There is a type (a) rule that maps the object encoding the accepting configuration of $\mathcal{T}$ to $\boldsymbol{yes}$. We do not include the second Turing machine that solves un-reachability from Lemma 19. $\mathcal{T}$ is a logspace machine and so its configuration graph is of polynomial size, it follows that the membrane system is of polynomial size. It is relatively straightforward to verify that $f \in \mathsf{FAC}^0$.

The input encoder $e(x)$ simulates $r(x)$ to find the set of unary words $(u_1, \ldots, u_k)$, then outputs an object $c_{i,u}$ for each $u \in r(x)$, which encode the vertex of the configuration graph corresponding to the initial configurations of Turing machine $\mathcal{T}$ input u. Since $r \in \mathsf{FAC}^0$ it is not difficult to see that $e \in \mathsf{FAC}^0$.

We now show that the membrane system $\Pi_{|x|}$ on input $e(x)$ accepts if $x \in L$ and otherwise rejects.

Suppose $x \in L$. This implies that at least one word in $r(x)$ is in the tally set T and so $\mathcal{T}$ accepts on at least one of these inputs. The input membrane of $\Pi_{|x|}$ contains $e(x)$ which includes the object $c_{i,u}$ which encodes the configuration graph vertex that represents the initial configuration of Turing machine $\mathcal{T}$ on input 1^u. In the proof of Lemma 19 we show how the construction of $\Pi_{|x|}$ is such that there is a sequence of rules from the input object $c_{i,u}$ to the ***yes*** object and so $\Pi_{|x|}$ on input $e(x)$ will accept.

Suppose $u \notin L$. This implies that none of the unary words $r(x)$ are in the tally set T and that $\mathcal{T}$ does not have any accepting computations on any of the words 1^j in $r(x)$. So, as in the proof of Lemma 19, this implies that none of the objects in $e(x)$ in the input membrane of $\Pi_{|x|}$ can evolve to the object ***yes*** in the *env* membrane. In this case the membrane system $\Pi_{|x|}$ on input $e(x)$ will halt without ***yes*** object; a rejecting computation for an acknowledger membrane system.

Therefore the pair of functions f and e provide a uniform family of polynomial time $\mathcal{AM}^0_{-d}$ membrane systems that accept $L \in \mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL})$. □

6 The computational power of recogniser$_{\geqslant 1}$ membrane systems

In this section we further investigate how the details in the definition of acceptance and rejection for recogniser membrane systems affect the computational power of uniform families of $\mathcal{AM}^0_{-d}$ systems.

In Section 5 we consider acknowledger membrane systems (Definition 9) where the absence of a ***yes*** object in the environment in the last step of any computation of a membrane system is sufficient to say that the system rejected its input. However, if we restrict to recogniser$_{\geqslant 1}$ membrane systems, which must produce one or more ***yes*** objects in the case of an accepting computation and one or more ***no*** objects in the case of a rejecting computation (Definition 8) it is no longer clear if our characterisation of $(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d}$ for acknowledger systems can still hold. The best lower-bound we find is $\mathsf{FAC}^0_{\mathrm{m}}(\mathsf{tallyNL})$, and we obtain upper-bounds of $\mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL})$ and $\mathsf{FAC}^0_{\mathrm{ctt}}(\mathsf{tallyNL})$.

In the semi-uniform case the upperbound FAC^0-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d} \subseteq \mathsf{NL}$ is unaffected by the restriction from acknowledger to recogniser$_{\geqslant 1}$ membrane systems. It also turns out that these more restricted recogniser$_{\geqslant 1}$ membrane systems have the same NL lower-bound on their power as acknowledger membrane systems (see Lemma 19).

Lemma 23. $(\mathsf{FAC}^0, \mathsf{FAC}^0)$*-uniform-*$\mathsf{PMC}\mathcal{AM}^0_{-d} \subseteq \mathsf{FAC}^0_{\mathrm{ctt}}(\mathsf{tallyNL})$*, for recogniser*$_{\geqslant 1}$ *and recogniser membrane systems.*

Proof. (Sketch) This proof closely follows that of Lemma 21 so we just highlight the differences. In the proof of Lemma 21 the language T is the set of words $1^{\langle o,|x|\rangle}$ where membrane system $f(1^{|x|}) = \Pi_{|x|}$ accepts if object o in the input membrane eventually evolves to the object ***yes*** in the *env* membrane. In this proof we consider the language T' that is the set of words $1^{\langle o,|x|\rangle}$ where in the membrane system $\Pi_{|x|}$ the object o does not evolve to the object ***no*** in the *env* membrane, in any computation. Via Lemma 13, this language is well-defined, i.e. can defined in terms of o and $|x|$. Also, Turing machine $\mathcal{T}$ from Lemma 21 (that solves reachability) can be modified [24, 56] to give $\mathcal{T}'$ (that solves unreachability) that accepts the language T'. That is, $\mathcal{T}'$ accepts if no object with the desired, and easy to check, property can be evolved by rule applications.

In Lemma 21 we defined the function $\tau \in \mathsf{FAC}^0$, that maps from $\{0,1\}^*$ to the set of tuples of unary words. Recall that $\tau(x)$ maps to a list that contains a unary string $1^{\langle o,|x|\rangle}$ for each o in $e(x)$. We now prove that τ is a conjunctive reduction from L to T'.

Assume $x \in L$, this implies that no object in $\Pi_{|x|}$ with input $e(x)$ eventually evolves to ***no*** in the *env* membrane. Hence $x \in L$ implies that $\forall w \in \tau(x), w \in T'$.

Assume $x \notin L$, this implies that *at least one* object in the initial configuration of $\Pi_{|x|}(e(x))$ eventually evolves a ***no*** object in the *env* membrane in each computation of $\Pi_{|x|}$. Hence $x \notin L$ implies that $\exists w \in \tau(x)$ such that $w \notin T'$. ☐

Lemma 24. $\mathsf{FAC}^0_{\mathrm{m}}(\mathsf{tallyNL}) \subseteq (\mathsf{FAC}^0, \mathsf{FAC}^0)$*-uniform-*$\mathsf{PMC}\mathcal{AM}^0_{-d}$*, for acknowledger and recogniser*$_{\geqslant 1}$ *membranes systems.*

Proof. Let $L \in \mathsf{FAC}^0_{\mathrm{m}}(\mathsf{tallyNL})$. That is, there exists a unary language $T \subseteq \{1\}^*$ that is recognised by non-deterministic logspace Turing machine $\mathcal{T}$, and a function $r \in \mathsf{FAC}^0$ that maps $x \in \{0,1\}^*$ to a unary word such that $r(x) \in T$ iff $x \in L$. Let $q(|x|) = \max(\{|r(w)| \mid w \in \{0,1\}^{|x|}\})$, that is, the largest word produced by r on any input of length $|x|$. Note that $q(|x|)$ is computable by f since $r \in \mathsf{FAC}^0$.

We present an FAC^0 uniform polynomial-time $\mathcal{AM}^0_{-d}$ membrane family $\mathbf{\Pi}$ that recognises L. The family is composed of two functions: $f\colon \{1\}^* \to \mathbf{\Pi}$, and e that maps each binary word to a multiset of objects from the appropriate member of $\mathbf{\Pi}$.

Each member $\Pi_{|x|} = f(1^{|x|})$ of $\mathbf{\Pi}$ has one single membrane, *env*, that is both the environment and the input membrane. On input $1^{|x|}$ the function f produces one configuration graph $G_{\mathcal{T},u}$ for machine $\mathcal{T}$ (that accepts T) on each input 1^u, $1 \le u \le q(|x|)$, and one configuration graph $G_{N,u}$ for machine $N_{\mathcal{T}}$ (that accepts the compliment of T) on each input 1^u, $1 \le u \le q(|x|)$. (Note that this is a generalization of the technique used in the proof of Lemma 19.)

Each of the $2q(|x|)$ configuration graphs are modified to give a set of rules and objects of a single membrane system $\Pi_{|x|}$ using the same technique as used in the proof of Lemma 19. In summary, the vertices of the configuration graphs become objects in $\Pi_{|x|}$ and the edges in the graph become type (a) rules. There is a rule mapping the object encoding the accepting configuration of $\mathcal{T}$ to ***yes*** and rule mapping object encoding the accepting configuration of $N_{\mathcal{T}}$ to ***no***. Both $\mathcal{T}$ and $N_{\mathcal{T}}$ are logspace machines and so their configuration graphs are of polynomial size and so the membrane system is of polynomial size. It is relatively straightforward to verify that $f \in \mathsf{FAC}^0$.

The input encoder $e(x)$ simulates $r(x)$ to find 1^u, then outputs two objects $c_{i,u}$ and $c_{j,u}$ which encode the vertex of the configuration graph corresponding to the initial configurations of Turing machines $\mathcal{T}$ and $N_{\mathcal{T}}$ respectively on input $1^u = r(x)$. Since $r \in \mathsf{FAC}^0$ it is not difficult to see that $e \in \mathsf{FAC}^0$.

We now show that the membrane system $\Pi_{|x|}$ on input $e(x)$ accepts if $x \in L$ and otherwise rejects.

Suppose $x \in L$. This implies that the word $r(x) = 1^u$ is in the tally set T and so at least one computation of $\mathcal{T}$ accepts 1^u. It also implies that there is no computation of $N_{\mathcal{T}}$ that accepts on input 1^u. The input membrane of $\Pi_{|x|}$ contains $e(x)$ which includes $c_{i,u}$ encoding the configuration graph vertex that represents the initial configuration of Turing machine $\mathcal{T}$ on input 1^u. In the proof of Lemma 19 we show how $\Pi_{|x|}$ has the property that there is a sequence of rules from the input object $c_{i,u}$ to the ***yes*** object and so $\Pi_{|x|}(e(x))$ will accept. Likewise there is no path from $c_{j,u}$ to ***no***.

Suppose $u \notin L$. This implies that the word $r(u) = 1^j$ is not in the tally set T and that therefore there is no accepting configuration of $\mathcal{T}$ on input 1^u, however, there is at least one accepting computation of $N_{\mathcal{T}}$ on the same input. In the proof of Lemma 19 we show how the construction of Π_n is such that there is a sequence of rules from the input object $c_{j,u}$ to the ***no*** object and so $\Pi_n(e(x))$ will reject. Likewise there is no path from $c_{i,u}$ to ***yes***.

Therefore the pair of functions f and e provide a uniform family of polynomial time $\mathcal{AM}^0_{-d}$ membrane systems that accept any language in $\mathsf{FAC}^0_{\mathrm{m}}(\mathsf{tallyNL})$. $\square$

7 Open Problems

The power of recogniser membrane systems. In Sections 4 and 5 of this paper we characterise the power of acknowledger membrane systems (Definition 9), which are a generalisation of recogniser membrane systems. In Section 6 we give upper and lower bounds on the power of the more restricted recogniser$_{\geqslant 1}$ membrane systems (Definition 8), which are closer in power to standard recogniser membrane systems. We also give upper bounds on the power of uniform and semi-uniform recogniser membrane systems (Definition 7), as well as showing that these classes are distinct.

However, we have not characterised the power of $\mathcal{AM}^0_{-d}$ recogniser membrane systems (Definition 7) with the kind of tight uniformity conditions used in this paper. In such systems, in an accepting computation exactly one ***yes*** object, or in a rejecting computation exactly one ***no*** object, is produced at the final step. A consequence of this is that our techniques for showing lower bounds on the power of acknowledger and recogniser$_{\geqslant 1}$ systems (Sections 4, 5 and 6) in terms of non-deterministic logspace-bounded Turing machines do not immediately carry over to recogniser systems.

As future work, we suggest that recogniser systems could be characterised via *unambiguous* non-deterministic logspace-bounded Turing machines [6]. An unambiguous machine accepts an input if and only if it has exactly one accepting computation. Perhaps the class of problems solved by semi-uniform families of recogniser $\mathcal{AM}^0_{-d}$ systems, i.e. FAC^0-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d}$, does not contain NL-complete problems since the system cannot control how many *yes* objects it produces? Perhaps these semi-uniform recogniser systems can solve s-t connectivity for "mangrove" graphs, i.e. graphs where there is exactly one path between each pair of vertices which is contained in unambiguous logspace [6]? Formally, we conjecture that FAC^0-semi-uniform-$\mathsf{PMC}^*\mathcal{AM}^0_{-d} = \mathsf{RUSPACE}(\log n)$ [6]. We also conjecture that for the analogous uniform families of recogniser systems $(\mathsf{FAC}^0, \mathsf{FAC}^0)$-uniform-$\mathsf{PMC}\mathcal{AM}^0_{-d} = \mathsf{FAC}^0_{\mathrm{m}}(\mathsf{RUSPACE}(\log n))$. If proven, our conjectures, taken together with previous results [6], would give a restatement of the relationship between the classes L, unambiguous logspace and NL in the membrane computing model, as well as the other classes shown in Figure 1.

Tight uniformity conditions for other classes of membrane systems. In this paper and others [33, 34, 35, 36, 37], we have put forward the idea of exploring the power of membrane systems under tight uniformity conditions. Others have since carried on this line of investigation [46]. Besides the main result in this paper (exhibiting systems where uniformity is a strictly weaker notion than semi-uniformity) this has led to various other characterisations of the power of a variety of classes of membrane systems and a teasing apart of their power. A number of other varieties of membrane systems (e.g. [20, 40]) characterise the complexity class P, but where the lower-bound actually depends on the use of P uniformity. As future work, it would be interesting to investigate these, and other, systems under suitably tight notions of uniformity or semi-uniformity.

Upper-bounding $\mathsf{tallyNL}$. While we know that $\mathsf{tallyNL} \subsetneq \mathsf{NL}$ it would be interesting to find other classes to upper bound $\mathsf{tallyNL}$. It is known that if a sparse language is complete for NL then $\mathsf{NL} \subseteq \mathsf{DLOGTIME}$ uniform-TC^0 [13, 23]. Is it possible to show that $\mathsf{tallyNL} \subseteq \mathsf{DLOGTIME}$ uniform-TC^0?

Classes reducible to $\mathsf{tallyNL}$. We conjecture that $\mathsf{FAC}^0_{\mathrm{m}}(\mathsf{tallyNL})$ is strictly contained in $\mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL})$. Giving an exact characterisation of recogniser$_{\geqslant 1}$ membrane systems studied in Section 6 may provide some insights into this. We also conjecture that $\mathsf{FAC}^0_{\mathrm{dtt}}(\mathsf{tallyNL}) \neq \mathsf{FAC}^0_{\mathrm{ctt}}(\mathsf{tallyNL})$. A lead to solve this may come from Ko [26] who showed that $\mathsf{P}_{\mathrm{ctt}}(\mathsf{tally}) \neq \mathsf{P}_{\mathrm{dtt}}(\mathsf{tally})$.

Acknowledgements

Thanks to an anonymous reviewer for a thorough reading and helpful comments. This paper appears in a special issue dedicated to Mario de Jesús Pérez-Jiménez on this 65th birthday; we thank Mario for giving us the opportunity to enjoy Sevilla and its wonderful cortados.

References

[1] Adleman, L.: Molecular computation of solutions to combinatorial problems, *Science*, **266**, 1994, 1021–1024.

[2] Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-Complete Problem by Recognizing P Systems with Restricted Active Membranes, *Fundamenta Informaticae*, **58**(2), 2003, 67–77.

[3] Alhazov, A., Pan, L.: Polarizationless P Systems with active membranes, *Grammars*, **7**, 2004, 141–159.

[4] Alhazov, A., Pérez-Jimnez, M. J.: Uniform Solution to QSAT Using Polarizationless Active Membranes, in: *Machines, Computations and Universality (MCU)*, vol. 4664 of *Lecture Notes in Computer Science*, Springer, 2007, 122–133.

[5] Allender, E., Koucký, M.: Amplifying lower bounds by means of self-reducibility, *Journal of the ACM*, **57**, March 2010, 14:1–14:36.

[6] Allender, E., Lange, K.-J.: RUSPACE$(\log n) \subseteq$ DSPACE$(\log^2 n/\log\log n)$, *Theory of Computing Systems*, **31**(5), 1998, 539–550.

[7] Arora, S., Barak, B.: *Computational Complexity: A Modern Approach*, Cambridge University Press, 2009, ISBN 978-0-511-53381-5.

[8] Barish, R. D., Rothemund, P. W. K., Winfree, E.: Two computational primitives for algorithmic self-assembly: copying and counting, *Nano Lett.*, **5**, 2005, 2586–2592.

[9] Barish, R. D., Schulman, R., Rothemund, P. W. K., Winfree, E.: An information-bearing seed for nucleating algorithmic self-assembly, *PNAS*, **106**(15), 2009, 6054–6059.

[10] Book, R. V., Ko, K.-I.: On sets truth-table reducible to sparse sets, *SIAM Journal of Computing*, **17**(5), 1988, 903–919.

[11] Borodin, A.: On relating time and space to size and depth, *SIAM Journal on Computing*, **6**(4), 1977, 733–744.

[12] Buhrman, H., Hemaspaandra, E., Longpre, L.: SPARSE reduces conjunctively to TALLY, *SIAM Journal of Computing*, **24**, June 1995, 673–681.

[13] Cai, J.-Y., Sivakumar, D.: Resolution of Hartmanis' conjecture for NL-hard sparse sets, *Theoretical Computer Science*, **240**(2), 2000, 257 – 269.

[14] Chandra, A. K., Stockmeyer, L. J., Vishkin, U.: Constant depth reducibility, *SIAM Journal of Computing*, **13**(2), 1984, 423–439.

[15] Chen, H.-L., Doty, D., Holden, D., Thachuk, C., Woods, D., Yang, C.-T.: Fast algorithmic self-assembly of simple shapes using random agitation, *DNA20: The 20th International Conference on DNA Computing and Molecular Programming*, LNCS, Springer, Kyoto, Japan, September 2014.

[16] Condon, A., Hu, A. J., Maňuch, J., Thachuk, C.: Less haste, less waste: on recycling and its limits in strand displacement systems, *Journal of the Royal Society – Interface focus*, **2**(4), 2012, 512–521.

[17] Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of Chemical Reaction Networks, in: *Algorithmic Bioprocesses*, Springer, 2009, 543–584.

[18] Dabby, N., Chen, H.-L.: Active self-assembly of simple units using an insertion primitive, *SODA: Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2012.

[19] Furst, M. L., Saxe, J. B., Sipser, M.: Parity, circuits and the polynomial-time hierarchy, *Theory of Computing Systems (formerly Mathematical Systems Theory)*, **17**(1), 1984, 13–27.

[20] Gutiérrez-Escudero, R., Pérez-Jiménez, M. J., Rius-Font, M.: Characterizing tractability by tissue-like P systems, in: *Workshop on Membrane Computing 10*, vol. 5957 of *Lecture Notes in Computer Science*, Springer, 2009, 269–181.

[21] Gutiérrez-Naranjo, M. A., Pérez-Jiménez, M. J., Riscos-Núñez, A., Romero-Campero, F. J.: Computational efficiency of dissolution rules in membrane systems, *International Journal of Computer Mathematics*, **83**(7), 2006, 593–611.

[22] Head, T., Rozenberg, G., Bladergroen, R. S., Breek, C. K. D., Lommerse, P. H. M., Spaink, H. P.: Computing with DNA by operating on plasmids, *Biosystems*, **57**(2), 2000, 87–93.

[23] Hesse, W., Allender, E., Mix Barrington, D. A.: Uniform constant-depth threshold circuits for division and iterated multiplication, *Journal of Computer and System Sciences*, **65**(4), 2002, 695–716.

[24] Immerman, N.: Nondeterministic space is closed under complementation, *SIAM Journal of Computing*, **17**(5), 1988, 935–938.

[25] Immerman, N.: *Descriptive Complexity*, Springer, 1999, ISBN 0387986006.

[26] Ko, K.-I.: Distinguishing conjunctive and disjunctive reducibilities by sparse sets, *Information and Computation*, **81**(1), 1989, 62–87.

[27] Ladner, R. E., Lynch, N. A., Selman, A. L.: A comparison of polynomial time reducibilities, *Theoretical Computer Science*, **1**(2), 1975, 103–123.

[28] Lipton, R. J.: DNA solution of hard computational problems, *Science*, **268**(5210), 1995, 542–545.

[29] Liu, Q., Wang, L., Frutos, A. G., Condon, A. E., Corn, R. M., Smith, L. M.: DNA computing on surfaces, *Nature*, **403**(6766), 2000, 175–179.

[30] Malchik, C., Winslow, A.: Tight Bounds for Active Self-Assembly Using an Insertion Primitive, *ESA: The 22nd European Symposium on Algorithms*, 2014, Accepted. Arxiv preprint `arXiv:1401.0359` [cs.FL].

[31] Mauri, G., Leporati, A., Porreca, A. E., Zandron, C.: Recent complexity-theoretic results on P systems with active membranes, *Journal of Logic and Computation*, 2013, (awaiting publication).

[32] Mix Barrington, D. A., Immerman, N., Straubing, H.: On uniformity within NC^1, *Journal of Computer and System Sciences*, **41**(3), 1990, 274–306.

[33] Murphy, N.: *Uniformity conditions for membrane systems: Uncovering complexity below P*, Ph.D. Thesis, National University of Ireland Maynooth, 2010.

[34] Murphy, N., Woods, D.: Active membrane systems without charges and using only symmetric elementary division characterise P, in: *8th International Workshop on Membrane Computing*, vol. 4860 of *Lecture Notes in Computer Science*, Springer, 2007, 367–384.

[35] Murphy, N., Woods, D.: A characterisation of NL using membrane systems without charges and dissolution, in: *Unconventional Computing, 7th International Conference, UC 2008, Vienna, Austria,*, vol. 5204 of *Lecture Notes in Computer Science*, Springer, 2008, 164–176.

[36] Murphy, N., Woods, D.: On acceptance conditions for membrane systems: characterisations of L and NL, *Proceedings of the International Workshop on The Complexity of Simple Programs*, 1, Electronic Proceedings in Theoretical Computer Science, 2009, Arxiv preprint: `arXiv:0906.3327v1` [cs.CC].

[37] Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions, *Natural Computing*, **10**(1), 2011, 613–632.

[38] Murphy, N., Woods, D.: AND and/or OR: Uniform polynomial-size circuits, *MCU: Proceedings of Machines, Computations and Universality*, 128, Electronic Proceedings in Theoretical Computer Science, 2013, Arxiv preprint: `arXiv:1212.3282v2` [cs.CC].

[39] Ouyang, Q., Kaplan, P. D., Liu, S., Libchaber, A.: DNA solution of the Maximal Clique Problem, *Science*, **278**(5337), 1997, 446–449.

[40] Pan, L., Pérez-Jiménez, M. J.: Computational complexity of tissue-like P systems, *Journal of Complexity*, **26**(3), 2010, 296–315.

[41] Papadimitriou, C. H.: *Computational Complexity*, Addison Wesley, 1993, ISBN 0201530821.

[42] Parberry, I.: *Circuit complexity and neural networks*, MIT Press, 1994, ISBN 0-262-16148-6.

[43] Păun, G., Rozenberg, G., Salomaa, A., Eds.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, Inc., New York, NY, USA, 2010, ISBN 0199556679, 9780199556670.

[44] Pérez-Jiménez, M. J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: *The Oxford Handbook of Membrane systems*, chapter 12: Complexity – Membrane Division, Membrane Creation, Oxford University Press, 2009, 302–336.

[45] Pérez-Jiménez, M. J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes, *Natural Computing*, **2**(3), 2003, 265–285.

[46] Porreca, A. E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes, in: *Proceedings of the 13th International Conference on Membrane Computing 2012*, vol. 7762 of *Lecture Notes in Computer Science*, Springer, 2013, 342–357.

[47] Post, E. L.: Recursively enumerable sets of positive integers and their decision problems, *Bulletin of the American Mathematical Society*, **50**(5), 1944, 284–316.

[48] Păun, G.: Membrane computing, in: *Fundamentals of computation theory*, vol. 2751 of *Lecture Notes in Computer Science*, Springer, 2003, 284–295.

[49] Păun, G.: Further twenty six open problems in membrane computing, *Proceedings of the Third Brainstorming Week on Membrane Computing*, Fénix Editoria, 2005.

[50] Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades, *Science*, **332**(6034), 2011, 1196.

[51] Rothemund, P. W., Winfree, E.: The program-size complexity of self-assembled squares, *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, ACM, 2000.

[52] Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks, *Natural Computing*, **7**(4), 2008, 615–633.

[53] Soloveichik, D., Winfree, E.: The computational power of Benenson automata, *Theoretical Computer Science*, **344**, 2005, 279–297.

[54] Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes, *SIAM Journal of Computing*, **36**(6), 2007, 1544–1569.

[55] Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers?, *Natural Computing*, **2**(3), 2003, 287–298.

[56] Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata, *Acta Informatica*, **26**(3), 1988, 279–284.

[57] Thachuk, C., Condon, A.: Space and energy efficient computation with DNA strand displacement systems, in: *18th International Conference on DNA Computing and Molecular Programming*, vol. 7433 of *Lecture Notes in Computer Science*, Springer, 2012, 135–150.

[58] Thachuk, C. J.: *Space and energy efficient molecular programming and space efficient text indexing methods for sequence alignment*, Ph.D. Thesis, University of British Columbia, 2013.

[59] Vollmer, H.: *Introduction to Circuit Complexity: A Uniform Approach*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999, ISBN 3540643109.

[60] Woods, D., Chen, H.-L., Goodfriend, S., Dabby, N., Winfree, E., Yin, P.: Active self-assembly of algorithmic shapes and patterns in polylogarithmic time, *ITCS'13: Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, ACM, 2013, Full version: `arXiv:1301.2626` [cs.DS].