Simulating counting oracles with cooperation

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

[1] Dipartimento di informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca,
Viale Sarca 336, 20126, Milan, Italy
alberto.leporati@unimib.it luca.manzoni@unimib.it
giancarlo.mauri@unimib.it claudio.zandron@unimib.it
[2] Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
antonio.porreca@lis-lab.fr

Summary. We prove that monodirectional shallow chargeless P systems with active membranes and minimal cooperation working in polynomial time precisely characterise $\mathbf{P}_{\|}^{\#\mathbf{P}}$, the complexity class of problems solved in polynomial time by deterministic Turing machines with a polynomial number of parallel queries to an oracle for a counting problem.

1 Introduction

Many variants of P systems with active membranes [8] are able to solve traditionally intractable problems: with charges and bidirectional communication, uniform families of them are able to solve $\mathbf{P}^{\#\mathbf{P}}$-complete problems when only one level of membrane nesting (i.e., shallow systems) is allowed [2, 3], and **PSPACE**-complete problems when this restriction is removed [9]. The presence of simple cooperation rules, like the ones provided by antimatter, where two opposite objects can annihilate each other, allows the systems to reach $\mathbf{P}^{\#\mathbf{P}}$ with a shallow membrane structure, also when the systems have no charges [5]. Even when the communication is severely restricted, as in monodirectional systems, where send-in is forbidden, uniform families of P systems with active membranes with charges characterize $\mathbf{P}^{\mathbf{NP}}$ or, if shallow, the class $\mathbf{P}_{\|}^{\mathbf{NP}}$, as shown in [4]. It is interesting to see that this is not the case for monodirectional systems with antimatter: the additional cooperation provided by object annihilation makes possible to "count" once, thus allowing families of this kind of systems the ability to reach $\mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}_{\|}^{\#\mathbf{P}}$, even with only one level of nesting [5].

In this paper we continue the investigation of the importance of cooperation to increase the computational power of P systems. In particular, we show that monodirectional systems with minimal cooperation [10] working in polynomial

time also characterize the class of all decision problems solvable in polynomial time by a deterministic Turing machine with access to a *single* query of a $\#\mathbf{P}$ oracle, i.e., $\mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}_{\|}^{\#\mathbf{P}}$.

The paper is organized as follows: Section 2 introduces the basic notions necessary for the rest of the paper. Section 3 shows how a single $\#\mathbf{P}$ query can be simulated, and in Section 4 the main result is presented. Section 5 contains the conclusions, and shows some directions for future research.

2 Basic Notions

In this paper we consider (semi)uniform families of P systems with active membranes without charges and using minimal cooperative evolution rules $[ab \to w]_h$, send-out rules $[a]_h \to [\ ]_h\ b$ and elementary division rules $[a]_h \to [b]_h\,[c]_h$, where a, b, and c are single objects and w is a multiset of objects. For the technical details we refer the reader to Valencia-Cabrera et al. [10].

We also consider polynomial-time Turing machines with oracles for counting problems in the complexity class $\#\mathbf{P}$ [7] and, in particular, the complexity classes $\mathbf{P}^{\#\mathbf{P}[1]}$, where only one query is allowed, and $\mathbf{P}_{\|}^{\#\mathbf{P}}$, when any polynomial number of queries is allowed, but they must all be carried out *in parallel*, that is, all query strings are prepared in advance before actually interrogating the oracle (in other words, later queries are not adaptive with respect to the answers to previous ones). The two classes $\mathbf{P}^{\#\mathbf{P}[1]}$ and $\mathbf{P}_{\|}^{\#\mathbf{P}}$ actually turn out to be equivalent:

Proposition 1 (Leporati et al. [5]). *A polynomial number of parallel $\#\mathbf{P}$ queries can be simulated by a single $\#\mathbf{P}$ query in polynomial time (in symbols $\mathbf{P}_{\|}^{\#\mathbf{P}} = \mathbf{P}^{\#\mathbf{P}[1]}$).*

Proof. A single query does never depend on the results of previous queries, thus $\mathbf{P}^{\#\mathbf{P}[1]} \subseteq \mathbf{P}_{\|}^{\#\mathbf{P}}$ by definition.

Conversely, let M be a deterministic Turing machine running in polynomial time $p(n)$ with parallel oracle queries for a function $f \in \#\mathbf{P}$, and let N be a nondeterministic Turing machine having $f(x)$ accepting computations for each input string x of length n and running in polynomial time $q(n)$.

Then $f(x) \leq 2^{q(|x|)}$ for each input string x, since $2^{q(n)}$ is the maximum number of computations of N on an input of length n (assuming binary nondeterministic choices). Clearly, due to its running time, the machine M can only ask queries with query strings of length bounded by $p(n)$, which means that each query answer is an integer bounded by $2^{q(p(n))}$, and M can ask up to $p(n)$ queries.

Let $x_1, x_2, \ldots, x_{p(n)}$ be the query strings of M on a run on a given input, letting $x_i = \epsilon$ if M asks less than i queries, and let $g\colon \Sigma^\star \to \mathbb{N}$, with Σ the union of the query alphabet and the separator symbol \$, be defined as

$$g(x_1\$x_2\$\ldots\$x_{p(n)}) = \sum_{i=1}^{p(n)} B^i \times f(x_i)$$

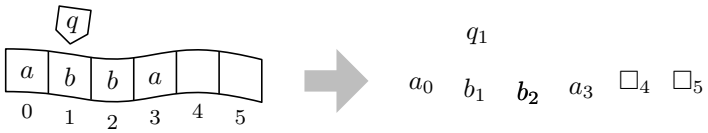
where $B = 2^{q(p(n))} + 1$; this corresponds to encoding all the query answers as a base-B integer. Then, a single query to g contains all the information that can be obtained by asking up to $p(n)$ parallel queries to f, since each value $f(x_i)$ can be recovered in polynomial time by computing

$$f(x_i) = \left\lfloor \frac{g(x_1\$x_2\$\ldots\$x_{p(n)})}{B^{i-1}} \right\rfloor \bmod B.$$

The function g is also in #**P**, since this class is closed under summations and products [1], and this proves $\mathbf{P}_{\|}^{\#\mathbf{P}} \subseteq \mathbf{P}^{\#\mathbf{P}[1]}$. □

3 Simulating a single #P query monodirectionally

It is quite easy to simulate efficiently (actually, in linear time) a deterministic Turing machine working in polynomial time, and thus using only a tape length, by means of a uniform family of P systems [6]. A configuration of the Turing machine can be encoded as a multiset of objects as follows:



that is, each symbol (including blanks) is subscripted by an index corresponding to the number of the tape cell, and the state of the machine is also represented as an object, subscripted by the index of the cell currently under the tape head. Blank tape cells are represented by the $\Box_i$ objects. Then, each transition of the machine, say $\delta(q, b) = (r, a, d)$ with $d = \pm 1$, is simulated by a set of cooperative evolution rules replicated for each legitimate tape position:

$$[q_i\ b_i \to r_{i+d}\ a_i]_h \qquad \text{for } 0 \le i < s(n) \tag{1}$$

where $s(n)$ is the polynomial space bound of the machine tape. These rules replace the symbol b_i under the tape head by a_i, and update the state symbol q_i to r_{i+d}, which also updates the position of the tape head.

A *nondeterministic* Turing machine can be simulated by dividing elementary membranes, replacing the rules (1) by

$$[q_i\ b_i \to \langle q_i, b_i\rangle]_h \qquad \text{for } 0 \le i < s(n) \tag{2}$$
$$[\langle q_i, b_i\rangle]_h \to [\langle r_{i+d}, a_i\rangle']_h\ [\langle s_{i+e}, c_i\rangle']_h \qquad \text{for } 0 \le i < s(n) \tag{3}$$
$$[\langle r_{i+d}, a_i\rangle' \to r_{i+d}\ c_i]_h \qquad \text{for } 0 \le i < s(n) \tag{4}$$
$$[\langle s_{i+e}, b_i\rangle' \to s_{i+e}\ c_i]_h \qquad \text{for } 0 \le i < s(n) \tag{5}$$

in the case of a nondeterministic transition such as $\delta(q, b) = \{(r, a, d), (s, c, e)\}$. The rules of type (2) "pack" the head-state object and the object representing the symbol under the tape head into a single object (which is necessary in order to respect the membrane division rule format). The rules of type (3) then perform the transition rule by dividing the membrane and rewriting the packed object into the two objects representing the two possible evolutions of the configuration, one in each of the resulting membranes; these objects are primed, in order to signal that they are the *result* of the transition and not the left-hand side. Finally, the two resulting objects are "unpacked" by the rules of type (4) and (5), thus obtaining the two possible Turing machine configurations inside the divided membranes. In order to avoid synchronisation issues due to the three-step simulation of a nondeterministic transition vs the one-step simulation of deterministic ones, we also slow down the latter accordingly, using the rules

$$\begin{aligned}
&[q_i\ b_i \to \langle q_i, b_i \rangle]_h && \text{for } 0 \leq i < s(n) \\
&[\langle q_i, b_i \rangle \to \langle r_{i+d}, a_i \rangle']_h && \text{for } 0 \leq i < s(n) \\
&[\langle r_{i+d}, a_i \rangle' \to r_{i+d}\ a_i]_h && \text{for } 0 \leq i < s(n)
\end{aligned}$$

in the case of a deterministic transition such as $\delta(q, b) = (r, a, d)$.

Thus, a single membrane (or a number of membranes obtained by division of a single initial one, in the case of nondeterminism) can efficiently simulate a polynomial-size tape Turing machine and, in particular, a Turing machine working in polynomial time. On the other hand, by using several nested membranes it is possible to efficiently simulate oracle queries [6]. With bidirectional P systems (i.e., standard P systems using both send-in and send-out rules) the simulation of the Turing machine is paused, then one usually sends the query string into a child membrane, where another Turing machine for the oracle language is simulated, possibly using membrane division; the answer is sent out and the simulation of the original Turing machine is resumed.

With monodirectional P systems we proceed in the opposite direction: we first duplicate and send out the multiset encoding the configuration of the Turing machine being simulated, then the oracle machine is simulated in the innermost membrane, and the result is sent out, where the simulation of the original Turing machine can then resume [6]. This process is depicted in Figure 1.

When the simulated Turing machine answering the query is nondeterministic, several result-objects yes are sent out from the divided membranes; they can be counted and operated upon by the Turing machine simulated in the external membrane by converting them in the binary representation of their multiplicity. This can be accomplished by using cooperative evolution rules as follows:

$$\begin{aligned}
&[\mathsf{yes} \to 1_0]_k \\
&[1_i\ 1_i \to 1_{i+1}]_k && \text{for } 0 \leq i < m
\end{aligned}$$

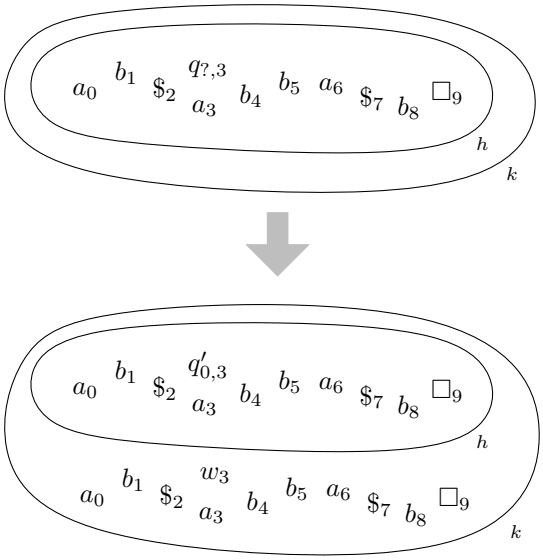


Fig. 1. Simulating an oracle query by means of monodirectional P systems. The portion of configuration delimited by \$ corresponds to the oracle query string, and the tape head is located on its first symbol. When the Turing machine being simulated inside membrane h enters the query state $q_?$, its configuration is duplicated and sent out. The head-state object inside membrane h now represents the state q'_0, the initial state of the Turing machine to be simulated in order to answer the oracle query, while the head-state object outside (in membrane k) represent a "dummy" symbol waiting for state w.

where m is the maximum number of bits for the answer, which can be computed as in the proof of Proposition 1. These rules produce the multiset of 1_i for all positions i where the number of yes objects produced as the answer to the query expressed in binary notation, contains a 1. By combining this with the multiset $0_m\ 0_{m-1}\ \cdots\ 0_1\ 0_0$ using the rules

$$[0_i\ 1_i \to 1_i]_k \qquad\qquad \text{for } 0 \leq i \leq m$$

i.e., by deleting the objects 0_i corresponding to the existing objects 1_i, we obtain the binary notation for the answer to the query, which can then be processed by the original Turing machine (now being simulated inside membrane k) as part of its tape.

The existence of the simulation described here, together with Proposition 1, prove the following result:

Theorem 1. *A deterministic polynomial time Turing machine with a polynomial number of queries to a* #**P** *problem can be simulated by shallow chargeless monodirectional P systems with active membranes and minimal cooperation rules in polynomial time.* □

4 Simulating P systems with a single #P query

The query used by the Turing machine that simulates a shallow chargeless P system with active membranes and minimal cooperation, working in polynomial time, is the following:

Query 1. *Given the description of a P system Π, an elementary membrane label h, an object type a and a time t in unary notation, how many objects of type a are collectively sent out by membranes with label h at time t?*

It is now necessary to prove that the answer to the query can be actually computed by a function in #**P**, i.e., it is a "valid" oracle query for the Turing machines that we are considering. Here we only give a sketch of the proof; all the details can be found in [5].

Lemma 1. *Query 1 is in #**P**.*

Proof. Query 1 can be answered by essentially simulating a single-membrane P system; indeed, if Π is monodirectional, no object can enter membrane h from the parent membrane, and if h is elementary, neither can objects from children membranes. By the Milano Theorem [11], a P system without division (thus, in particular, a single membrane without division), even with cooperative evolution rules, can be simulated in deterministic polynomial time. By allowing nondeterminism, the divisions $[a]_h \to [b]_h\ [c]_h$ of membrane h can be simulated in polynomial time by nondeterministically choosing whether to simulate the "left" (where a is rewritten as b) or the "right" membrane (where a is rewritten as c) resulting from the division. After simulating t steps, this results in a nondeterministic computation tree with a leaf for each instance of membrane h. Each computation must accept if and only if an object of type a is sent out at time t, which gives us a number of accepting computations identical to the number of objects a that are collectively sent out at time t by membranes labelled by h, proving that the query is in #**P**. □

Since we have shown that the query is actually computable by a function in #**P**, we are now ready to prove the main theorem of this section:

Theorem 2. *A family of (semi)uniform shallow chargeless monodirectional P systems with active membranes running in polynomial time can be efficiently simulated with a single #**P** query.*

Proof. Given an input string x, the corresponding P system Π_x can be constructed in polynomial time by a deterministic Turing machine M that simulates the (two) Turing machine(s) establishing the (semi)uniformity condition.

Before beginning the actual simulation of Π_x, we can ask a number of queries to an oracle for Query 1; in particular, we ask Query 1 for each possible value of h (labels of elementary membranes), of a (symbols of the alphabet of Π_x), and of t (time steps between 0 and $p(|x|)$, where p is the polynomial

running time of the family). Clearly, this is a polynomial number of parallel queries. These queries can then be combined into a single #**P** query by means of Proposition 1.

Then, the external membrane of Π_x can be simulated by Turing machine M. Since this membrane does not divide, by the Milano Theorem [11] it can be simulated deterministically in polynomial time, except for the objects coming from the children membranes, which are allowed to divide. But these have already been precomputed by asking the oracle queries, and can simply be added with the corresponding multiplicity in the correct time step to the configuration of the outermost membrane. The simulation can then be carried out correctly until the result object is sent out to the environment, and the simulation algorithm accepts or rejects correspondingly. □

5 Conclusions

We have shown that minimal cooperation for monodirectional, shallow P systems with active membranes without charges is sufficient to reach and characterize $\mathbf{P}_{\parallel}^{\#\mathbf{P}}$. This minimal amount of cooperation seems actually necessary and it might be expressed either explicitly (as done here), or implicitly (as with antimatter [5]). However, the minimal amount of cooperation actually required to "count", thus allowing the construction of a #**P** oracle, is still an open research avenue. In fact, while minimal cooperation can simulate annihilation rules in P systems with antimatter, it is unclear if there exist ways of performing cooperative actions that are even weaker, while still attaining the ability to "count" and perform #**P** queries.

References

1. Fortnow, L.: Counting complexity. In: Hemaspaandra, L.A., Selman, A.L. (eds.) Complexity Theory Retrospective II, pp. 81–107. Springer (1997)
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., Sosík, 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.: Monodirectional P systems. Natural Computing 15(4), 551–564 (2016)
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.: Subroutines in P systems and closure properties of their complexity classes. Theoretical Computer Science (2018), in press
7. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
8. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
9. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
10. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Polarizationless P systems with active membranes: Computational complexity aspects. Journal of Automata, Languages and Combinatorics 21(1–2), 107–123 (2016)
11. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)