Subroutines in P systems and closure properties of their complexity classes[⋆]

Alberto Leporati, Luca Manzoni, Giancarlo Mauri,
Antonio E. Porreca, Claudio Zandron

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

Summary. The literature on membrane computing describes several variants of P systems whose complexity classes $\boldsymbol{C}$ are “closed under exponentiation”, that is, they satisfy the inclusion $\mathbf{P}^{\boldsymbol{C}} \subseteq \boldsymbol{C}$, where $\mathbf{P}^{\boldsymbol{C}}$ is the class of problems solved by polynomial-time Turing machines with oracles for problems in $\boldsymbol{C}$. This closure automatically implies closure under many other operations, such as regular operations (union, concatenation, Kleene star), intersection, complement, and polynomial-time mappings, which are inherited from $\mathbf{P}$. Such results are typically proved by showing how elements of a family of P systems $\boldsymbol{\Pi}$ can be embedded into P systems simulating Turing machines, which exploit the elements of $\boldsymbol{\Pi}$ as subroutines. Here we focus on the latter construction, abstracting from the technical details which depend on the specific variant of P system, in order to describe a general strategy for proving closure under exponentiation.

1 Introduction

Complexity classes of the form $\mathbf{P}^{\boldsymbol{C}}$, characterised by polynomial-time Turing machines with oracles for languages in $\boldsymbol{C}$ [19], automatically inherit from $\mathbf{P}$ many closure properties. For instance, the determinism of Turing machines characterising $\mathbf{P}$ implies closure under complement, simply by switching the accepting and rejecting states of the machine. Under certain assumptions on $\boldsymbol{C}$, further closure properties are satisfied.

Let us say that a “reasonable” k-ary operation on languages is a function $f\colon \left(2^{\Sigma^\star}\right)^k \to 2^{\Sigma^\star}$ such that $f(L_1,\ldots,L_k) \in \mathbf{P}^{L_1,\ldots,L_k}$, that is, it can be computed efficiently with oracles for the k languages. A class $\boldsymbol{C}$ is closed under reasonable operations if $f(L_1,\ldots,L_k) \in \boldsymbol{C}$ for each reasonable operation f

⋆ This work was partially supported by Fondo d’Ateneo 2016 of Università degli Studi di Milano-Bicocca, project 2016-ATE-0492 “Sistemi a membrane: classi di complessità spaziale e temporale”.

and for each $L_1, \ldots, L_k \in \boldsymbol{C}$. The reasonable operations include all usual set theoretic ones (complement, union, intersection and all derived operations) and common language theoretic operations, such as the regular ones (concatenation, union, and Kleene star) [9].

For instance, a class $\boldsymbol{C}$ of the form $\mathbf{P}^{\boldsymbol{D}}$ is closed under reasonable operations whenever $\boldsymbol{D}$ is an *upward directed set*, that is, for each $L_1, L_2 \in \boldsymbol{D}$ there exists $L \in \boldsymbol{D}$ such that $L_1 \leq L$ and $L_2 \leq L$, where $\leq$ denotes polynomial-time reducibility; an oracle for L can thus answer queries for both L_1 and L_2 after a polynomial-time reduction. Any class $\boldsymbol{D}$ with a complete problem L is directed, since all languages in $\boldsymbol{D}$ can be reduced to L. This shows that classes such as $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{coNP}}$ and $\mathbf{P}^{\mathbf{PP}}$ (which coincides with $\mathbf{P}^{\#\mathbf{P}}$) are closed under reasonable operations. Furthermore, it clearly suffices to find a subset $\boldsymbol{E} \subseteq \boldsymbol{D}$ with $\mathbf{P}^{\boldsymbol{E}} = \mathbf{P}^{\boldsymbol{D}}$ and prove $\mathbf{P}^{\boldsymbol{E}}$ closed under reasonable operations to obtain the same result for $\mathbf{P}^{\boldsymbol{D}}$. This is the case for $\boldsymbol{D} = \mathbf{NP} \cup \mathbf{coNP}$ (and $\boldsymbol{E} = \mathbf{NP}$), a class that frequently appears in the membrane computing literature [8, 21, 23, 31, 30] and is not known to be itself directed. In fact this would imply $\mathbf{NP} = \mathbf{coNP}$, since there would be a language $L \in \mathbf{NP} \cup \mathbf{coNP}$ such that $L_1 \leq L$ for an $\mathbf{NP}$-complete language L_1 and $L_2 \leq L$ for a $\mathbf{coNP}$-complete language L_2. Other classes $\boldsymbol{C}$ trivially closed under reasonable operations are those satisfying $\mathbf{P}^{\boldsymbol{C}} = \boldsymbol{C}$, such as $\mathbf{PH}$, the polynomial hierarchy [29], and $\mathbf{CH}$, the counting hierarchy [33].

Several variants of P systems have been proved able to simulate polynomial-time Turing machines with oracles, exploiting a family $\boldsymbol{\Pi}$ of P systems deciding a language L as "subroutines", by embedding them into the membrane structure of larger P systems providing the input and processing the output of the elements of $\boldsymbol{\Pi}$ [24, 12, 13, 16, 14]. This implies the closure under exponentiation of the corresponding complexity classes, in symbols $\mathbf{P}^{\mathbf{PMC}_{\mathcal{D}}} \subseteq \mathbf{PMC}_{\mathcal{D}}$, for some kind of P system $\mathcal{D}$. In this paper we describe this subroutine construction in a manner as independent from the specific variant of P systems as possible. In particular, we consider the cases of cell-like P systems (Section 2), tissue P systems (Section 2.1), and introduce a new construction for monodirectional cell-like P systems (Section 3).

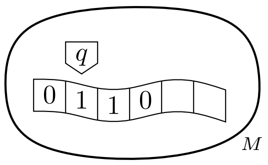
Many computational complexity results in membrane computing have the form $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{D}}$, that is, some class $\mathcal{D}$ of P systems (e.g., active membranes without charges and dissolution using minimal cooperative rules [30, Corollary 6.6]) can solve in polynomial time all $\mathbf{NP}$ and $\mathbf{coNP}$ problems. We argue that this is unlikely to ever be an exact characterisation, since the features that allow us to solve $\mathbf{NP}$-complete problems efficiently are the same that allow the subroutine construction, and this would imply $\mathbf{NP} = \mathbf{coNP}$.

2 Subroutines in cell-like P systems

Several Turing machine simulations by means of polynomial-time uniform families of P systems have been proposed in the literature; some of these apply to unrestricted Turing machines [26, 2], while others are limited to machines working in logarithmic space [25], polynomial time [7, 6], polynomial space [32, 24, 17, 10, 12, 13, 15, 16, 14], or exponential space [1]. Most of these solutions [32, 24, 1, 2, 7, 17, 10, 12, 13, 15, 6, 16, 14] are able to simulate Turing machines working in polynomial time with a polynomial slowdown.

The current configuration of the simulated Turing machine can be encoded in several equivalent ways by the simulating P system. A simple, common encoding [12, 13, 15, 16, 14] for a configuration of a polynomial-space Turing machine in state q, having the tape head in position i, and the tape containing the string $x = x_1 \cdots x_m$ is given by the multiset $q_i x_{1,1} \cdots x_{m,m}$, where the object q_i encodes state and head position, and the symbol x_j contained in tape cell j is encoded by the object $x_{j,j}$ of the multiset (i.e., it is subscripted with its position on the tape). The simplest mechanism to simulate a step of the Turing machine is using cooperative rewriting rules; suppose that $\delta(q, a) = (r, b, +1)$ describes the transition of the machine reading symbol a in state q to state r, symbol b and movement to the right. This can be trivially simulated by the cooperative rewriting rule $[q_i\ a_i \to r_{i+1}\ b_i]_M$, which is repeated for each cell position i up to the maximum length of the tape. Notice that *minimal cooperation*, with only two objects on the left-hand side of the rules, suffices for this purpose [31, 30]. All published solutions known to the authors [12, 13, 15, 16, 14] use alternative mechanisms (such as membrane charges, antimatter annihilation, antiport communication) to perform essentially the same operation.

Irrespective of the actual encoding of the configuration of the Turing machine and the mechanism employed to simulate a computation step, we will use the following symbol



to denote a membrane structure with root M inside which the simulation is carried over.

Let us now consider the case of a Turing machine M with an oracle for language $L \subseteq \Sigma^\star$. Suppose that M writes on its tape the query string $x \in \Sigma^\star$ and enters its query state. Suppose that language L is decided by a family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of recogniser P systems [22, 18]. By embedding the (empty) membrane structure of the P system Π_x inside the membrane structure of the P system simulating M (Fig. 1) and initialising the configuration of Π_x when the simulation reaches the query state, we can simulate the oracle and read the result of the query as the output of Π_x [24]. It is possible to send-in the initial multisets contained in Π_x, which typically requires some synchronisation using timers, so that all initial objects appear simultaneously (Fig. 2).

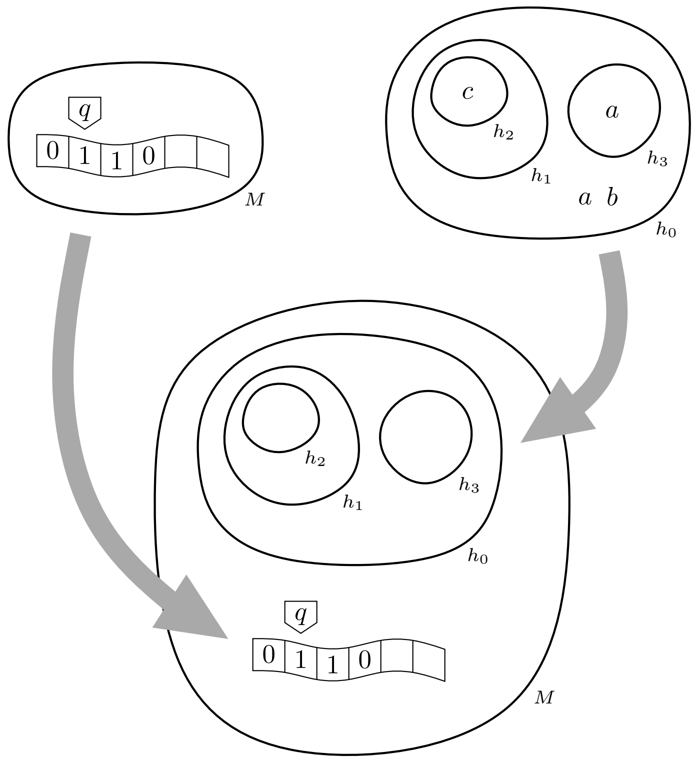


Fig. 1. The (empty) membrane structure of a P system is embedded into another P system simulating a Turing machine M.

Since the query string x is, in general, unknown before the beginning of the simulation, several P systems must be embedded in order to be able to process query strings of different length. Suppose that language L is decided by a uniform family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of P systems, where input strings of the same length n are associated to P systems sharing the same membrane structure and rules $\Pi(n)$, which only differ with respect to the initial multisets they contain. Then, multiple empty membrane structures $\Pi(0), \Pi(1), \ldots, \Pi(m)$ can be embedded, up to the maximum possible query string length (an upper bound is given, for instance, by the length of the tape of M), and the correct one is selected at runtime by the P system simulating M (Fig. 3).

When the simulated Turing machine performs multiple queries with query strings of the same length during its computation, it is possible to embed multiple copies of each P system $\Pi(n)$ [24]. Each of these copies can then be used for a single query (Fig. 4). A possible alternative is to reset the configuration of P system $\Pi(n)$ after the query simulation has been performed [12].

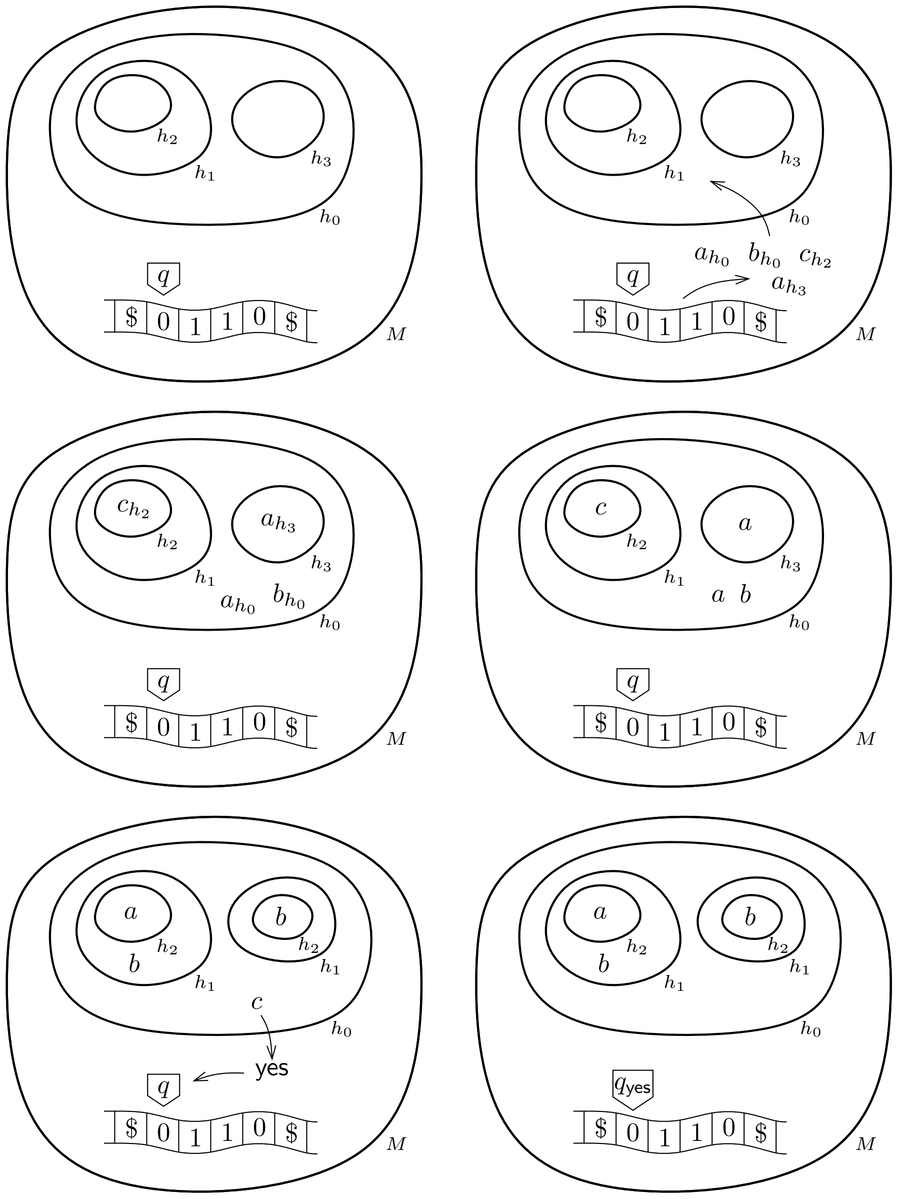


Fig. 2. The query string x on the tape of the simulated Turing machine is encoded as the initial configuration of the embedded P system Π_x. Each object is subscripted by the label of its target membrane, which it reaches by using send-in rules. When all objects have reached their destination, they simultaneously lose the subscripts, and the computation of Π_x begins. The output of Π_x is then read and incorporated into the state of the simulated Turing machine, as in the configuration following the query.

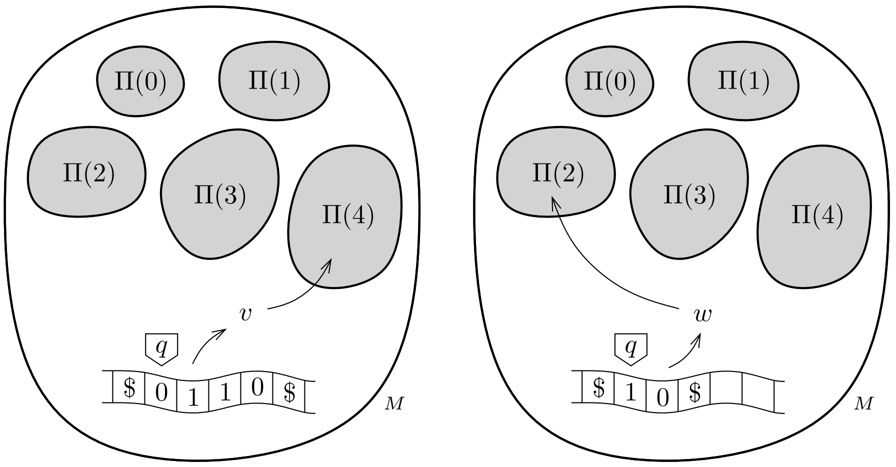


Fig. 3. Query strings of different lengths can be processed by distinct embedded P systems, selected when the oracle query is simulated.

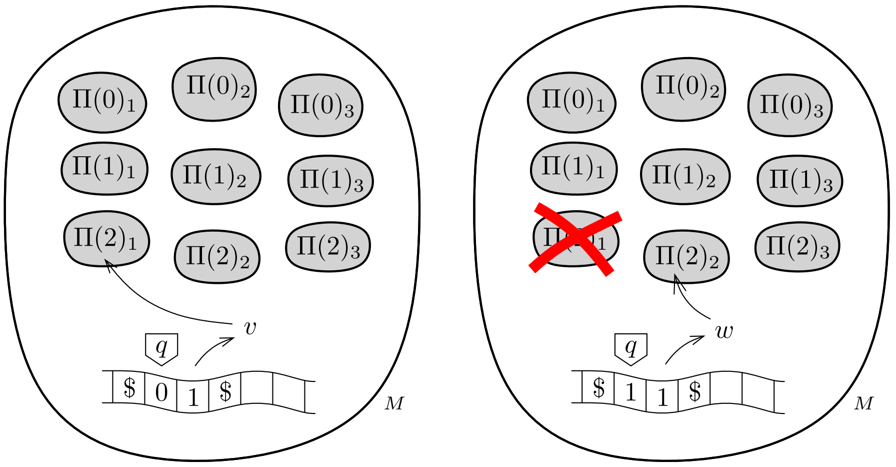


Fig. 4. Multiple query strings of the same length can be processed by replicating the P systems simulating the oracle and using each of them only once.

2.1 Subroutines in tissue P systems

The oracle query construction for cell-like P systems embeds elements of a family of P systems into a membrane where a Turing machine is simulated. This construction can be adapted to tissue P systems by having the Turing machine simulation take place in a cell, and placing the elements of the family of tissue P systems deciding the oracle language on the side [16]. The communication needed in order to simulate the oracle queries in this variant is not hierarchical, but between adjacent cells (Fig. 5).

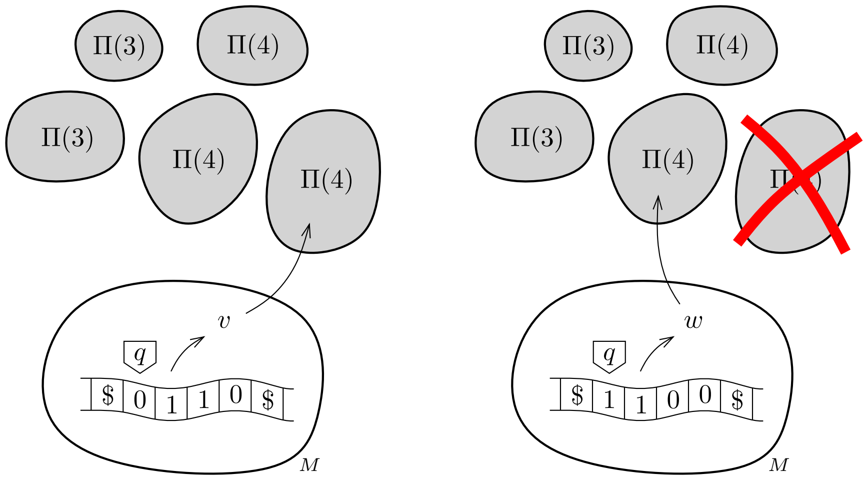


Fig. 5. Simulating oracle queries in tissue P systems.

3 Subroutines in monodirectional cell-like P systems

In monodirectional cell-like P systems [13, 14], where send-in rules are disallowed, the construction of Section 2 does not apply. However, we can turn the construction of Fig. 1 inside out, by having the Turing machine simulation embedded *into the input membrane* of the P system Π simulating the oracle (Fig. 6). Instead of sending in the encoding of the query string, it is the encoding of the configuration of the Turing machine that is sent *out*, through the whole membrane structure of Π, where it waits for the oracle query result (Fig. 7) [13, 14]. This is needed because the P system simulating the oracle can only send its result outwards and, since the system is monodirectional, the only way to intercept it is to move out the entire simulation of the Turing machine.

Unlike the bidirectional case, the objects of the initial configuration of Π cannot be arranged during the query simulation, as that requires send-in if the membrane structure of Π is not linear or if the input membrane is not elementary. A solution is to have them already in their correct position in the initial configuration of the combined P system but with a timer subscript, which is deleted at a predefined time step t, when a query may take place. If the simulated Turing machine does not perform a query at time t, it can be adapted so that it makes a “dummy” query at that time, ignoring its result.

As in the bidirectional case, the oracle strings are usually only available at runtime, and thus multiple P systems simulating oracles must be arranged in advance, in order to accommodate any possible query string. Given the restriction on the direction of communication, a natural solution is to *nest* these auxiliary P systems, placing each one inside the input membrane of the next one. When a query takes place, the Turing machine configuration

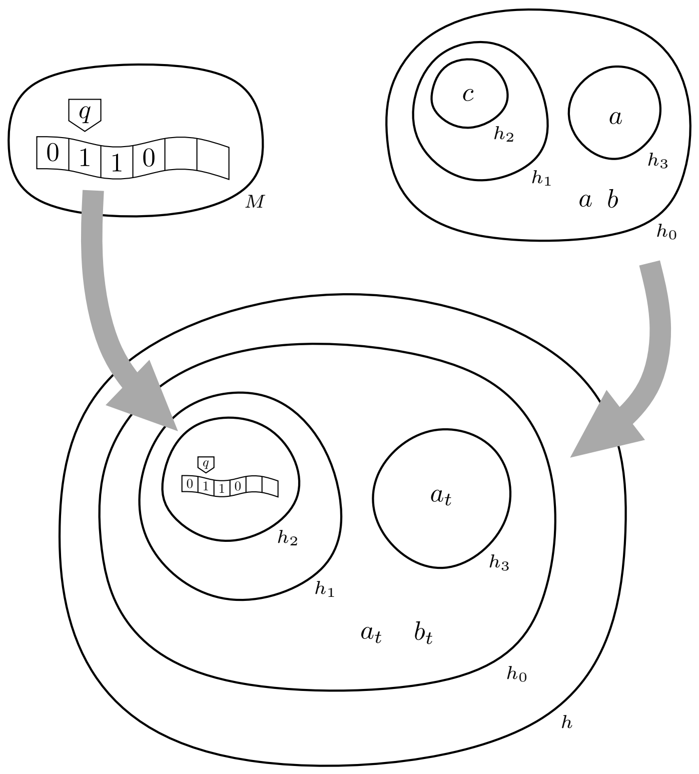


Fig. 6. In the monodirectional case, the P system simulating the Turing machine is embedded into the input membrane of the P system Π simulating the oracle; the other objects in the initial configuration of Π have an associated timer for synchronisation purposes.

is first moved to the first P system suitable for the query string, where the multiset encoding it is left; then, the Turing machine configuration is moved outside the whole set of auxiliary P systems, where it waits for the result of the query (Fig. 7).

If multiple queries are carried out, it suffices to repeatedly nest the array of auxiliary P systems, always using the input membranes as junction points, and repeating the query procedure as many times as necessary (Fig. 8) [13, 14].

4 Discussion and open problems

Many complexity theory results for P systems have the form $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{D}}$ for some class $\mathcal{D}$ of P systems, and by closure under complementation this implies $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{D}}$ [22]. Solving **NP**-complete problems usually requires some form of "context-sensitivity", such as cooperative evolution rules

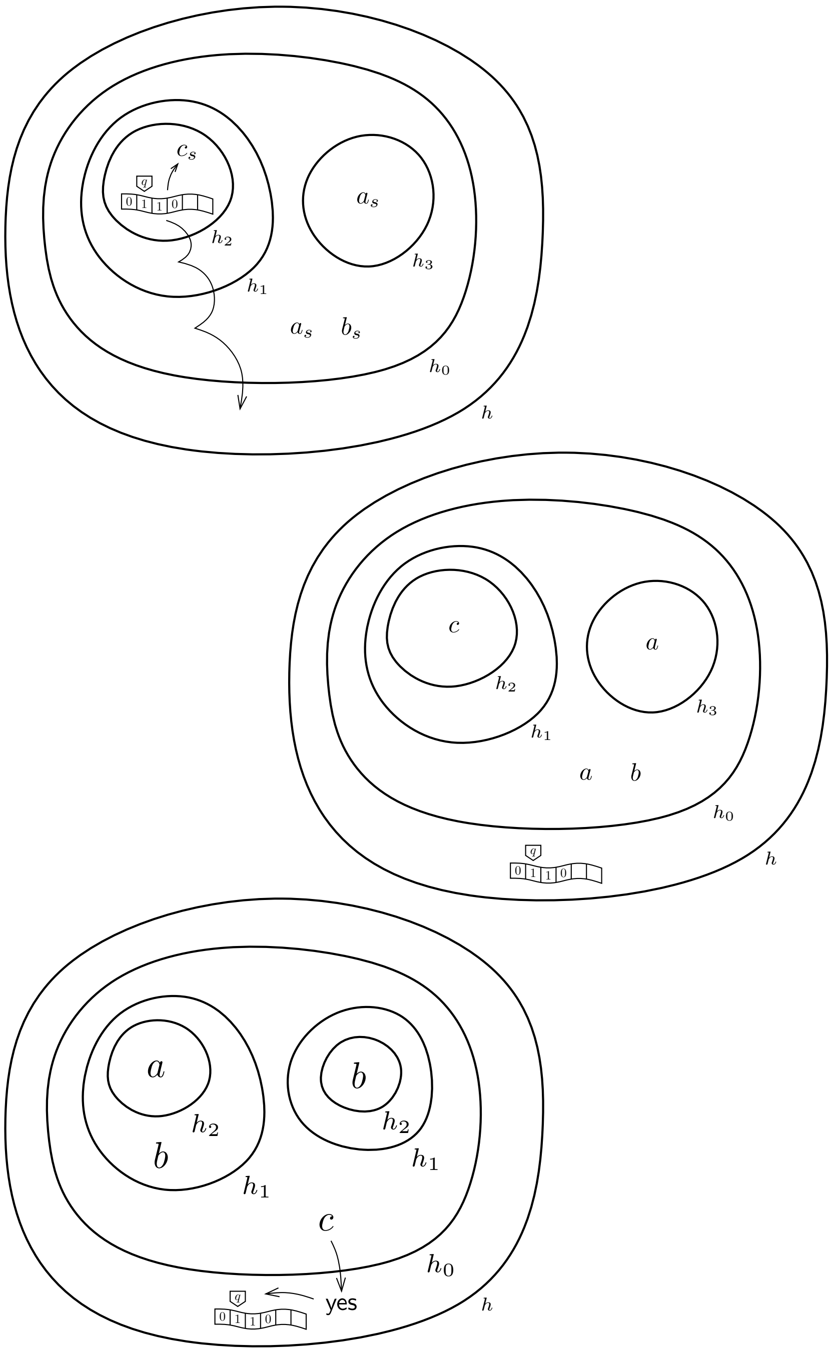


Fig. 7. Query simulation procedure for monodirectional cell-like P systems.

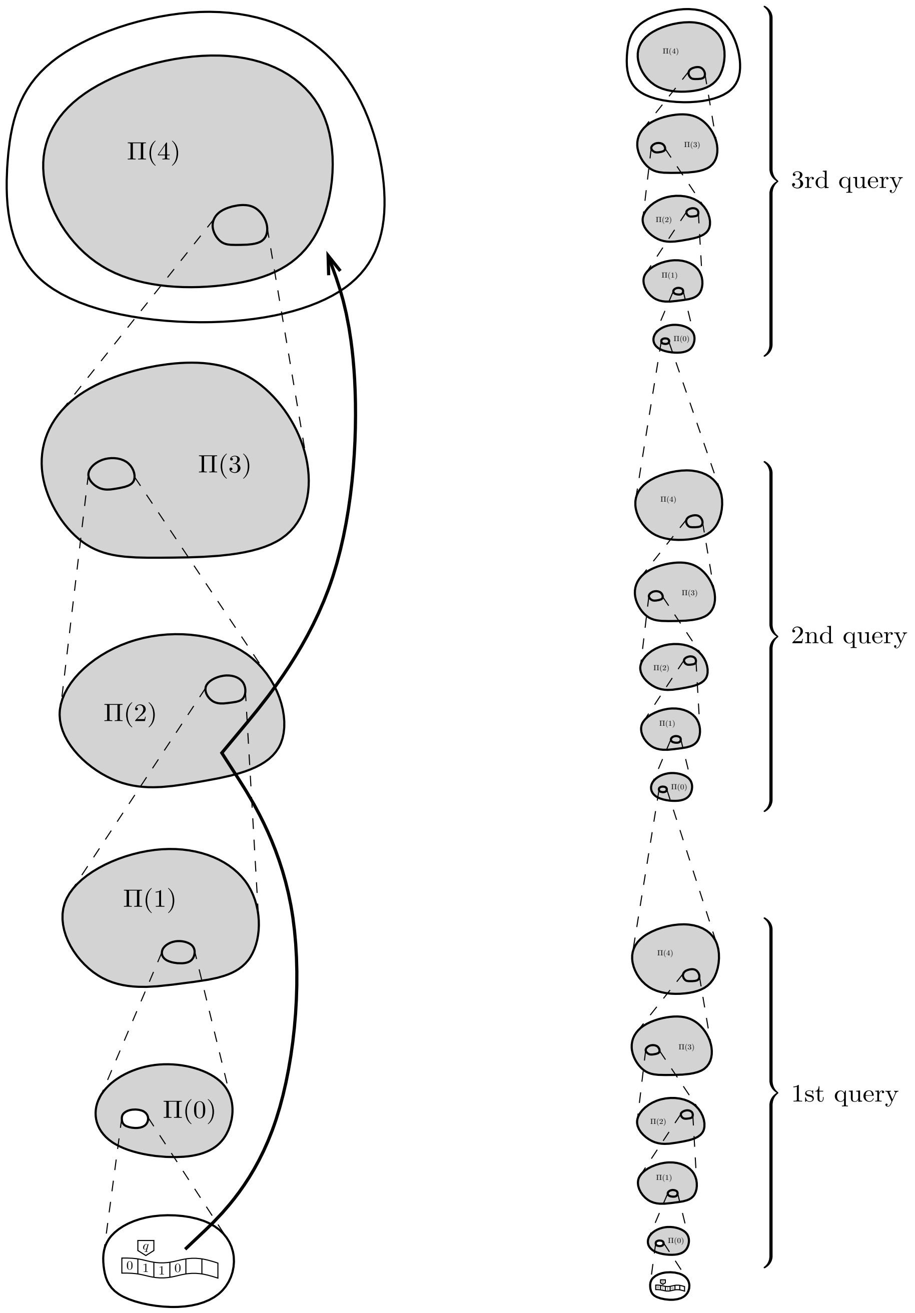


Fig. 8. Repeated nesting of monodirectional cell-like P systems to perform queries of a priori unknown length (on the left) and for multiple queries of a priori unknown length (on the right).

(even minimal cooperation), membranes charges, membrane dissolution, or antimatter annihilation [31, 30, 20, 4, 5]. However, these same features typically suffice to carry out the oracle simulation constructions described in Section 2 or 3, which imply that $\mathbf{PMC}_{\mathcal{D}}$ has the form $\mathbf{P}^{\boldsymbol{C}}$ for some class $\boldsymbol{C}$. This means that it is quite unlikely that $\mathbf{NP} \cup \mathbf{coNP}$ is an exact characterisation of $\mathbf{PMC}_{\mathcal{D}}$.

Indeed, if $\mathbf{NP} \cup \mathbf{coNP}$ were of the form $\mathbf{P}^{\boldsymbol{C}}$, then $\mathbf{P}^{\mathbf{NP} \cup \mathbf{coNP}} = \mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}}$. But $\mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}} = \mathbf{P}^{\boldsymbol{C}}$: if $L_1 \in \mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}}$, there exists a Turing machine M_1 with oracle for $L_2 \in \mathbf{P}^{\boldsymbol{C}}$ such that $L(M_1) = L_1$, and a Turing machine M_2 with an oracle for $L_3 \in \boldsymbol{C}$ such that $L_2 = L(M_2)$. Let M be a Turing machine with oracle for L_3. This machine simulates M_1 until it enters its query state, then it simulates M_2 until it enters its own query state; since M and M_2 have the same oracle, the queries of M_2 can be answered directly. Then $L(M) = L(M_1) = L_1$, and we can conclude that $\mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}} = \mathbf{P}^{\boldsymbol{C}}$. But then $\mathbf{P}^{\mathbf{NP} \cup \mathbf{coNP}} = \mathbf{P}^{\boldsymbol{C}} = \mathbf{NP} \cup \mathbf{coNP}$. Since $\mathbf{P}^{\mathbf{NP} \cup \mathbf{coNP}} = \mathbf{P}^{\mathbf{NP}}$, this class has complete problems, for instance the standard complete problem H of deciding if a Turing machine M with $\mathbf{NP}$ oracle accepts a string x within t steps (where t is given in unary notation). Then either $H \in \mathbf{NP}$ or $H \in \mathbf{coNP}$, and it is hard for both $\mathbf{NP}$ and $\mathbf{coNP}$; this means that either there exists a $\mathbf{NP}$-hard problem in $\mathbf{coNP}$, or a $\mathbf{coNP}$-hard problem in $\mathbf{NP}$: in both cases, this implies $\mathbf{NP} = \mathbf{coNP}$.

Furthermore, it is often the case [24, 12, 14, 16] that the amount of context-sensitivity that allows us to simulate Turing machines with oracles also suffices, at least in the bidirectional case, to simulate not only oracles for decision problems, but their counting version, which is usually more powerful. For instance, several variants of P systems without non-elementary membrane division rules characterise the complexity class $\mathbf{P}^{\mathbf{PP}}$ (which coincides with $\mathbf{P}^{\#\mathbf{P}}$) [19, 11, 14, 16].

Finally, notice that the above remarks apply even to variants of P systems powerful enough to characterise $\mathbf{PSPACE}$ in polynomial time [27, 3, 28, 4], even if the algorithms developed in order to prove these results do not usually employ an oracle simulation construction. Indeed, the class $\mathbf{PSPACE}$ is closed under exponentiation: $\mathbf{P}^{\mathbf{PSPACE}} = \mathbf{PSPACE}$.

The property investigated in this paper, closure under exponentiation, that is, satisfying $\mathbf{P}^{\boldsymbol{C}} = \boldsymbol{C}$, seems to apply to a wide range of variants of P systems. A natural follow-up question is whether the more common *closure under oracles* (or *under subroutines*), where $\boldsymbol{C} = \boldsymbol{C}^{\boldsymbol{C}}$, does also apply for some of these variants. This also requires establishing a notion of "P system with oracle"; a first definition has been already given in [12], albeit for purely technical reasons.

An interesting open problem, although a presumably challenging one from a technical standpoint, is to formally characterise the membrane computing features (such as combination of rules or properties of the membrane structures) that enable the oracle simulation construction.

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: The computational power of exponential-space P systems with active membranes. In: Martínez-del-Amor, M.A., Păun, Gh., Pérez-Hurtado, I., Romero-Campero, F.J. (eds.) Proceedings of the Tenth Brainstorming Week on Membrane Computing, vol. I, pp. 35–60. Fénix Editora (2012)
2. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. Theoretical Computer Science 529, 69–81 (2014)
3. 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)
4. Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution to QSAT using polarizationless active membranes. In: Durand-Lose, J., Margenstern, M. (eds.) Machines, Computations, and Universality, 5th International Conference, MCU 2007, Lecture Notes in Computer Science, vol. 4664, pp. 122–133. Springer (2007)
5. Díaz-Pernil, D., Alhazov, A., Freund, R., Gutiérrez-Naranjo, M.A., Leporati, A.: Recognizer P systems with antimatter. Romanian Journal of Information Science and Technology 18(3), 201–217 (2015)
6. Gazdag, Z., Kolonits, G.: Remarks on the computational power of some restricted variants of P systems with active membranes. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 209–232. Springer (2017)
7. Gazdag, Z., Kolonits, G., Gutiérrez-Naranjo, M.A.: Simulating Turing machines with polarizationless P systems with active membranes. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014. Lecture Notes in Computer Science, vol. 8961, pp. 229–240 (2014)
8. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. Theoretical Computer Science 371(1–2), 54–61 (2007)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd Edition. Addison-Wesley (2006)
10. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Constant-space P systems with active membranes. Fundamenta Informaticae 134(1–2), 111–128 (2014)
11. 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)
12. 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)
13. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing 15(4), 551–564 (2016)

14. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. Theoretical Computer Science (2017), in press
15. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Shallow non-confluent P systems. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 307–316 (2017)
16. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E.P., Zandron, C.: Characterising the complexity of tissue P systems with fission rules. Journal of Computer and System Sciences (2017), in press
17. Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes. Journal of Automata, Languages and Combinatorics 19(1–4), 173–184 (2014)
18. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
19. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
20. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
21. Pérez-Jiménez, M.J.: The P versus NP problem from the membrane computing view. European Review 22(01), 18–33 (2014)
22. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
23. Pérez-Jiménez, M.J., Sosík, P.: An optimal frontier of the efficiency of tissue P systems with cell separation. Fundamenta Informaticae 138(1–2), 45–60 (2015)
24. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems simulating oracle computations. In: Gheorghe, M., Păun, Gh., Salomaa, A., Rozenberg, G., Verlan, S. (eds.) Membrane Computing, 12th International Conference, CMC 2011, Lecture Notes in Computer Science, vol. 7184, pp. 346–358. Springer (2012)
25. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing, 13th International Conference, CMC 2012, Lecture Notes in Computer Science, vol. 7762, pp. 342–357. Springer (2013)
26. Romero-Jiménez, A., Pérez-Jiméz, M.J.: Simulating Turing machines by P systems with external output. Fundamenta Informaticae 49(1–3), 273–287 (2002)
27. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
28. 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)
29. Stockmeyer, L.J.: The polynomial-time hierarchy. Theoretical Computer Science 3(1), 1–22 (1976)
30. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. Fundamenta Informaticae 153(1–2), 147–172 (2017)
31. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Reaching efficiency through collaboration in

membrane systems: Dissolution, polarization and cooperation. Theoretical Computer Science (2017), in press

32. Valsecchi, A., Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 10th International Workshop, WMC 2009, Lecture Notes in Computer Science, vol. 6501, pp. 461–478. Springer (2010)
33. Wagner, K.W.: The complexity of combinatorial problems with succinct input representation. Acta Informatica 23(3), 325–356 (1986)