Shallow Laconic P Systems can Count*

Alberto Leporati[1], Luca Manzoni[1,3], Giancarlo Mauri[1],
Antonio E. Porreca[2], and Claudio Zandron[1]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo)
Università degli Studi di Milano-Bicocca
Viale Sarca 336, 20126 Milan, Italy. {leporati,mauri,zandron}@disco.unimib.it
[2] Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
antonio.porreca@lis-lab.fr
[3] Dipartimento di Matematica e Geoscienze, Università degli Studi di Trieste
Via Valerio 12/1, 34127 Trieste, Italy
lmanzoni@units.it

Abstract. Uniform families of shallow P systems with active membranes and charges are known to characterize the complexity class $\mathsf{P}^{\#\mathsf{P}}$, since this kind of P systems are able to "count" the number of objects sent out by the dividing membranes. Such a power is absent in monodirectional systems, where no send-in rules are allowed: in this case, only languages in $\mathsf{P}_{\|}^{\mathsf{NP}}$ can be recognized. Here we show that even a tiny amount of communication (namely, allowing only a single send-in per membrane during the computation) is sufficient to achieve the ability to count and solve all problems in the class $\mathsf{P}_{\|}^{\#\mathsf{P}}$, where all queries are performed independently.

1 Introduction

It is already known that bidirectional communication is necessary for uniform families of P systems with charges and only one level of membrane nesting, i.e., shallow, to solve problems in $\mathsf{P}^{\#\mathsf{P}}$ in polynomial time [1]. With monodirectional communication, only problems in $\mathsf{P}_{\|}^{\mathsf{NP}}$ can be solved and, even if polynomial depth in the membrane structure is allowed, only P^{NP} can be reached [3]. There exists, however, an entire spectrum of possibilities between monodirectional and full bi-directional communication; in particular, there are multiple ways of limiting the amount of bi-directional communication, for example by limiting the number of send-in rules that a membrane or its descendants (i.e., the membranes obtained from it by division) might apply during the computation. This is exactly the kind of limitation that we are imposing here, where each membrane is allowed only one send-in rule during the entire computation. Once the rule has been applied, neither that membrane nor any of the membranes obtained from it by division will apply another send-in rule in any future time step. We call P systems respecting this restriction *laconic*. Notice that we are not forbidding the application of multiple send-in rules, laconic P systems are a subclass of all

* This is an extended version of a paper presented at CMC20.

P systems in which in *every* reachable computation at most one send-in rule is applied for each membrane or one of its descendants.

Laconic P systems use, in some sense, the minimum amount of bidirectional communication; the inner membranes can "talk" with their ancestors (i.e., apply send-out rules) as much as they want, but the response that they will get from the outer membranes is limited to only one symbol. While this restriction is quite strong, we show that the remaining communication power is sufficient to allow the process of counting, thus enabling uniform families of shallow laconic P systems to solve problems in $\mathsf{P}_{\|}^{\#\mathsf{P}}$, a class that entirely includes the polynomial hierarchy [9]. This class is supposedly larger than the class $\mathsf{P}_{\|}^{\mathsf{NP}}$ of problems solvable by uniform families of shallow monodirectional P systems with active membranes. Therefore, even a small amount of communication produces a significant increase of the computational power of P systems. Contrarily to many other models, where $\mathsf{P}^{\#\mathsf{P}}$ is reached (see, for example, [1,2,4]), in this case we are only able to reach the conjecturally smaller complexity class $\mathsf{P}_{\|}^{\#\mathsf{P}}$.

One interesting aspect of this result is that the construction used to build a system simulating a Turing machine remains practically unchanged with respect to the one employed for monodirectional P systems. The only step when send-in is actually used is in the process of *counting*, which is actually the one described in details in the following. This shows that forbidding any kind of bidirectional communication does not hinder the ability of performing the simulation of a large number of parallel computations, but the seemingly "trivial" process of counting objects which is, instead, not trivial at all for a P system since it requires a minimal amount of bidirectional communication.

The results presented here are another step in our exploration of the computational power of bidirectional communication inside P systems, with the aim of understanding the "communication flows" of P systems and how they influence the ability to efficiently solve decision problems.

This paper is organized as follows: in Section 2 we recall some basic notions and we formally define laconic P systems. Section 3 describes the construction employed to prove the main result of the paper. In particular, the simulation of Turing machines employed for monodirectional systems is recalled in Section 3.1; the most delicate technical construction (and the main difference with respect to monodirectional systems) is the counting process described in Section 3.2; the main result is stated in Section 3.3. Finally, some directions of further investigation are detailed in Section 4.

2 Basic Notions

In this section we briefly recall some basic definitions about P systems and complexity classes; we also introduce the notion of *laconic* P systems.

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* [8], whereas for the definitions of the complexity classes used in this paper we refer to [6]. Here we just recall the formal

definition of P systems with active membranes using weak elementary division rules [7,10].

Definition 1. *A* P system with active membranes *of initial degree* $d \geq 1$ *is a tuple*

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

where:

- Γ *is a finite alphabet, i.e., a finite non-empty set of symbols, which are usually called* objects*;*
- Λ *is a finite set of labels;*
- μ *is a membrane structure, that is, a rooted* unordered *tree, usually represented by nested brackets, consisting of* d *membranes uniquely labelled by elements of* Λ*;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, 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 at the beginning of the computation.

The rules in R are of the following types:

(a) *Object evolution rules*, of the form $[a \to w]_h^\alpha$.
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).

(b) *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$.
They can be applied to a membrane labelled by h, having charge α and such that the external membrane contains an occurrence of the object a; the object a is sent into such membrane becoming b and, simultaneously, the charge of h is changed to β.

(c) *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$.
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 $[a]_h^\alpha \to [b]_h^\beta\, [c]_h^\gamma$.
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, respectively; the object a is replaced, respectively, by b and c, while the other objects of the multiset contained in membrane h are replicated in both membranes.

The instantaneous *configuration* of a membrane of label h consists of its charge α and the multiset w of objects it contains at a given time. It is denoted by $[w]_h^\alpha$. The *(full) configuration* $\mathcal{C}$ of a P system Π at a given time is a rooted, unordered tree; the root is a node corresponding to the external environment of Π, and has a single subtree corresponding to the current membrane structure of Π. Furthermore, the root is labelled by the multiset located in the environment, and the remaining nodes by the configurations $[w]_h^\alpha$ of the corresponding membranes. In the *initial configuration* of Π, the configurations of the membranes are $[w_{h_1}]_{h_1}^0, \ldots, [w_{h_d}]_{h_d}^0$.

A P system is *shallow* if it contains at most one level of membranes inside the outermost membrane. This means that all the membranes contained in the outermost membrane are elementary, i.e., they do not contain any other nested membrane.

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 an applicable evolution, communication, or division rule must be subject to exactly one of them. Analogously, each membrane can only be subject to one communication or division rule (types (b)–(e)) 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 or the application of further conflicting blocking rules.
- 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 microsteps whereby each membrane evolves only after their internal configuration (including, recursively, the configurations of the membrane substructures it contains) has been updated. In particular, 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 cannot be divided, and any object sent out from it cannot re-enter the system again. Hence, the environment only has a *passive* role and acts mainly as a place where the result of the computation can be collected.

A *halting computation* of a P system Π is a finite sequence $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ 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 $\mathcal{C}_k$. A *non-halting* computation $\vec{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as language *recognisers* by employing two distinguished objects yes and no: in this case 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. In this paper we deal with *confluent* P systems, for which all computations starting from the same initial configuration are either all accepting or all are rejecting.

In order to solve decision problems (or, equivalently, decide languages) over an alphabet Σ, we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x deciding the membership of x in a language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [5].

Definition 2. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ 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 its input membrane.*

The family $\boldsymbol{\Pi}$ 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 $x \in \Sigma^\star$.

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 that encoding of Π_x (e.g., the number of objects is not expressed in binary), 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 [5] for further details on the encoding of P systems.

Informally, a P system is said to be *laconic* if, in all its possible computations, the number of applications of send-in communication rules by each membrane is bounded above by one.

Let Π be a P system with active membranes with initial configuration $\mathcal{C}_0$. Let us associate to each membrane $[\,]_h$ in $\mathcal{C}_0$ a value $\textsc{in}([\,]_h)$ which is initially 0. The value $\textsc{in}([\,]_h)$ for any membrane in configuration $\mathcal{C}_i$ is defined recursively in the following way:

- If in the transition from $\mathcal{C}_{i-1}$ to $\mathcal{C}_i$ membrane $[\,]_h$ has applied a send-in rule, then its $\textsc{in}([\,]_h)$ value is increased by one.

– If $[\,]_h$ has been obtained by a membrane division applied in the step from $\mathcal{C}_{i-1}$ to $\mathcal{C}_i$ then its $\text{IN}([\,]_h)$ value is the same as the membrane that originated it.
– In all other cases the value of $\text{IN}([\,]_h)$ remains unchanged.

With this notion it is now possible to formally define a laconic P system.

Definition 3. *Let Π be a P system with active membranes and initial configuration $\mathcal{C}_0$. Π is said to be laconic if, for every possible configuration $\mathcal{C}_i$ attainable from $\mathcal{C}_0$, all membranes $[\,]_h$ that it contains are such that $\text{IN}([\,]_h) \leq 1$.*

Let us recall [6] that, for an oracle $\mathcal{O}$, the class $\mathsf{P}^{\mathcal{O}}$ corresponds to the class of all decision problems solvable by a Turing machine working in polynomial time with access to the oracle $\mathcal{O}$. The class $\mathsf{P}^{\mathcal{O}[1]}$ limits the number of oracle queries to one, while in $\mathsf{P}_{\|}^{\mathcal{O}}$ the Turing machine can perform a polynomial number of queries, but they must all be independent, that is, they can all be performed in parallel. Clearly, $\mathsf{P}^{\mathcal{O}[1]} \subseteq \mathsf{P}_{\|}^{\mathcal{O}} \subseteq \mathsf{P}^{\mathcal{O}}$, where the equality of the inclusions depends on the specific oracle chosen. As shown in [4], the complexity class $\mathsf{P}^{\#\mathsf{P}[1]}$ is the same as $\mathsf{P}_{\|}^{\#\mathsf{P}}$. That is, performing one oracle query or a polynomial number of them in parallel does not change the computational power of these systems, working with $\#\mathsf{P}$ oracles.

3 Simulation of Turing Machines with Counting Oracles

To show that shallow laconic P systems are able to solve all problems in $\mathsf{P}^{\#\mathsf{P}[1]}$ and, consequently, all problems in $\mathsf{P}_{\|}^{\#\mathsf{P}}$, we will simulate a deterministic Turing machine M with access to an oracle for a problem in $\#\mathsf{P}$, which is itself simulated in two steps. First of all, to answer the oracle queries we simulate a non-deterministic TM M' in the elementary membranes. This latter simulation, in which non-determinism is obtained by membrane division, produces as many yes objects as there are accepting computations. The number of yes objects might be exponential with respect to the size of the input and of the tape of machine M, which must be able to read the answer of its oracle query. Therefore, the P system will perform a conversion from unary to binary in order to write the query answer into the tape of machine M.

Without loss of generality, we assume that the first action performed by M is an oracle query. In fact, if this is not the case, the machine E of the uniformity condition can perform the simulation of M up to the point where the first (and only) query is performed. This allows us to simplify the construction of the simulation of M that takes place into the outermost membrane, as we do not need to specify all the operations necessary to start the query procedure. We also assume that both M and M' have a binary alphabet. This allows us to reuse the same TM simulation described in [3].

The P system will contain the following types of membranes:

– Membrane with label M is the skin membrane where the simulation of M is performed.

– Membranes with label M' perform the simulation of the oracle query by simulating M'.
– Membranes with label C_i are used for counting; in particular, the i-th digit being one in the result will be determined using membranes with label C_i.
– Membranes with label B_i are used to actually produce the i-th bit of the results by checking if all membranes of type C_i have been filled while at least one object of type yes_i still remains in the skin membrane. Here, for the conversion from unary to binary we rewrite each yes object into as many subscripted objects as the number of binary digits needed to represent the query answer.

The initial membrane structure of the P system is shown in Fig. 1. In the following we will describe the role and initial content of all the membranes, starting from the input membrane M', which simulates the oracle, and we will describe in details how the other elementary membranes perform the counting.

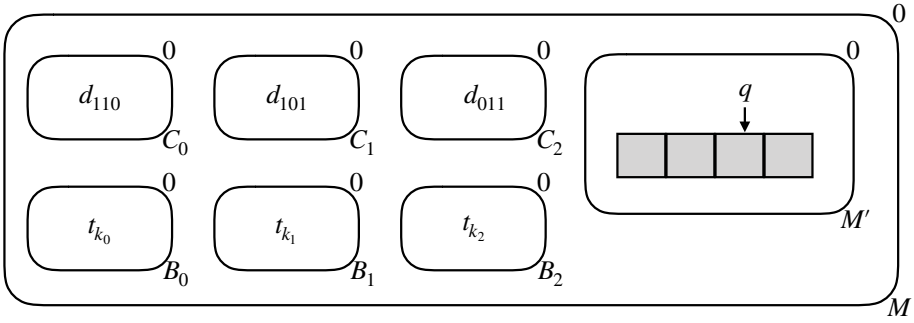

Fig. 1. The initial membrane structure of the P system

3.1 Turing Machine Simulation

To simulate a TM we employ the same construction used for monodirectional P systems in [3]. Here we just briefly recall the inner computing mechanism and the encoding used. Let Q be the set of states of the TM; without loss of generality we assume that $\Sigma = \{0, 1\}$ is the alphabet of the TM, and that $\delta\colon Q \times \Sigma \to Q \times \Sigma \times \{\triangleleft, \triangleright\}$ is the transition function.

As with the construction employed by monodirectional systems, we assume that we have a unique TM tape where in the even positions the tape of M is present, while odd ones contain the tape of M'; we will call this machine Sim. In the following, when we refer to either M or M' we are actually referring to Sim in a state where it simulates either M or M' and uses only half of the positions in the tape. Initially, membrane M' contains the entire initial configuration of the system, i.e., the objects encoding Sim, with the encoding of the tape of M'

already containing the oracle query. A copy of the initial encoding of Sim is then sent out into the skin membrane M in order to simulate the TM M. For the detail of how this send-out process can be performed, we refer the reader to [3], which employs a very similar method. We assume that M already has ℓ predefined positions (the size of the oracle answer to the query) named $s_0, \ldots, s_{\ell-1}$ that are set to zero and where M expects to find the answer to the oracle query, with s_0 the position of the least significant digit and $s_{\ell-1}$ the position of the most significant one. After this initial send-out process has been completed, the simulation of M' performed by Sim can start.

We now describe how Sim can be simulated by the rules in membrane M. The rules for the simulation of Sim inside membrane M' are the same except, obviously, for the membrane label and the addition of the weak elementary division rules corresponding to the non-deterministic transitions of M', as will be detailed in the following.

If $x_1, \ldots, x_n$ is the content of the tape of the machine, q its state, and the tape head is located in position i of the tape, then the configuration of the TM will be encoded in the following way: the state q is represented by the object q; if x_j is 0 then no object will be present (i.e., 0 is represented by the absence of an object), and if x_j is 1 then it will be represented by the object 1_{j-i}. That is, the subscript of the object 1_{j-1} represents the relative position of the cell j with respect to the tape head. More specifically, the initial configuration of the machine is also encoded as the initial content of them membrane with the aforementioned encoding. Each step of the TM is simulated by 7 steps of the P system by employing only evolution and send-out rules, according to the following procedure: first of all, the simulation of one step of the TM is triggered by the change of the membrane polarization from neutral to negative performed when the object $\ominus$ (initially contained in membrane M) is sent out by the following rule:

$$[\ominus]^0_M \to [\ ]^-_M \#$$

Once the charge of M has been changed to negative, all the objects representing the content of the TM tape are primed except for the one that might be present with subscript 0 (i.e., the one under the tape head):

$$[1_i \to 1'_i]^-_M \qquad \text{for } i \neq 0$$

In the same time step, the object under the tape head is sent out:

$$[1_0]^-_M \to [\ ]^+_M \#$$

If there is no object 1_0 in M, i.e., the symbol under the tape head was 0, then the previous rule cannot be applied, and the charge of M will remain negative. Therefore, by observing the polarization of the membrane, the object representing the state of the TM will be able to determine whether the symbol under the tape head was zero or one. This is performed by first rewriting the object:

$$[q \to q'\ \odot]^-_M \qquad \text{for } q \in Q$$

and then checking the charge of M by using two different kinds of evolution rules:

$$[q' \to (q,0)]^-_M \qquad \text{for } q \in Q$$
$$[q' \to (q,1)]^+_M \qquad \text{for } q \in Q$$

At the same time the object $\odot$ restores the neutral charge of M:

$$[\odot]^\alpha_M \to [\,]^0_M \,\# \qquad \text{for } \alpha \in \{+,-\}$$

Since all the information needed to perform the TM transition is now stored inside a single object, either of the form $(q,0)$ or $(q,1)$, it is now possible to simulate the application of the transition function δ (as specified in detail in [3]):

$$[(q,a) \to (r,b,d)]^0_M \qquad \text{where } \delta(q,a) = (r,b,d)$$

Simultaneously, the objects on the tape are primed again:

$$[1'_i \to 1''_i]^0_M \qquad \text{for } i \neq 0$$

The object of the form (r,b,d) is now "unpacked" into its components by one of the following four kinds of rules:

$$[(r,0,\triangleleft) \to \hat{r}\ \oplus]^0_M \qquad \text{for } r \in Q$$
$$[(r,0,\triangleright) \to \hat{r}\ \ominus]^0_M \qquad \text{for } r \in Q$$
$$[(r,1,\triangleleft) \to \hat{r}\ 1''_0\ \oplus]^0_M \qquad \text{for } r \in Q$$
$$[(r,1,\triangleright) \to \hat{r}\ 1''_0\ \ominus]^0_M \qquad \text{for } r \in Q$$

Each rule produces the new object representing the state, the new symbol on the tape (only if it is one, since zeros are encoded by the absence of objects), and the direction used to modify the subscripts of all the symbols encoding the TM tape.

The symbol $\oplus$ or $\ominus$ that is now contained in M is sent out, changing the charge of M to either positive or negative, respectively:

$$[\oplus]^0_M \to [\,]^+_M \,\# \qquad [\ominus]^0_M \to [\,]^-_M \,\#$$

At the same time, objects of type $\hat{r}$ are rewritten:

$$[\hat{r} \to \hat{r}'\ \odot]^0_M \qquad \text{for } r \in Q$$

Once the charge of M has changed to either positive or negative, the subscripts of the objects encoding the TM tape can be rewritten:

$$[1''_i \to 1_{i-1}]^-_M \qquad \text{for } -(n-1) \leq i \leq n-1$$
$$[1''_i \to 1_{i+1}]^+_M \qquad \text{for } -(n-1) \leq i \leq n-1$$

The charge of M is now set back to neutral:

$$[\odot]^\alpha_M \to [\,]^0_M \,\# \qquad \text{for } \alpha \in \{+,-\}$$

In the same step, object $\hat{r}'$ is rewritten into r and the object $\ominus$ is recreated:

$$[\hat{r}' \to r \ominus]_M^{\alpha} \qquad \text{for } r \in Q \text{ and } \alpha \in \{+, -\}$$

This completes the simulation of one computation step of the TM. Notice that by replacing the previous rule by one *not* producing $\ominus$, we can stop the TM simulation. Acceptance (resp., rejection) is then simply accomplished by sending out $\hat{r}'$ as yes (resp., no) when r is an accepting (resp., rejecting) state.

Non-deterministic choices are simulated, as usual, by performing membrane division instead of object evolution. This also allows to perform the simulation of machine M' by a similar set of rules. We remark that, once the simulation of a step has been completed, the simulation of the next step does not start as long as the charge of the simulating membrane remains neutral. This feature is useful to stop the simulation of the TM until a certain condition has been verified, such as having the result of an oracle query written on the TM tape.

We assume that, if a computation of M' accepts, then the object $\mathsf{yes}_{\mathcal{O}}$ is sent out to the skin membrane, representing the fact that the corresponding computation of machine M' has accepted. Notice that this might not be the first object sent out from the skin membrane, since "garbage" objects used to modify the charge of the skin membrane are used during the simulation of M. Finally, the simulation of machine M remains blocked until the skin membrane assumes negative charge, which happens when the result of the oracle query has been written into the TM tape, thus completing the query procedure.

3.2 Unary to Binary Conversion

The membranes M' simulating the nondeterministic TM, as described in the previous section and, in more details, in [3], send out as many $\mathsf{yes}_{\mathcal{O}}$ objects as the number of accepting computations. However, this means that the number is represented in unary and cannot be directly employed by the TM simulated in the skin membrane unless it can be written on the TM tape, which is only polynomial in length. Therefore, once all the $\mathsf{yes}_{\mathcal{O}}$ objects have reached the skin membrane, the system Π converts the number of those objects in binary.

Let m be the number of binary digits necessary to represent the maximum number of accepting computations of M. For example, $m = \log_2 2^{p(n)} = p(n)$ suffices. We now need to devise a simple technique to convert a unary number in the range from 0 to 2^m to a binary number of m bits. Let $p = b_{m-1}b_{m-2}\cdots b_1 b_0$ be the m-bit binary number representing the quantity of objects of type $\mathsf{yes}_{\mathcal{O}}$ present in the skin membrane. To find the value of b_i we need to check whether there are at least 2^i objects of type $\mathsf{yes}_{\mathcal{O}}$ once $\sum_{j=i+1}^{m} b_j 2^j$ objects have been removed.

To check if at least 2^i objects are present it is possible to use $2^i - 1$ membranes with the same label, in our case C_i, send-in one $\mathsf{yes}_{\mathcal{O}}$ object in each of them, and check if at least one object $\mathsf{yes}_{\mathcal{O}}$ still remains in the skin membrane after that. The most complex part of this process is the removal of $k = \sum_{j=i+1}^{m} b_j 2^j$ objects. To perform this step, we will set all bits b_j for $j > i$ to one, in order to

have the value of k known when we read the i-th bit. If, during this conversion procedure, we will encounter a bit b_j that is zero, we will write 0 to the tape of machine M, and then we will generate 2^j objects $\mathsf{yes}_{\mathcal{O}}$ to "set" the bit to one. Therefore, to check if the bit b_i is one we can use $k + 2^i - 1$ membranes where objects of type $\mathsf{yes}_{\mathcal{O}}$ are sent-in and check if, after that, at least one object $\mathsf{yes}_{\mathcal{O}}$ still remains in the outermost membrane. Notice that, under our assumption, $k + 2^i - 1$ is represented in binary as a m-bit number with all bits set to one except the i-th one. We will use this property to define the rules that generate exactly $k + 2^i - 1$ copies of the same membrane. Furthermore, since we have to repeat the binary conversion procedure detailed above for each bit, we will start by duplicating each object $\mathsf{yes}_{\mathcal{O}}$ into the objects $\mathsf{yes}_0, \ldots, \mathsf{yes}_{m-1}$, each used for determining the value of the corresponding bit of p. This means that the number of steps needed to perform this conversion is m times the maximum number of steps needed to determine a single digit of p, which is itself linear w.r.t. m; thus obtaining a quadratic upper bound w.r.t. m.

In the initial configuration, each membrane with label C_i contains a corresponding object having label d_w with $w = 1^{m-i-1}01^i$ which represents, in binary, the number of membranes of label C_i that will be created by membrane division. The first step is accomplished by the following rules:

$$[d_w]^0_{C_i} \to [d_{w'}]^0_{C_i}\ [d_{w''}]^0_{C_i} \tag{1}$$

where $w = 0^h1v$, $w' = 0^{h+1}v$, $w'' = 0^h10^{m-h-1}$ for $h \geq 0$, and $v \in \{0,1\}^{m-h-1}$ respects the regular expression 1^*01^*, since those are the only valid suffixes of strings of m bits where all but one bit are set to one. Rule (1) "splits" the bits of the subscript of d in such a way that, once no further rule of type (1) is applicable, the system applies

$$[d_w]^0_{C_i} \to [d_{w'}]^0_{C_i}\ [d_{w'}]^0_{C_i} \tag{2}$$

where $w = 0^h10^k$ and $w' = 0^{h+1}10^{k-1}$ with $h + k + 1 = m$ and $k > 0$. That is, rules of type (2) perform a division and a binary shift of the bits in the subscript until the subscript becomes $0^{m-1}1$. This means that if d_w with $w = 0^h10^k$ is present inside membrane C_i then the repeated application of the rules of type (2) will produce 2^k copies of C_i. A graphical depiction of this process is presented in Fig. 2. We want to recall that the number of distinct types of objects of the form d_w is polynomially bounded w.r.t. m:

- In rule (1), the word w has h zeros, followed by one or more ones, a zero, and, finally zero or more ones, for a total length of m. The word is uniquely determined by h and the length of the first "run" of ones, the total number of words of this form is quadratic w.r.t. m.
- In rule (2), the word w has the form 0^h10^k and a total length of m. Since the entire word is determined by the choice of h, the number of words of this form is linear w.r.t. m.

Since in rules (1) and (2) are uniquely determined by their left hand side, for each membrane label C_i there is a number of rules of these two kinds that is polynomially bounded by m.

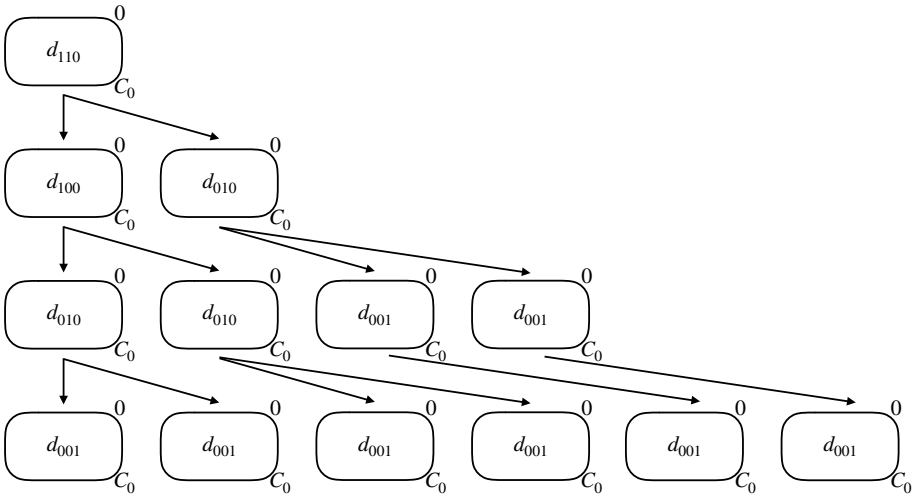

Fig. 2. How membrane division is used to create the correct number of membranes for the counting. Here the time increases from top to bottom and the arrows show if membrane division was applied (two arrows exiting a membrane) or not (a single arrow). In the case of 110 the subscript is "split" into single bits, and then additional membrane divisions are performed to obtain a total of six (110 in binary) identical membranes.

Once membranes with label M' have sent out as many objects of type $\mathsf{yes}_{\mathcal{O}}$ as the number of accepting computations, the following rules become applicable:

$$[\mathsf{yes}_{\mathcal{O}} \to \mathsf{yes}_0\ \mathsf{yes}_1\ \ldots\ \mathsf{yes}_{m-1}]^0_M \tag{3}$$

$$\mathsf{yes}_i\ [\ ]^0_{C_i} \to [\#]^+_{C_i} \qquad \text{for } 0 \le i < m \tag{4}$$

Using rule (3) each copy of the object $\mathsf{yes}_{\mathcal{O}}$ is rewritten into m subscripted copies corresponding to the different binary digits that we are going to produce. Each of the resulting copies is then sent-in into a membrane with label C_i (corresponding to the i-th digit of the result) by rules of type (4), which also change the charge of the membranes to $+$, to avoid further applications of the same rule: since the system we are producing is laconic, we must forbid multiple send-ins in the same membrane.

If all membranes with label C_i were "filled" and at least one object of type yes_i remains in the skin membrane, then the i-th bit of the result should be set to one. This task will be performed by the object contained in the membrane B_i, which will find if at least one object of type yes_i is still present in the environment thanks to the application of the rules:

$$\mathsf{yes}_i\ [\ ]^-_{B_i} \to [\#]^+_{B_i} \qquad \text{for } 0 \le i < m \tag{5}$$

Notice that the rule of type (5) for membrane B_i can be applied only after all possible applications of rules of type (4) have been performed, since membrane B_i

only assumes negative charge after that. This is accomplished with the following rules:

$$[t_{i,j} \to t_{i,j-1}]^0_{B_i} \qquad \text{for } 1 < j \leq k_i \text{ and } 0 \leq i < m \tag{6}$$

$$[t_{i,1} \to t_{i,0} N]^0_{B_i} \qquad \text{for } 0 \leq i < m \tag{7}$$

$$[N]^0_{B_i} \to [\,]^-_{B_i} \# \qquad \text{for } 0 \leq i < m \tag{8}$$

Each of the membranes B_i contains a timer object $t_{i,j}$ that is counting down from a predefined value k_i, that will be detailed later, to 0 using rules of type (6) for all steps except the last, and of type (7) for the last step. When the timer reaches 0 it changes the charge of B_i to negative using an auxiliary object and send-out rules of type (8). The following rules are then used to detect the presence or absence of objects of type yes_i in the skin membrane:

$$[t_{i,0} \to t'_{i,1}]^0_{B_i} \qquad \text{for } 0 \leq i < m \tag{9}$$

$$[t'_{i,1} \to t'_{i,0}]^-_{B_i} \qquad \text{for } 0 \leq i < m \tag{10}$$

$$[t'_{i,0}]^+_{B_i} \to [\,]^+_{B_i}\, \mathsf{yes}'_i \qquad \text{for } 0 \leq i < m \tag{11}$$

$$[t'_{i,0}]^-_{B_i} \to [\,]^-_{B_i}\, \mathsf{no}'_i \qquad \text{for } 0 \leq i < m \tag{12}$$

The timer waits with rules of types (9), which is applied at the same time the corresponding rule of type (8), and (10), which is applied simultaneously with the corresponding rule of type (5), before observing if the charge has changed to positive. If this is the case, then an object of type yes_i has entered the membrane and thus the i-th binary digit of the result should be 1: this is accomplished by using rules of type (11). Otherwise it should be 0 and rules of type (12) are applied to allow the further bookkeeping operations that are necessary to actually allow the correct determination of further digits of the result.

If the bit b_i of p is set to zero, then, according to the conversion procedure, we must set it to one. This is accomplished by generating 2^i objects of each of the types $\mathsf{yes}_{i-1}, \ldots, \mathsf{yes}_0$ by rewriting the object no'_i with the following rules:

$$[\mathsf{no}'_i \to f_{i,0}]^0_M \qquad \text{for } 0 < i < m \tag{13}$$

$$[f_{i,j} \to f_{i,j+1}\, f_{i,j+1}]^0_M \qquad \text{for } 0 < i < m \text{ and } 0 \leq j < i \tag{14}$$

$$[f_{i,i} \to \mathsf{yes}_{i-1}\, \mathsf{yes}_{i-2}\, \ldots\, \mathsf{yes}_0]^0_M \qquad \text{for } 0 < i < m \tag{15}$$

After the first rewriting step by rules of type (13), the actual duplication procedure is performed by the repeated application of rules of type (14). At the end of the duplication process, rules of type (15) produce the objects $\mathsf{yes}_{i-1}, \ldots, \mathsf{yes}_0$.

To determine the values of k_i necessary for the timer objects, there are two facts to take into account: each timer starts counting at the beginning of the computation and, before reaching 0 and changing the charge of the membrane B_i, the timer has to wait for all objects of type yes_i to enter membranes C_i to allow the application of rules of type (4). Assuming that the simulation of machine M' by the inner membranes of the same label requires $q(n)$ steps (which include the runtime $p(n)$ of M' and the slowdown needed to perform

the simulation), after $q(n)$ steps the objects of type yes are present in the skin membrane. They must then be rewritten into $\mathsf{yes}_{m-1}, \ldots, \mathsf{yes}_0$ and enter into membranes $C_{m-1}, \ldots, C_0$, respectively. Therefore, membrane B_{m-1} can assume negative charge after $q(m) + 2$ steps, which will be the value of k_{m-1}. Before changing the charge of membrane B_i, all membranes B_j for $j > i$ must have already produced their bit of answer. Each of them, once it acquires a negative charge must wait one step for an object of type yes_j to be sent-in, one step to produce either yes'_j or no'_j, and one step to possibly rewrite no'_j to $f_{j,0}$. Then, j steps are required to produce the objects $f_{j,j}$, another step to produce the objects $\mathsf{yes}_{j-1}, \ldots, \mathsf{yes}_0$ and finally one step to allow those objects to be sent-in into membranes $C_{j-1}, \ldots, C_0$, for a total of $5 + j$ steps. Therefore, k_i can be defined as $q(n) + 2 + \sum_{j=i+1}^{m-1}(5 + j)$.

To complete the conversion from unary to binary, we need to write the resulting binary number into the tape of M and change the charge of the skin membrane to negative to start the simulation of M. Both these tasks can be accomplished via the following rules:

$$[\mathsf{yes}'_i \to 1_{r_i}]^0_M \qquad \text{for } 0 < i < m \tag{16}$$

$$[\mathsf{yes}'_0 \to 1_{r_0}\ N]^0_M \tag{17}$$

$$[\mathsf{no}'_0 \to N]^0_M \tag{18}$$

$$[N]^0_M \to [\ ]^-_M\ \# \tag{19}$$

The rules of type (16) generate from the objects yes'_i the corresponding values 1_{r_i} that must appear on the TM tape. Notice that we do not need to perform any action for the bits that are 0 since they are encoded as the absence of an object. For the least significant bit, rule (17) is applied if it is one and rule (18) if it is set to zero. The additional object N is also produced, which is used by rule (19) to restart the simulation of machine M inside the skin membrane once the result of the oracle query is written into the TM tape.

3.3 Main Result

Let us notice that the previous construction enriches the one used for monodirectional systems with a way of counting the number of objects of a certain type inside the skin membrane. The construction of the rules, objects, and the initial membrane structure can be performed in polynomial time, thus allowing us to state the following result:

Theorem 1. *Uniform families of confluent* laconic *shallow P systems with active membranes with charges, using object evolution, send-in, send-out, and weak membrane division rules can solve all problems in* $\mathsf{P}^{\#\mathsf{P}[1]} = \mathsf{P}^{\#\mathsf{P}}_{\parallel}$. □

4 Conclusions

We have introduced a constrained class of P systems, called laconic P systems, that are an intermediate model between monodirectional P systems with active

membranes and traditional P systems with active membranes where bidirectional communication is not limited. Even if the amount of communication allowed in this kind of systems is extremely limited (at most one send-in rule per membrane is permitted during the computation), uniform families of confluent shallow laconic P systems can solve at least all problems in $\mathsf{P}_{\|}^{\#\mathsf{P}}$, which is supposedly larger than the class $\mathsf{P}_{\|}^{\mathsf{NP}}$ solved by their monodirectional counterpart. This shows that for this kind of systems the ability to "count" is deeply linked with the presence of bidirectional communication, which is a step in the direction of understanding the flows of communications inside P systems.

Possible future avenues of research include the complete characterization of the computational power of shallow laconic P systems and the study of their power in the case of deeper membrane structures, such as those having constant or linear depth. In particular, in the last case "classical" P systems characterize PSPACE, while monodirectional ones characterize P^{NP} and, while we expect laconic systems to be in the higher end of this spectrum, we still do not known in that case how being laconic influences the computational power. Another interesting research direction is exploring what problems can be solved with different bounds of IN. A bound of 0 implies monodirectionality, while the case of a bound of 1 has been explored in this paper. Can larger bounds allow to reach intermediate classes of problems between P^{NP} and PSPACE?

References

1. 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., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014. Lecture Notes in Computer Science, vol. 8961, pp. 284–299. Springer (2014), https://doi.org/10.1007/978-3-319-14370-5_18
2. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Tissue P systems can be simulated efficiently with counting oracles. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Membrane Computing, 16th International Conference, CMC 2015. Lecture Notes in Computer Science, vol. 9504, pp. 251–261. Springer (2015), https://doi.org/10.1007/978-3-319-28475-0_17
3. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing **15**(4), 551–564 (2016), https://doi.org/10.1007/s11047-016-9565-2
4. 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), https://doi.org/10.1016/j.tcs.2017.03.045
5. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing **10**(1), 613–632 (2011), https://doi.org/10.1007/s11047-010-9244-7
6. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
7. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics **6**(1), 75–90 (2001)
8. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)

9. Toda, S.: PP is as hard as the polynomial-time hierarchy. SIAM Journal on Computing **20**(5), 865–877 (1991), https://doi.org/10.1137/0220053
10. 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), http://content.iospress.com/articles/fundamenta-informaticae/fi87-1-06