

Solving QSAT in Sublinear Depth

Alberto Leporati¹, Luca Manzoni¹, Giancarlo Mauri¹,
Antonio E. Porreca^{1,2}, and Claudio Zandron¹

¹ Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy

{leporati, luca.manzoni, mauri, porreca, zandron}@disco.unimib.it

² Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
antonio.porreca@lis-lab.fr

Abstract. Among **PSPACE**-complete problems, QSAT, or *quantified SAT*, is one of the most used to show that the class of problems solvable in polynomial time by families of a given variant of P systems includes the whole **PSPACE**. However, most solutions require a membrane nesting depth that is *linear* with respect to the number of variables of the QSAT instance under consideration. While a system of a certain depth is needed, since depth 1 systems only allows to solve problems in $\mathbf{P}^{\#\mathbf{P}}$, it was until now unclear if a linear depth was, in fact, necessary. Here we use P systems with active membranes with charges, and we provide a construction that proves that QSAT can be solved with a *sublinear* nesting depth of order $\frac{n}{\log n}$, where n is the number of variables in the quantified formula given as input.

1 Introduction

The solution of the quantified SAT problem (QSAT) by means of P Systems with a “deep” membrane structure is a common way to prove that uniform families of P systems are able to solve all the problems in **PSPACE** [1,12]. Most of these approaches exploit the fact that, to verify the validity of a quantified formula φ , it is sufficient to “dedicate” a level of the membrane structure to a single quantifier, with each membrane acting either as an *or* gate (for existential quantifiers) or as an *and* gate (for universal quantifiers). The result is a membrane structure whose depth is, in the case of uniform families, linear with respect to the number of variables in φ .

We already know that part of the power of P systems with active membranes with charges is given by the depth of the membrane structure, that allows them to generate finely partitioned structures otherwise impossible to attain with depth 1 systems. In fact, depth 1 systems have been proved [2] to be limited to solve in polynomial time the problems in $\mathbf{P}^{\#\mathbf{P}}$, a class which is conjecturally smaller than **PSPACE**. Furthermore, even with constant depth the currently known problems that can be solved all reside inside the counting hierarchy [3] and we conjecture this inclusion to actually be an upper bound on the computational

power of constant depth P systems. Even in other models of P systems, like tissue P systems or depth 1 P systems with antimatter, the class $\mathbf{P}^{\#\mathbf{P}}$ provides a strict upper bound on the computational power [5,4].

Since the nesting depth of the membrane structure in the presence of division is such an important factor, it is somewhat surprising that there is a very large gap in our knowledge: the computational power endowed to P systems by a sublinear (but more than constant) depth is currently unknown. Existing characterisations show that those kinds of systems contain at least the whole counting hierarchy **CH** and are limited above by **PSPACE**. Their exact characterisation is, however, currently unknown. Also in traditional complexity theory the landscape of classes inhabiting the space between **CH** and **PSPACE** is surprisingly empty: does that mean that everything will collapse to **CH** or will reach **PSPACE** or that there are new, unknown, complexity classes that are naturally characterised by uniform families of P systems?

Here we start our investigation by proving that the linear depth (with respect to the number of variables) usually employed to solve QSAT is actually unnecessary and that, in fact a sublinear depth of $O(\frac{n}{\log n})$ is sufficient. We prove this result by “compressing” the usual membrane structure and delegating part of its duties to the internal membranes. Each one of them is, in fact, able to simulate a membrane sub-structure that is traditionally of logarithmic depth. While this first result seems, at a first glance, to suggest that even smaller depths might be sufficient to solve QSAT, we notice that it is unlikely that this same technique can be employed to further reduce the nesting depth: to obtain further results, different techniques might be needed.

2 Basic Notions

For an introduction to membrane computing and the related notions of formal language theory and multiset processing, we refer the reader to *The Oxford Handbook of Membrane Computing* [10]. Here we recall the formal definition of P systems with active membranes using weak non-elementary division rules [9,13].

Definition 1. A P system with active membranes with weak non-elementary division rules of initial degree $d \geq 1$ is a tuple

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \dots, w_{h_d}, R)$$

where:

- Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called objects;
- Λ is a finite set of labels;
- μ is a membrane structure (i.e., a rooted unordered tree, usually represented by nested brackets) consisting of d membranes labelled by elements of Λ in a one-to-one way;
- w_{h_1}, \dots, w_{h_d} , with $h_1, \dots, h_d \in \Lambda$, are multisets (finite sets with multiplicity) of objects in Γ , describing the initial contents of each of the d regions of μ ;

- R is a finite set of rules.

Each membrane possesses, besides its label and position in μ , another attribute called *electrical charge*, which can be either neutral (0), positive (+) or negative (–) and is always neutral before the beginning of the computation.

The rules in R are of the following types:

- (a) *Object evolution rules*, of the form $[a \rightarrow w]_h^\alpha$.
They can be applied inside a membrane labelled by h , having charge α and containing an occurrence of the object a ; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w) without changing the charge of h .
- (b) *Send-in communication rules*, of the form $a []_h^\alpha \rightarrow [b]_h^\beta$.
They can be applied to a membrane labelled by h , having charge α and such that the external region contains an occurrence of the object a ; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β .
- (c) *Send-out communication rules*, of the form $[a]_h^\alpha \rightarrow []_h^\beta b$.
They can be applied to a membrane labelled by h , having charge α and containing an occurrence of the object a ; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h becomes β .
- (e) *Elementary division rules*, of the form $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$.
They can be applied to a membrane labelled by h , having charge α , containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and charges β and γ ; the object a is replaced, respectively, by b and c , while the other objects of the multiset are replicated in both membranes.
- (f) *Weak non-elementary division rules*, of the form $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$.
They can be applied to a membrane labelled by h , having charge α , and containing an occurrence of the object a , even if it contains further membranes; the membrane is divided into two membranes having label h and charges β and γ ; the object a is replaced, respectively, by b and c , while the rest of the contents (including whole membrane substructures) is replicated in both membranes.

A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules: inside each membrane, several evolution rules can be applied simultaneously.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, or division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). Analogously, each membrane can only be subject to one communication or division rule (types (b)–(f)) per computation step; these rules will be called *blocking rules* in the rest of the paper. In other words, the only objects

and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.

- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously in an atomic way. However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps whereby each membrane evolves only after its internal configuration (including, recursively, the configurations of the membrane substructures it contains) has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane (the root of the tree) cannot be divided and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence $\mathcal{C} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$ of configurations, where \mathcal{C}_0 is the initial configuration, every \mathcal{C}_{i+1} is reachable from \mathcal{C}_i via a single computation step, and no rules of Π are applicable in \mathcal{C}_k .

P systems can be used as language *recognisers* by employing two distinguished objects *yes* and *no*: we assume that all computations are halting, and that either one copy of object *yes* or one of object *no* is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recogniser P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$. Each input x is associated with a P system Π_x deciding the membership of x in a language $L \subseteq \Sigma^*$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [7].

Definition 2. A family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$ is (polynomial-time) uniform if the mapping $x \mapsto \Pi_x$ can be computed by two polynomial-time deterministic Turing machines E and F as follows:

- $F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n with a distinguished input membrane.
- $E(x) = w_x$, where w_x is a multiset encoding the specific input x .
- Finally, Π_x is simply Π_n with w_x added to a specific membrane, called the input membrane.

The family $\mathbf{\Pi}$ is said to be (polynomial-time) semi-uniform if there exists a single deterministic polynomial-time Turing machine H such that $H(x) = \Pi_x$ for each $x \in \Sigma^*$.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This

restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [7] for further details on the encoding of P systems.

2.1 Polynomial Charges

As shown in [6], it is possible to expand the traditional model of membranes with charges by using a polynomial amount of charges instead of only the usual three. When a polynomial slowdown is acceptable, like in the situation under study, the traditional and the enhanced model have the same computational power with the same membrane nesting depth. Therefore, the construction provided here assumes the presence of a polynomial amount of charges, but the results also hold for the traditional model with three charges. In particular, in this extended model the definition of a P system is enriched with a finite set Ψ that defines which charges can assume a membrane; the traditional case, $\Psi = \{-, 0, +\}$. Since a polynomial-time uniform family of this kind of P systems is still constructed using a pair of polynomial-time deterministic Turing machines, also the size of the set of charges is polynomially bounded with respect to the input size. In the rest of the paper for clarity we will represent this extended charges with tuples of various lengths. Nonetheless, each tuple is to be considered as a unique object taken from the set of charges.

3 Construction

Let $\varphi = Q_1 Q_2 \dots Q_n \varphi^*$ be a quantified Boolean formula with n quantifiers over n variables $V = \{x_1, \dots, x_n\}$ and φ^* its non-quantified version in conjunctive normal form; i.e., $\varphi^* = C_1 \wedge C_2 \wedge \dots \wedge C_m$ where each C_j , for $1 \leq j \leq m$, is a disjunctive clause. In particular, we will deal with clauses composed of exactly three literals, in order to obtain a number of possible clauses that is polynomial with respect to the number of variables [8]. Finally, we denote by $\text{pos}(C_j)$ the set of variables that appear as positive literals in C_j and by $\text{neg}(C_j)$ the set of variables that appears in C_j as negative literals. The **PSPACE**-complete problem that will be solved by a uniform family of P systems is the *Quantified SAT* problem in its 3-CNF variant [8], where the output is the truthiness of a quantified formula in 3-CNF.

To encode the input formula φ we employ the following encoding:

- There are $8\binom{n}{3}$ bits, one for each possible clause. The i -th bit set to one means that the i -th clause (in lexicographic order) is present in φ .
- There are then n bits, one for each variable, where the i -th bit set to one means that x_i is universally quantified. If the i -th bit is set to 0 then x_i is existentially quantified.

The previous encoding allows to know from the length of the input the number of variables present in φ ; a similar encoding was already used in [11], which we refer to for additional details. Therefore, the Turing machine F of the uniformity condition is able to construct the P system with the knowledge of n and not only of the length of φ .

We are going to define a set of symbols, each one denoting a logarithmic number of quantifiers. We assume, without loss of generality, that the variables appear quantified in the order x_1, x_2, \dots, x_n , so that Q_i quantifies variable x_i . If this is not true a simple renaming of the variables can provide a formula satisfying this condition. We also assume that the number of variables n is divisible by $k = \lceil \log_2 n \rceil$; if this does not hold then we apply a padding, that in any case add no more than $2k$ variables universally quantified. Let $\ell = \frac{n}{k}$ and let Q_j for $1 \leq j \leq \ell$ be the string of quantifiers $Q_{(j-1)k+1}Q_{(j-1)k+2} \cdots Q_{jk}$ where each of the $\frac{n}{\log_2 n}$ sequences is made of k quantifiers. For example, for a formula $\exists x_1 \forall x_2 \forall x_3 \exists x_4 \forall x_5 \exists x_6 \varphi^*$ with 6 variables, $\lceil \log_2 6 \rceil = 3$, thus $Q_1 = \exists x_1 \forall x_2 \forall x_3$ and $Q_2 = \exists x_4 \forall x_5 \exists x_6$.

Notice that, for a fixed $j \in \{1, \dots, \ell\}$, the values that can be assumed by Q_j are only polynomial with respect to n : each of the quantifiers $Q_{(j-1)k+1}, \dots, Q_{jk}$ can only be universal or existential and the variable it quantifies over are only k in number. Therefore, Q_j can assume at most $2^k < 2n$ distinct values.

3.1 Initial Configuration of the P System

The initial structure of the P system constructed by machine F of the uniformity condition consists of $\ell + 1$ linearly nested membranes. The membranes are labelled from 1 (the outermost membrane) to $\ell + 1$ (the innermost membrane). Intuitively, the first ℓ non-elementary membranes will be used to evaluate the formula φ , while the innermost (elementary) membrane will be used to evaluate the formula φ^* for a given assignment.

For each $i \in \{2, \dots, \ell + 1\}$, the membrane with label i contains objects representing all the variables quantified in Q_{i-1} . For example, the membrane with label 2 contains, in the initial configuration, the objects x_1, \dots, x_k , the membrane with label 3 contains x_{k+1}, \dots, x_{2k} , and so on. Notice that, since the variables appear quantified in order and the number of variables can be obtained from the length of the input, this construction can be performed by machine F of the uniformity condition. Furthermore, the elementary membrane contains the object end.

The actual formula φ is encoded by two sets of symbols. The quantifiers are grouped in contiguous strings, each one containing k of them and encoded in the objects Q_1, \dots, Q_ℓ . For each clause C_1, \dots, C_k in φ^* the object $C_{i,i}$ is present, obtained by subscripting the clause C_i with i , its position inside the formula φ^* . For the sake of simplicity we will use the notation C_i in both cases, when no confusion arises. Both kinds of objects are placed in the outermost membrane.

3.2 Generation of the Assignments

The generation of the assignments is performed in multiple steps. First of all, each object Q_j representing the j -th string of quantifiers must be sent into membrane j , whereas the objects C_1, \dots, C_m representing the clauses of φ^* must be sent into the innermost membrane. Finally, the objects x_1, \dots, x_n trigger the necessary membrane divisions that allow us to obtain a full 2^k -ary tree of depth ℓ . As a result there will still be 2^n elementary membranes, one for each possible assignment.

At the beginning of the computation the objects Q_2, \dots, Q_ℓ are all sent in by rules of type 1. Once object Q_j enters the membrane with label j , by rules of type 2 it is rewritten to a junk object, changing the membrane charge to $(Q_j, 0)$, to record which quantifiers will be evaluated in that membrane and at the moment 0 objects representing clauses have entered the membrane. Object Q_1 is treated differently since it is already inside membrane 1, so it sets the charge by send out instead of being sent in (rule 3):

$$Q_j []_i^{(Q_i, 0)} \rightarrow [Q_j]_i^{(Q_i, 0)} \quad \text{for } 1 < j < i \leq \ell \quad (1)$$

$$Q_j []_j^0 \rightarrow [\#]_j^{(Q_j, 0)} \quad \text{for } 1 < j \leq \ell \quad (2)$$

$$[Q_1]_1^0 \rightarrow []_1^{(Q_1, k, 0, 0)} \# \quad (3)$$

The charge $(Q_1, k, 0, 0)$ represents the fact that the quantifiers in Q_1 needs to be evaluated, that currently the k -th one is the first that will be evaluated; the meaning of the other two values will be clarified in the following, when the evaluation procedure for the quantifiers is described.

Once all objects of type Q_j are in place, the objects C_1, \dots, C_m are sent in from the outermost to the innermost membrane in order:

$$C_i []_j^{(Q_j, q)} \rightarrow [C_i]_j^{(Q_j, q+1)} \quad \text{for } 1 < j \leq \ell, 0 \leq q < m-1, \text{ and } 1 \leq i \leq m \quad (4)$$

$$C_i []_{\ell+1}^q \rightarrow [C_i]_{\ell+1}^{q+1} \quad \text{for } 1 < j \leq \ell, 0 \leq q < m-1, \text{ and } 1 \leq i \leq m \quad (5)$$

$$C_i []_j^{(Q_j, m-1)} \rightarrow [C_i]_j^{(Q_j, k, 0, 0)} \quad \text{for } 1 < j \leq \ell \text{ and } 1 \leq i \leq m \quad (6)$$

$$C_i []_{\ell+1}^{m-1} \rightarrow [C_i]_{\ell+1}^{(n-k, 0)} \quad \text{for } 1 \leq i \leq m \quad (7)$$

Rules of types 4 and 5 send in the objects representing the first $m-1$ clauses. When an object C_i enters a membrane, it changes the charge to record that another clause has entered. The last clause to enter, when being sent into non-elementary membrane, it modifies the charge to allow for the next phase of the assignment generation to start (rules of type 6). When it enters the innermost membrane (rules of type 7) the charge is changed to record the number of objects of type t_i and f_i that must still enter the membrane before starting the evaluation of the assignment.

Once the charges of the non-elementary membranes have been set to the form $(\mathbf{Q}_j, k, 0, 0)$, the division process can start. Since all membranes except the outermost one already contain k objects representing the variables of φ , the charge set by the entrance of the object C_m is sufficient to trigger the applicability of the following rules:

$$[x_i]_j^{(\mathbf{Q}_j, k, 0, p)} \rightarrow [f_{i,k}]_j^{(\mathbf{Q}_j, k, 0, p)} [t_{i,k}]_j^{(\mathbf{Q}_j, k, 0, p+h(i,j))} \quad \text{for } 1 \leq j \leq \ell \text{ and } 1 \leq i \leq n \quad (8)$$

$$[x_i]_{\ell+1}^{(n-k,p)} \rightarrow [f_i]_{\ell+1}^{(n-k,p)} [t_i]_{\ell+1}^{(n-k,p+h(i,j))} \quad \text{for } 1 \leq i \leq n \quad (9)$$

where $h(i, j)$ is defined as $2^{k-(i-(j-1)k)}$. That is, p can be seen as a k -bit number, ranging from 0 to $2^k - 1$, where $h(i, j)$ sets to 1 the bit corresponding to the position of the variable x_i in the current string \mathbf{Q}_j of quantifiers. For non-elementary membranes, rules of type 8 perform the division and update the charge to allow each membrane resulting from the division to have a unique identifier p that can be used to order all the membranes with the same label j resulting from division that are located inside the same membrane. The objects $t_{i,k}$ and $f_{i,k}$ will be rewritten $k+1$ times before entering into the inner membranes. This allows for the division phase to be completely performed before the send-in happens. For elementary membranes, rules of type 9 perform the division, also adding the identifier p . The result of this division will be a complete tree with a branching factor of 2^k , due to the k division that happened inside each membrane.

Finally, it is necessary for the generated assignments to move into the elementary membranes to be evaluated. Since each non-elementary membrane has 2^k children, it is necessary to generate enough copies of each assignment to enter all the children membranes; this is made through rules of type 11. To avoid any conflict with the division process, each object representing the assignment waits for k steps before being duplicated using rules of type 10:

$$[\alpha_{i,t} \rightarrow \alpha_{i,t-1}]_j^{(\mathbf{Q}_j, k, 0, p)} \quad \text{for } 1 \leq t \leq k, 1 \leq i \leq n, 1 \leq j \leq \ell, \text{ and } \alpha \in \{t, f\} \quad (10)$$

$$[\alpha_{i,0} \rightarrow \underbrace{\alpha_i \cdots \alpha_i}_{2^k \text{ times}}]_j^{(\mathbf{Q}_j, k, 0, p)} \quad \text{for } 1 \leq j \leq \ell, 1 \leq i \leq n, \text{ and } \alpha \in \{t, f\} \quad (11)$$

Once the duplication process has been performed, the objects can actually be sent in, either in a non-elementary membrane (rules of type 12), where the rewriting process of rules of types 10 and 11 will be repeated, or inside an elementary membrane (rules of type 13), where the counter for the missing variable assignments present in the charge will be decremented by one:

$$\alpha_i []_j^{(\mathbf{Q}_j, k, 0, p)} \rightarrow [\alpha_{i,k}]_j^{(\mathbf{Q}_j, k, 0, p)} \quad \text{for } 1 \leq i \leq k, 1 \leq j \leq \ell, \text{ and } \alpha \in \{t, f\} \quad (12)$$

$$\alpha_i []_{\ell+1}^{(n-i,p)} \rightarrow [\alpha_i]_{\ell+1}^{(n-i-1,p)} \quad \text{for } 1 \leq i \leq n \text{ and } \alpha \in \{t, f\} \quad (13)$$

Notice that since the number of objects that can be involved in send-in rules present in each membrane is 2^k , which is also the number of membranes where they can enter, they distribute uniformly (i.e., one per membrane) across all children membranes. At the end of the process, once all assignments are inside the 2^n elementary membranes, the charge of all the elementary membranes will be of the form $(0, p)$. The appearance of this charge will make it possible to start the evaluation of the assignments in the next phase.

Notice how the entire process of generating the assignments requires only polynomial time. Each membrane performs at most k divisions (requiring k time steps) and each membrane receives (via send-in) only a polynomial amount of objects (bounded above by the sum of the number of variables, the number of clauses, and the number of quantifiers). Furthermore, the number of possible types of objects and charges remains polynomial during the entire process.

3.3 Evaluation of Assignments

The assignment evaluation is performed inside the elementary membranes. The main idea is that the objects representing the assignment exit one at a time, writing their value in the charge of the membrane. The objects representing the clauses of φ^* are rewritten according to the different charges. If all the clauses are satisfied once the last object of the assignment has been sent out, then that assignment satisfies φ^* . Otherwise, the assignment does not satisfy φ^* .

For all $p \in \{0, \dots, 2^k - 1\}$ the following rules are present in the system:

$$[\alpha_1]_{\ell+1}^{(0,p)} \rightarrow []_{\ell+1}^{(\alpha_1,p)} \# \quad \text{for } \alpha \in \{\mathbf{t}, \mathbf{f}\} \quad (14)$$

$$[\beta_i]_{\ell+1}^{(\alpha_{i-1},p)} \rightarrow []_{\ell+1}^{(\beta_i,p)} \# \quad \text{for } \alpha, \beta \in \{\mathbf{t}, \mathbf{f}\} \quad (15)$$

These rules send-out as junk all the objects representing the assignment of the formula φ^* , starting from the first one (rules of type 14), and then sending out all of them one at a time and in order (rules of type 15). This allows the following rules to “read” the assignment from the charge and rewrite an object representing a clause C_j if one of the variables has an assignment that satisfies it:

$$[C_j \rightarrow \mathbf{yes}'_p]_{\ell+1}^{(\mathbf{t}_i,p)} \quad \text{if } x_i \in \mathbf{pos}(C_j) \quad (16)$$

$$[C_j \rightarrow \mathbf{yes}'_p]_{\ell+1}^{(\mathbf{f}_i,p)} \quad \text{if } x_i \in \mathbf{neg}(C_j) \quad (17)$$

Once all the objects representing the assignment have been sent out, the object **end** is sent out to change the charge of the membrane using rules of type 18. Then the objects \mathbf{yes}'_p , representing the clauses that have been satisfied, wait one step by rewriting themselves in \mathbf{yes}_p using rules of type 19. At the same time, if at least one object representing one of the clauses has not been rewritten, it exits as $\mathbf{no}_{k,p}$, signalling that the assignment did not satisfy the formula φ^* , and sets the charge to $\#$, inhibiting the application of any other rule (rules of type 20). If this does not happen, then one of the \mathbf{yes}_p objects is sent out as $\mathbf{yes}_{k,p}$, signalling

that the assignment did satisfy φ^* (rules of type 21):

$$[\text{end}]_{\ell+1}^{(\alpha_m, p)} \rightarrow []_{\ell+1}^{\text{end}} \# \quad \text{for } \alpha \in \{\text{t}, \text{f}\} \quad (18)$$

$$[\text{yes}'_p \rightarrow \text{yes}_p]_{\ell+1}^{\text{end}} \quad (19)$$

$$[C_j]_{\ell+1}^{\text{end}} \rightarrow []_{\ell+1}^{\#} \text{no}_{k,p} \quad \text{for } 1 \leq j \leq m \quad (20)$$

$$[\text{yes}_p]_{\ell+1}^{\text{end}} \rightarrow []_{\ell+1}^{\#} \text{yes}_{k,p} \quad (21)$$

The evaluation of an assignment according to these rules requires a time that is linear with respect to the number of variables n . Furthermore, since all the involved quantities (e.g., possible values for p , number of clauses with three literals) are polynomial with respect to n , the number of rules to be defined is also polynomial. Therefore, the evaluation of the assignments requires a polynomial time and all rules necessary to perform it can be constructed in polynomial time.

3.4 Quantifiers

For the description of the following rules it is useful to introduce the concept of *quantification tree*, that is, the “shape” of the computation performed to actually decide if φ is a valid formula. This is a complete binary tree of depth n where to each internal level is associated a quantifier of φ . In particular Q_1 is associated to the root (depth 0), Q_2 is associated to the nodes at depth 1, and so on until the leaves are reached. The leaves are labeled with either t or f; the exact value depends on the satisfiability of φ^* given an assignment obtained by looking at the path from the root to the particular leaf of the tree (x_i is f if to reach depth i from depth $i - 1$ the path went to the left and t if it went to the right). To establish the validity of φ each internal node acts either as an \wedge gate (if the quantifier associated to that depth is a universal one) or as an \vee gate (if the associated quantifier is existential). In the following, we will say that the evaluation of the quantifiers moves up a level or moves to other siblings on the same level. This is to be interpreted as a movement on this quantification tree, which is the general method employed to establish the validity of φ by this algorithm. However, while usually this entire quantification tree is explicitly represented by the membrane structure, here it is partially present in the membrane structure and partially “simulated” by the sequential application of rules inside the non-elementary membranes. Since each level of the membrane structure of this construction “compresses” $\log_2 n$ levels of a traditional construction, the resulting tree has a depth that is reduced by a factor of $\frac{1}{\log_2 n}$, thus obtaining a depth of $O\left(\frac{n}{\log_2 n}\right)$.

The evaluation of the quantifiers is performed by alternating two steps: one internal to a membrane and one where the results are sent out to the parent membrane. Each non-elementary membrane receives from its 2^k children a result, either yes or no, of a partial evaluation of the formula φ . These results are numbered from 0 to $2^k - 1$ and are treated like the leaves of a complete binary tree of depth k , each level of the tree representing a different quantifier of Q_j ; that is, a fragment of the quantification tree. The results are combined two at a

time in a sequential manner, according to the quantifiers of Q_j . For example the first 2^k results are combined with \wedge or \vee (depending on the last quantifier of Q_j) to obtain 2^{k-1} results that will be further combined until a single one is produced. This result of the evaluation of all the quantifiers in Q_j is then sent out to the parent membrane. In the case of the outermost membrane, this is actually the result of evaluating the entire formula φ . While more involved, this procedure is similar to the one usually employed for solving QSAT with P systems with a linear depth with respect to n . The main difference is that, here, instead of exploiting a deeper membrane structure, it is necessary to “compress” some levels of the membrane structure and perform part of the evaluation sequentially inside a single membrane instead of using multiple nested membranes.

In the following, we assume that $0 \leq c < 2^r - 1$ and $0 \leq p < 2^k$. Every membrane with label $j \in \{1, \dots, \ell\}$ has associated the following type of rules:

$$[\alpha_{r,c}]_j^{(Q_j, r, c, p)} \rightarrow []_j^{(Q_j, r, c+1, p, \alpha)} \# \quad \text{for } 0 \leq r \leq k \text{ and } \alpha \in \{\text{yes, no}\} \quad (22)$$

Here the charge contains two indices, r and c , that indicate, respectively, that the current evaluation is of the r -th quantifier of Q_j and that the c -th result is the one that will be read. The values r and c can be interpreted as coordinates in (a fragment of) the quantification tree, where r denotes the depth and c a node among the ones at depth r . To complete the evaluation, the $(c+1)$ -th result should be combined with the c -th one, as follows:

$$[\beta_{r,c} \rightarrow \gamma_{r-1, \lfloor \frac{c}{2} \rfloor} \spadesuit]_j^{(Q_j, r, c, p, \alpha)} \quad \text{for } 1 \leq j \leq \ell \text{ and } \alpha, \beta \in \{\text{yes, no}\} \quad (23)$$

where γ is $\alpha \wedge \beta$ if the r -th quantifier in Q_j is a universal one and $\alpha \vee \beta$ if it is an existential one. Where α and β are two siblings in the evaluation tree and are combined to obtain the value γ of their parent. The object \spadesuit appearing in rules of type 23 is used to signal (by being sent out) that the object representing $\alpha \wedge \beta$ (for a universal quantifier) or $\alpha \vee \beta$ (for an existential quantifier) has been produced and, thus, that the evaluation can continue. This action is performed by the following rules:

$$[\spadesuit]_j^{(Q_j, r, c, p, \alpha)} \rightarrow []_j^{(Q_j, r, c+1, p)} \# \quad \text{for } 1 \leq j \leq \ell \text{ and } \alpha \in \{\text{yes, no}\} \quad (24)$$

$$[\spadesuit]_j^{(Q_j, r, 2^r - 1, p, \alpha)} \rightarrow []_j^{(Q_j, r-1, 0, p)} \# \quad \text{for } 1 \leq j \leq \ell \text{ and } \alpha \in \{\text{yes, no}\} \quad (25)$$

$$[\alpha_{0,0}]_j^{(Q_j, 1, 1, p)} \rightarrow []_j^\# \alpha_{k, p'} \quad \text{for } 1 < j \leq \ell \text{ and } \alpha \in \{\text{yes, no}\} \quad (26)$$

$$[\alpha_{0,0}]_1^{(Q_j, 1, 1, p)} \rightarrow []_1^\# \alpha \quad \text{for } \alpha \in \{\text{yes, no}\} \quad (27)$$

Rules of type 24 modify the charge to evaluate the next two results in the quantification tree while remaining at the same level. Once a level has been exhausted, the evaluation moves up by means of rules of type 25. Once the entire part of the quantification tree that is “simulated” inside a membrane of label j is exhausted, rules of type 26 move the result to the parent membrane. The result will be at the bottom (k -th) level of the quantification tree of the parent membrane and its position among all the other results will be given by value p

that was previously stored in the membrane charge. Finally, if the membrane where the evaluation ended is the outermost one, then the produced result is actually the result of the entire computation, and it is sent out as either **yes** or **no**.

Notice that the part of the quantification tree that is “simulated” by each non-elementary membrane is polynomial in the number of nodes that it contains. Therefore, the sequential evaluation performed in this construction still requires only polynomial time. Since among membranes at the same level all evaluations are performed in parallel, the time needed to produce the result of the entire computation multiplies this time by a factor that depends only on the depth of the membrane structure. Therefore, the P system resulting from this construction is able to produce an answer in a time which is polynomial with respect to the input size.

3.5 Main Result

The construction presented here shows that QSAT in its 3-CNF variant can be solved with a nesting depth that is sublinear with respect to the number of variables. This is the first solution, as far as the authors know, that goes below a linear nesting depth. In particular:

Theorem 1. *Uniform families of P systems with active membranes with charges and weak non-elementary division rules can solve the QSAT problem in 3-CNF form, where the quantified formula given as input has n variables, using a depth of $O\left(\frac{n}{\log n}\right)$ \square*

Even if the construction employed uses polynomial charges, instead of the usual 3, it has already been proved that one system can be converted into the other with only a polynomial slowdown and no increase in depth [6]. We want to remark that the availability of additional charges allowed a more compact and easier construction.

4 Conclusions

While solving **PSPACE**-complete problems, QSAT in particular, with P systems with active membranes with charges employing weak non-elementary division rules is not a new result, the construction provided here is the first one where the nesting depth of the membrane structure is sub-linear with respect to the number of variables in input. This is a first step in the direction of characterising the power of families of P systems with sublinear depth. While we have provided a construction reducing the depth needed in a specific problem (QSAT with a formula in 3-CNF), but it is still open what is the impact of this result in term of complexity classes. We want to remark that this construction cannot be directly employed to reduce the depth of the membrane structure below $O\left(\frac{n}{\log_2 n}\right)$, because we are already employing a polynomial number of charges,

types of objects, and rules. Employing the same method to further reduce depth, for example to $O\left(\frac{n}{(\log_2 n)^2}\right)$, would require a superpolynomial number of charges, object types, and rules – thus violating the uniformity condition of the family.

The remaining investigation work is vast: a complete characterisation of the families of constant depth is still in the work and other classes, like the one of families of logarithmic depth, are unexplored. Charting this unknown space of complexity classes is a long-term objective that will probably be necessary to attain in order to completely understand the complex interaction between nesting depth and computational power in P systems.

References

1. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae* 58(2), 67–77 (2003)
2. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: Gheorghe, M., Rozenberg, G., Sosik, P., Zandron, C. (eds.) *Membrane Computing*, 15th International Conference, CMC 2014. *Lecture Notes in Computer Science*, vol. 8961, pp. 284–299. Springer (2014)
3. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. *Fundamenta Informaticae* 138(1–2), 97–111 (2015)
4. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Characterising the complexity of tissue P systems with fission rules. *Journal of Computer and System Sciences* 90, 115–128 (2017)
5. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. *Theoretical Computer Science* 701, 161–173 (2017)
6. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: A toolbox for simpler active membrane algorithms. *Theoretical Computer Science* 673, 42–57 (2017)
7. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. *Natural Computing* 10(1), 613–632 (2011)
8. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley (1993)
9. Păun, G.: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics* 6(1), 75–90 (2001)
10. Păun, G., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press (2010)
11. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. *Natural Computing* 10(1), 167–182 (2011)
12. Sosik, P.: The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing* 2(3), 287–298 (2003)
13. Zandron, C., Leporati, A., Ferretti, C., Mauri, G., Pérez-Jiménez, M.J.: On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution. *Fundamenta Informaticae* 87, 79–91 (2008)