Eighteenth Brainstorming Week on Membrane Computing

Sevilla, February 4 – 7, 2020

David Orellana-Martín
Gheorghe Păun
Agustín Riscos-Núñez
Ignacio Pérez-Hurtado
Editors

Eighteenth Brainstorming Week on Membrane Computing

Sevilla, February 4 – 7, 2020

David Orellana-Martín
Gheorghe Păun
Agustín Riscos-Núñez
Ignacio Pérez-Hurtado
Editors

RGNC REPORT 1/2020

Research Group on Natural Computing
Universidad de Sevilla

Sevilla, 2020

ISBN: XXX-XX-XXXXX-X

Printed by: Artes Gráficas Moreno, S.L.
`http://www.agmoreno.net/`

Preface

The Eighteenth Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from February 4 to 7, 2020, hosted by the Research Group on Natural Computing (RGNC) from the Department of Computer Science and Artificial Intelligence of Universidad de Sevilla. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and all the next editions have been taking place in Sevilla since then, always at the beginning of February.

In the style of previous meetings in this series, was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation. Several "provocative" talks were delivered, mainly devoted to open problems, research topics, announcements, conjectures waiting for proofs, or ongoing research works in general (involving both theory and applications). Joint work sessions were scheduled on the afternoons to allow for collaboration among the about 25 participants – see the list in the end of this preface.

The papers included in this volume, arranged in the alphabetic order of the authors, were collected in the form available at a short time after the brainstorming; several of them are still under elaboration. The idea is that the proceedings are a working instrument, part of the interaction started during the stay of authors in Sevilla, meant to make possible a further cooperation, this time having a written support.

A selection of papers from this volume will be considered for publication in the new *Journal of Membrane Computing*, published by Springer-Verlag (`www.springer.com/41965`).

Other papers elaborated during the 2020 edition of BWMC will be submitted to other journals or to suitable conferences. The reader interested in the final version of these papers is advised to check the current bibliography of membrane computing available in the domain website `http://ppage.psystems.eu`.

***

The list of participants as well as their email addresses are given below, with the aim of facilitating the further communication and interaction:

1. Artiom Alhazov, Institute of Mathematics and Computer Science of Academy of Sciences of Moldova, Moldova
artiom@math.md
2. José A. Andreu-Guzmán, Universidad de Sevilla, Spain
josandguz@gmail.com
3. Daniel Cagigas-Muñiz, Universidad de Sevilla, Spain
dcagigas@us.es
4. Daniel Cascado-Caballero, Universidad de Sevilla, Spain
danicas@us.es
5. Rodica Ceterchi, University of Bucharest, Romania
rceterchi@gmail.com
6. Lúdek Cienciala, Silesian University in Opava, Czech Republic
ludek.cienciala@fpf.slu.cz
7. Lucie Ciencialová, Silesian University in Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz
8. Rudolf Freund, Technological University of Vienna, Austria
rudolf.freund@tuwien.ac.at
9. Carmen Graciani, Universidad de Sevilla, Spain
cgdiaz@us.es
10. Ricardo Graciani-Díaz, Universitat de Barcelona, Spain
graciani@fqa.ub.edu
11. Sergiu Ivanov, IBISC, Université Évry, Université Paris-Saclay, France
sergiu.ivanov@univ-evry.fr
12. Pramod Kumar Sethy, Eötvös Loránd University, Hungary
pksethy@inf.elte.hu
13. Albert J. Liñán-Maho, ERIC LifeWatch, Universidad de Sevilla, Spain
almaho@us.es
14. Miguel A. Martínez-del-Amor, Universidad de Sevilla, Spain
mdelamor@us.es
15. David Orellana-Martín, Universidad de Sevilla, Spain
dorellana@us.es
16. Gheorghe Păun, Romanian Academy, Romania
curteadelaarges@gmail.com
17. Ignacio Pérez-Hurtado, Universidad de Sevilla, Spain
perezh@us.es
18. Mario de J. Pérez-Jiménez, Universidad de Sevilla, Spain
marper@us.es
19. Agustín Riscos-Núñez, Universidad de Sevilla, Spain
ariscosn@us.es
20. José L. Rodríguez-Andreu, ERIC LifeWatch, Universidad de Sevilla, Spain
jlrodandd@us.es

21. Álvaro Romero-Jiménez, Universidad de Sevilla, Spain
romero.alvaro@us.es
22. Luis Valencia-Cabrera, Universidad de Sevilla, Spain
lvalencia@us.es
23. Daniel Valenta, University of Opava, Czech Republic
f180337@fpf.slu.cz
24. György Vaszil, University of Debrecen, Hungary
vaszil.gyorgy@inf.unideb.hu
25. Claudio Zandron, University of Milano-Bicocca, Italy
claudio.zandron@unimib.it

As mentioned above, the meeting was organized by the Research Group on Natural Computing from Universidad de Sevilla (`http://www.gcn.us.es`)– and all the members of this group were enthusiastically involved in this (not always easy) work.

The meeting was partially supported from various sources: (i) Research Project TIN2017-89842-P cofinanced by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union, (ii) VI Plan Propio, *Vicerrectorado de Investigación de la Universidad de Sevilla*, and (iii) Department of Computer Science and Artificial Intelligence from *Universidad de Sevilla*.

The Editors
(Sep 2020)

Contents

Some open problems

Artiom Alhazov

Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
E-mail: `artiom@math.md`

1 Some questions from information theory

The research topic I would like to propose is COUNTING. I do not mean #**P**, I mean enumerating membrane structures or configurations of a given "size", modulo isomorphism or not. After finishing this email I reformulated this question as: "how much information is stored in a configuration of a membrane system with symbol objects?"

Modulo isomorphism: as "size" it is enough to specify the number m of membranes and t of objects. In total: since alphabets, e.g., membrane alphabet (labels) and object alphabet (symbols) are not bounded, the number of configurations, even with a specified number of membranes and objects would be infinite unless we somehow bound all alphabets. One of alternative approaches: fix the alphabets, in particular —O—=n kinds of objects, the number m of membranes and maximal number k of objects of each kind in each region. Then the total number of objects is not fixed, but bounded by m*n*k.

Some preliminary results.

I One membrane, modulo isomorphism. $t = 0 \to 1$ (empty multiset), $t = 1 \to 1$, $t = 2 \to 2(aa, ab)$, $t = 3 \to (aaa, aab, abc)$, $t = 4 \to 5$, $t = 5 \to 7$, $t = 6 \to 11$. Can be specified by a recurrent two-argument function. Seems to correspond to number sequence `https://oeis.org/A000041`.

II Membrane structures without objects, without considering labels/polarizations, modulo isomorphism. $m = 1 \to 1$ (only skin), $m = 2 \to 1$ (two nested membranes), $m = 3 \to 2$ ($[\,[\ \ ]\,[\ \ ]\,]$ and $[\,[\,[\ \ ]\,]\,]$), $m = 4 \to 4$, $m = 5 \to 9$. Seems to correspond to number sequence `https://oeis.org/A000081`.
Note: with labels, already $m = 2$ gives 2 different configurations, $[\,[\ \ ]_2\,]_1$ and $[\,[\ \ ]_1\,]_1$.

III One membrane, one label, one polarization, no isomorphism. Fixing the alphabet size $|O| = n$, the number of different multisets of cardinality t is the number of t-combinations with repetitions of set O, equal to $C(n+t-1, t)$, where

$C(n,k) = n!/(k!*(n-k)!)$. `https://en.wikipedia.org/wiki/Combination\#Number_of_combinations_with_repetition`

IV m membranes, $|O| = n$, at most k objects of any kind in any membrane, no isomorphism. Then the number of different configurations would seem to be equal to the number of different membrane structures with m membranes, multiplied by $(mn)^{(k+1)}$. Correction: but even in this case it is not so simple, because $[\,[\,a\,]\,[\quad]\,]$ and $[\,[\quad]\,[\,a\,]\,]$ are the same configuration; the expression above only holds under the assumption that there are no indistinguishable membranes, e.g., all membranes have different labels.

Goal: to have a general formula for each typical set of parameters specifying "size", of all configurations of this "size".

If counting tree structures is difficult, start with tissue.

Why? to understand how much information is indeed stored in a configuration of a P system, because the general impression that, with m membranes and t objects, there are approximately exponentially many different configurations, is too inaccurate and in some settings incorrect.

2 Some questions from simulators

Let me try to put down some of the ideas discussed last year; hopefully it will be useful, if not programmed in the near-future, then at least as a small publication.

1. Simulator. It would be very useful for theory to have a proper tool computing **the set of all possible transitions** from a given configuration. (Yes, I remember you are more focused on applications like zebra muscles, and you are quite concerned that it does not scale well. However, enough theory is anyway done, and there can be multiple simulators for PLingua. A few times I have been so upset that I thought about programming something like that myself, but what I do is normally not compatible, not user- friendly and definitely not in Java) Basically, having fixed the current configuration C, for each rule r it is easy to compute the maximal number $max(r,C)$ of times it can be applied in parallel. By dividing, for each object a in $lhs(r)$, $|C|_a$ by $|lhs(r)|_a$ rounding down, and taking the minimum. Same works in a distributed way, assuming proper flattening, possibly on the fly. In the *worst* case, all possibilities are among the combinations, for each rule r, of applying it from 0 to $max(r,C)$ times. It only takes to verify that the multiset union of $lhs(r)$ times the number of applications of r, summed over all rules r, is contained in r. That would be asynchronous mode. For any other mode, compliance is also to be checked. Of course, for maxpar that would be non-applicability of ANY further rule to the idle objects. Of course, in particular cases the set of possible transitions could be computed more efficiently.
2. Semantics and membranes. The recent advances in PLingua, like defining user rule types, seem to be quite useful. Yet, the main value I see is being able to

specify rules other than the most usual ones in the model (and, in particular, being able to combine rules from different models), and a comfortable way to write them is secondary, though also nice. A thing which is often related to syntax is how to apply it, the most needed versions being "in max. parallelism" and "sequentially". In particular, rules (a) are normally treated as parallel, even though sequential version has been considered, while rules (b), (c), (d), (e), ... are normally considered sequential, even though without polarizations alternative semantics has been studied.

Imagining rules involving more than one membrane, we need to be more precise. I proposed to indicate for each user type of rules (how exactly is a secondary question) which membranes are resources and which membranes are context. Then, resources are $lhs(r)$ and contexts are like promoters. Clearly, under usual definitions a rule would be sequential with respect to resources and parallel with respect to contexts.

If that is too difficult, usually it is enough to have implicit semantics: any membrane written completely in $lhs(r)$ is a reactant, a membrane written in $rhs(r)$ is a product, and a membrane containing "$\rightarrow$" is a context. Then, $[a \rightarrow u]$ is a parallel rule, but $[a] \rightarrow [u]$ would be a sequential rule. Similarly, it automatically follows in $[[] \rightarrow [][]]$ that the external membrane is a context, while internal membranes are resources, so we already know what is parallel and what is sequential. However, if the user wants such a rule to be sequential also w.r.t. the outer membrane, then it can be written as $[[]] \rightarrow [[][]]$.

Unfortunately, this convention alone does not suffice for automatic deduction of parallelism for rules like (b_0), (c_0). Because the context is not outside. (Yes, they could be written as boundary rules, but this syntax is neither universal nor compatible with traditional syntax for active membranes). But of course something can always be invented, e.g., when specifying rule types, write "$[p$" vs "$[s$" (parallel vs sequential) or "$[r$" vs "$[c$" (resource vs context).

Moreover, I was told that there is some problem with templates without external membrane, except the standard types

3. Dynamic membranes. Clearly, without explicitly indicated semantics $[] \rightarrow [[]]$ would be a sequential membrane creation, while $[a \rightarrow [b]]$ would be a parallel membrane creation. Then, $[[a] \rightarrow b]$ is a membrane dissolution, where the external membrane is a context. But what is the behavior of other objects, those not specified in the rules explicitly? The main variant is of course, upon creation the new membrane will only contain b, and upon dissolution all the contents of the old membrane is released in the outer membrane. But of course there are other rules, although less studied. Last year I suggested to use some wildcard, or mask, e.g., \$1, to represent other objects (similarly, something like #1 can represent other membranes, and for technical reasons different characters may be chosen; I use these ones to explain the idea how to describe semantics different from the main one).

$[a\$1 \rightarrow \$1[b]]$ usual parallel membrane creation

$a\$1 \rightarrow \$1[b]$ same without the outer context

$[a\$1] \to [b[\$1]]$ create a new membrane around the existing one and send b there
$[[a\$1] \to b\$1]$ usual membrane dissolution
$[[a\$1] \to b]$ lose contents of the dissolved membrane
$[a\$1] \to [b\$1][c\$1]$ usual membrane division
$[a\$1] \to [b\$1][c]$ create a sibling membrane, without replicating contents
$[a\$1] \to [\$1(O)][\$1(O')]$ membrane separation
$[a\$1[\$2]] \to [\$2[a\$1]]$ exchange objects in two membranes if the first one contains a
Then, there may be different kinds of non-elementary membrane division $[a] \to [b][c]$ same syntax as for elementary membranes, replicate objects and membranes. Can be written as $[a\$1\#1] \to [b\$1\#1][c\$1\#1]$
$[[][]] \to [[]][[]]$ separating submembranes. But what exactly happens to other submembranes if there are more than these two?
$[\$1\#1[][]] \to [\$1\#1[]][\$1\#1[]]$ replicating other objects and membranes
$[\$1\#1[][]] \to [\$1(O)\#1[]][\$1(O')\#1[]]$ separating objects and replicating membranes
$[\$1\#1[][]] \to [\$1\#1(H)[]][\$1\#1(H')[]]$ replicating objects and separating membranes
$[\$1\#1[][]] \to [\$1(O)\#1(H)[]][\$1(O')\#1(H')[]]$ separating objects and membranes
Overall, I think there may be some reasonable consistent universal way how to describe the precise evolution not only of dedicated objects and membranes, but also related objects and membranes, because mass action is needed (the first classical example of the mass action is dissolution, of course currently programmed explicitly).

4. Tests. From time to time, researchers consider rules that were not considered in the original model. I believe many (though not all) of these issues can be captured by the thoughts above.
   a sequential (a), parallel (b), (c), (d), (e), ...
   b where other objects and membranes go - division vs separation, outside vs delete, ...
   c external rules: $a[] \to u[]$, $a[] \to b$, $a[] \to b[][]$, ...
   d ...
5. Other models besides active membranes. With suitable choice of parallel/sequential semantics made clear, $r \in R_i$ can be written as $[r]_i$. In most cases, membrane i must be treated as context, hence, rules are parallel with respect to it.
   Antiport: $u[v] \to v[u]$
   Evolutional antiport or boundary rules: $u[v] \to u'[v']$
   Transitional: pretty standard, except multiple targets would be represented as multi-membrane context, and dissolution semantics is normally assumed parallel (multiple δ = one dissolution), not sequential.
   Spiking: mostly similar, the main difference are additional regular expressions.

Promoters, inhibitors - how much is already captured by PLingua??
Priorities - is there already a well-established syntax for them?
Notice that strong and week priorities can co-exist: these are just additional filters for the set of the next configurations (see part 1: Simulator) besides the derivation mode. As discussed with Rudi a few days after BWMC19, filters like priorities should be applied BEFORE the derivation mode filter.

6. Other derivation modes. A new (mostly studied in the last few years) important derivation mode for many models is **set maximally-parallel**. Same as maximally parallel, but in each step each rule may be only applied once. Technically similar to having a dedicated catalyst for each rule.
   Some of the classical modes that would be most important to also have are sequential and asynchronous. Asyn is even easier than maxpar - just remove the maximality filter. Sequential is of course the easiest to implement.
7. New ways of rule control. Activation and blocking (I hope to soon finish formalizing the concept also for zero-delay).
8. One of the "worst" things that could happen. "Denying". This is how we call the situation where there exists at least one applicable rule, but there is no valid multiset of rules. An example is "> 1 mode" in the situation where only one rule is applicable. This situation has been carefully avoided in the first years of membrane computing, but it does not present a problem (except it is unusual), e.g., this is similar to what happens to partially blind register machines when they try to decrement a register containing zero, which is not allowed by the model.
   Finally, a question is - can all of this co-exist in the same context? I still think it could. If anyone has an example of ANY membrane features that seem incompatible, please let me know, and *maybe* I will be able to convince you that there is no problem. Reminder - a universal look at P systems models: network of cells, see a few publications on the Formal Framework for a) static structures, b) dynamic structures, c) spiking.
   R. Freund, S. Verlan: **A Formal Framework for Static (Tissue) P Systems**. In: Eleftherakis G., Kefalas P., Păun Gh., Rozenberg G., Salomaa A. (eds) Membrane Computing. WMC 2007. Lecture Notes in Computer Science 4860. Springer, Berlin, Heidelberg, 2007, 271-284. `https://link.springer.com/chapter/10.1007%2F978-3-540-77312-2_17`
   R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez, S. Verlan: **A Formalization of Membrane Systems with Dynamically Evolving Structures**. International Journal of Computer Mathematics 90(4), 801–815 (2013) `https://doi.org/10.1080/00207160.2012.748899`
   S. Verlan, A. Alhazov, R. Freund, S. Ivanov: **A Formal Framework for Spiking Neural P Systems**. In Proceedings of the 20th International Conference on Membrane Computing, CMC20, Curtea de Argeș (Păun, Gh., Ed.). Bibliostar, Râmnicu Vâlcea, 2019, pp. 523–535. `http://membranecomputing.net/cmc20/pdf/procCMC20.pdf#page=250`

3 Some questions from variety

1. **Anti-membranes.**
   Reminder: rules of types $[]_h \to []_j []_k$, $[]_h []_{h'} \to \lambda$; could also be with objects.
   A. Alhazov, R. Freund, S. Ivanov: **(Tissue) P Systems with Anti-Membranes**. In Seventeenth Brainstorming Week on Membrane Computing (Orellana-Martín, D.; Păun, Gh.; Riscos-Núñez, A.; Andreu-Guzmán, J. A., Eds.), Sevilla. RGNC report 1/2019, University of Seville, Artes Gráficas Moreno, S.L., 2019, 29–30. `http://www.gcn.us.es/files/17bwmc/029_AntiMembranes.pdf`
   and
   A. Alhazov, R. Freund, S. Ivanov: **P Systems with Anti-Membranes**. In Proceedings of the 20th International Conference on Membrane Computing, CMC20, Curtea de Argeș (Păun, Gh., Ed.). Bibliostar, Râmnicu Vâlcea, 2019, 249–256. `http://membranecomputing.net/cmc20/pdf/procCMC20.pdf#page=250`
   1) Can we still do anything non-trivial if changing membrane labels is forbidden?
   2) Is it possible, e.g., to simulate boolean circuits?
   3) What if we forbid changing labels but allow a limited (3?) number of polarizations? let's say annihilation needs *some* form of polarization agreement
   4) Descriptional complexity of a small universal NFPAMS
   5) Which ingredients are needed to solve `SAT` with anti-membranes?
   6) How we can exploit deeper membrane structures? For instance, annihilation of nested membranes outside-in performs an ordered sequence of membrane dissolutions.
   7) antiMembranes for efficiency? In any way that is not a trivial translation of the previous research from objects to membranes.
2. **Channels.**
   For symport/antiport P systems, in tissue case, it is *usually* assumed that channels do not admit any parallelism. There has been a few exceptions. 1) Some Rudi's talk with PPT slides many years ago, where cells were represented by huge colored circles, I do not remember the title. 2)
   A. Alhazov, R. Freund, M. Oswald: **Tissue P Systems with Antiport Rules and Small Numbers of Symbols and Cells**. In: De Felice C., Restivo A. (eds) Developments in Language Theory. DLT 2005. Lecture Notes in Computer Science 3572. Springer, Berlin, Heidelberg, 2005, 100-111. `https://doi.org/10.1007/11505877_9`
   , where in $Ot'P$, primed letter t indicated that it was allowed to have distinct channels (i, j) and (j, i). 3) A more recent paper
   H. Adorna, A. Alhazov, L. Pan, B. Song: **Simulating Evolutional Symport/Antiport by Evolution-Communication and vice versa in Tissue P Systems with Parallel Communication**. In: Gheorghe M., Rozenberg

G., Salomaa A., Zandron C. (eds) Membrane Computing. CMC 2017. Lecture Notes in Computer Science 10725. Springer, Cham, 2018, 1-14. `https://doi.org/10.1007/978-3-319-73359-3_1`
relating evolutional symport/antiport with evolution-communication – in order to make it possible having direct simulation with a slowdown by a factor of a constant, communication needed to be massively parallel. 4) Older research on neural P systems, probably by [Krishna,Rama], long time before spiking. . . anyway, that last one was quite a different model. - Parallel VS sequential channels in tP systems. Improve results with mcre from $\mathbf{NP}\cup\mathbf{co}-\mathbf{NP}$ to **PSPACE**.

3. **Global rules.**
considered by A. Păun and once briefly by myself. This relates to problem (Q6) in Gheorghe's open problem list `http://www.gcn.us.es/?q=18bwmc_openproblems`. If membrane structure is static and we do not care about descriptional complexity, making all rules global does not seem to restrict us at all: objects can always be renamed when moved, so they know where they are. However, the total number of rules in this reduction may increase, and this technique becomes more complicated, or even impossible, with dissolution. BTW, this may open an interesting discussion at solving hard problems in polytime. Besides, not all membranes are ~~created~~ equal: by definition, elementary membrane division is not applicable to membranes that are (currently) non-elementary, and the skin cannot be dissolved or divided (and sometimes it is forbidden for any object to enter it) - this trick *might* help distinguishing membranes when needed, however, requiring non-determinism or complicated simulation. On the other hand, with sufficient ingredients one working region is already enough, so we should stay in a restricted enough settings.
A. Păun: **On P Systems with Global Rules**. In: Jonoska N., Seeman N.C. (eds) DNA Computing. DNA 2001. Lecture Notes in Computer Science, vol 2340. Springer, Berlin, Heidelberg, 2002, 329-339. `https://doi.org/10.1007/3-540-48017-X_31`
A. Alhazov, R. Freund: **On the Efficiency of P Systems with Active Membranes and Two Polarizations**. In: Mauri G., Păun Gh., Pérez-Jiménez M.J., Rozenberg G., Salomaa A. (eds) Membrane Computing. WMC 2004. Lecture Notes in Computer Science, vol 3365. Springer, Berlin, Heidelberg, 2005, `https://doi.org/10.1007/978-3-540-31837-8_8`
A. Alhazov, R. Freund, S. Ivanov: **Length P Systems**. Fundamenta Informaticae 134(1-2), 2014, 17-37. `https://doi.org/10.3233/FI-2014-1088`
4. **Maximal consistency modes.**
Reminder: here applicability does not only depend on lhs. I have heard about a practical use of this mode in a BWMC2019 discussion from Agustin (though I forgot which application it was for, so I would not know what reference to cite). Usually in membrane computing rule applicability only depends on the left side of the rule (whether all reactants are present in the current configuration,

and, possibly, whether some additional conditions are satisfied, e.g., promoters, inhibitors, etc.).
Consider rules changing membrane polarization. Allow to apply multiple rules (maximal parallelism), as long as the polarization in their rhs is the same. Let me call it "polarization agreement". Need to be precise, probably need to choose the polarization corresponding to at least one applied rule, if possible.
Other examples of maximal consistency:
   - Parallel string rewriting without conflicts [D. Besozzi], many years ago, reference needed.
   - Rudi's target agreement/label agreement, original reference needed.
   - Any other shared resource to agree upon?

   Overall, I believe this feature deserves more attention.

5. **cP systems.**
= P systems with **complex objects**, see [Nicolescu]. Reminder: prolog-like rules using power of term rewriting and unification. Very powerful model, e.g., a solution of the **Travelling Salesman Problem** has been reported with **five** rules only [CooperNicolescu_ACMC2017]. Some longer time ago the colleagues in my institute wanted to attack with P systems the problem of finding Gröbner basis. Unfortunately, the data structures that can be represented and efficiently processed by usual P systems are limited, and hence they are not suited well to work, e.g., with **dynamic ordered lists of strings** (a solution via Turing machine is not elegant). It turns out that cP systems are much more flexible in representing and efficiently processing complicated data structures.
Some problems that have been addressed besides universality/computational completeness and NP-hard problems, by usual P systems:
   - sorting `https://doi.org/10.1007/3-540-29937-8_8`,
   - dictionary search and update `http://univagora.ro/jour/index.php/ijccc/issue/download/44/pdf_165`,
   - inflections `http://www.math.md/publications/csjm/issues/v17-n2/10082/`,
   - annotating affixes `https://doi.org/10.1007/978-3-642-54239-8_7`,
   - firing squad synchronization problem `https://doi.org/10.1007/978-3-540-95885-7_9`

   (more problems and solutions can be found in Applications of Membrane Computing, 2005 and Membrane Computing Handbook). Need: more problems that are practical, well defined and sufficiently simple (simpler than Gröbner basis), to be attacked by cP systems, but not completely trivial (needing, say, more than two rules).
   Need: more problems that are practical, well defined and sufficiently simple (simpler than Gröbner basis), to be attacked by cP systems, but not completely trivial (needing, say, more than two rules).

Alternative Space Definitions for P Systems with Active Membranes

Artiom Alhazov[1], Alberto Leporati[2], Luca Manzoni[3], Giancarlo Mauri[2], and Claudio Zandron[2]

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2] Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo)
Università degli Studi di Milano-Bicocca
Viale Sarca 336, 20126 Milan, Italy.
{alberto.leporati,giancarlo.mauri,claudio.zandron}@unimib.it

[3] Dipartimento di Matematica e Geoscienze
Università degli Studi di Trieste
lmanzoni@units.it

Summary. The first definition of space complexity for P systems was based on an hypothetical real implementation by means of biochemical materials, and thus it assumes that every single object or membrane requires some constant physical space. This is equivalent to using a unary encoding to represent multiplicities for each object and membrane.

A different approach can also be considered, having in mind an implementation of P systems in silico; in this case, the multiplicity of each object in each membrane can be stored using binary numbers, thus reducing the amount of needed space. In this paper, we give a formal definition for this alternative space complexity measure, we define the corresponding complexity classes and we compare such classes both with standard space complexity classes and with complexity classes defined in the framework of P systems considering the original definition of space.

Key words: Membrane Systems, Computational Complexity, Space Complexity

1 Introduction

P systems with active membranes have been introduced in [6], considering the idea of generating new membranes through division of existing ones. The exponential amount of resources that can be obtained in this way, in a polynomial number of computation steps, naturally leads to the definition of new complexity classes to be compared with the standard ones.

Initially, the research activity focused on the investigation of time complexity, for the various classes of P systems that can be obtained by introducing different features.

The first definition of space complexity for P systems has been introduced in [8], and it was based on an hypothetical real implementation by means of biochemical materials such as cellular membranes and chemical molecules. Under this assumption, it was assumed that every single object or membrane requires some constant physical space, and this is equivalent to using a unary encoding to represent multiplicities.

A different approach can also be considered, focusing the definition on the simulative point of view. By considering an implementation of P systems in silico, it is not strictly necessary to store information concerning every single object: the multiplicity of each object in each membrane can be stored using binary numbers, thus reducing the amount of needed space.

In this paper, we give a formal definition for this alternative space complexity measure, we define the corresponding complexity classes and we compare such classes both with standard space complexity classes and with complexity classes defined in the framework of P systems considering the original definition of space [8]. In particular, we will give partial results concerning the use of constant, polynomial or exponential amount of space, respectively.

The paper is organized as follows. In Section 2 we recall some definitions concerning P systems with active membranes and space requirements in P systems computations. In Section 3, we introduce a different definition for measuring space (which we call *binary space* to underline that information concerning objects is stored in binary) and we give some results following immediately from this definition. In Section 4 we compare the new binary space complexity classes with standard complexity classes and with space complexity classes for P systems based on the standard definition of space. Finally section 5 draws some conclusions and presents some future research topics on this subject.

2 Basic definitions

In this section, we shortly recall some definitions that will be useful while reading the rest of the paper. For a complete introduction to P systems, we refer the reader to *The Oxford Handbook of Membrane Computing* [7].

Definition 1. *A* P system with active membranes *having initial degree* $d \geq 1$ *is a tuple* $\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$*, where:*

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*; in the following, we assume* $\Gamma = \{O_1, O_2, \ldots, O_n\}$
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes, labelled by elements of* Λ *in a one-*

to-one way, defining regions (the space between a membrane and all membranes immediately inside it, if any);

- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are strings over* Γ *describing the initial multisets of objects placed in the* d *regions of* μ*;*
- R *is a finite set of rules over* Γ*.*

Membranes are polarized, that is, they have an attribute called *electrical charge*, which can be neutral (0), positive (+) or negative (−).

A P system can made a computation step by applying its rules to modify the membrane structure and/or the membrane content.The following types of rules can be used during the computation:

- *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 at least 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).
- *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 region contains at least an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β.
- *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 at least 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 is changed to β.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labelled by h, having charge α and containing at least an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b.
- *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 at least an occurrence of the object a but having no other membrane inside (in this case the membrane is said to be *elementary*); the membrane is divided into two membranes having both label h and charges β and γ, respectively; the object a is replaced, respectively, by b and c in the two new membranes, while the other objects in the initial multiset are copied to both membranes.
- *(Weak) Non-elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$
  These rules operate just like division for elementary membranes, but they can be applied to non–elementary membranes, containing membrane substructures and having a label h. Like the objects, the substructures inside the dividing membrane are replicated in the two new copies of it.

A configuration of a P system with active membranes is described by the current membrane structure (including the electrical charge of each membrane) and the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane several evolution rules can be applied simultaneously).
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same principle applies to each membrane that can be involved in communication, dissolution, or division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached as the result of a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; in this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence of configurations $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ 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 *recognizers* by employing two distinguished objects *yes* and *no*; exactly one of these must be sent out from the outermost membrane, and only in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting.

In order to solve decision problems (i.e., decide languages over an alphabet Σ), we use *families* of recognizer P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language

$L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [4].

These families of *recognizer* P systems can be used to solve decision problems as follows.

Definition 2. *Let Π be a P system whose alphabet contains two distinct objects yes and no, such that every computation of Π is halting and during each computation exactly one of the objects yes, no is sent out from the skin to signal acceptance or rejection. If all the computations of Π agree on the result, then Π is said to be* confluent*; if this is not necessarily the case, then it is said to be* non-confluent *and the global result is acceptance if and only if there exists an accepting computation.*

Definition 3. *Let $L \subseteq \Sigma^\star$ be a language, $\mathcal{D}$ a class of P systems (i.e. a set of P systems using a specific subset of features) and let $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ be a family of P systems, either confluent or non-confluent. We say that $\mathbf{\Pi}$* decides *L when, for each $x \in \Sigma^\star$, $x \in L$ if and only if Π_x accepts.*

Complexity classes for P systems are defined by imposing a uniformity condition on $\mathbf{\Pi}$ and restricting the amount of time or space available for deciding a language.

Definition 4. *Consider a language $L \subseteq \Sigma^\star$, a class of recognizer P systems $\mathcal{D}$, and let $f \colon \mathbb{N} \to \mathbb{N}$ be a proper complexity function (i.e. a "reasonable" one, see [5, Definition 7.1]). We say that L belongs to the complexity class $\mathbf{MC}^*_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L such that:*

- $\mathbf{\Pi}$ *is* semi-uniform*, i.e. there exists a deterministic Turing machine which, for each input $x \in \Sigma^\star$, constructs the P system Π_x in polynomial time with respect to $|x|$;*
- $\mathbf{\Pi}$ *operates in time f, i.e. for each $x \in \Sigma^\star$, every computation of Π_x halts within $f(|x|)$ steps.*

In particular, a language $L \subseteq \Sigma^\star$ belongs to the complexity class $\mathbf{PMC}^*_{\mathcal{D}}$ if and only if there exists a semi-uniform family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L in polynomial time.

The analogous complexity classes for non-*confluent P systems are denoted by $\mathbf{NMC}^*_{\mathcal{D}}(f)$ and $\mathbf{NPMC}^*_{\mathcal{D}}$.*

Another set of complexity classes is defined in terms of *uniform* families of recognizer P systems:

Definition 5. *Consider a language $L \subseteq \Sigma^\star$, a class of recognizer P systems $\mathcal{D}$, and let $f \colon \mathbb{N} \to \mathbb{N}$ be a proper complexity function. We say that L belongs to the complexity class $\mathbf{MC}_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L such that:*

- $\mathbf{\Pi}$ *is* uniform, *i.e. for each $x \in \Sigma^\star$ deciding whether $x \in L$ is performed as follows: first, a polynomial-time deterministic Turing machine, given the length $n = |x|$ as a unary integer, constructs a P system Π_n with a distinguished input membrane; then, another polynomial-time deterministic Turing machine computes an encoding of the string x as a multiset w_x, which is finally added to the input membrane of Π_n, thus obtaining a P system Π_x that accepts if and only if $x \in L$.*
- $\mathbf{\Pi}$ *operates in time f, i.e. for each $x \in \Sigma^\star$, every computation of Π_x halts within $f(|x|)$ steps.*

In particular, a language $L \subseteq \Sigma^\star$ belongs to the complexity class $\mathbf{PMC}_{\mathcal{D}}$ *if and only if there exists a uniform family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L in polynomial time.*

The analogous complexity classes for non-*confluent P systems are denoted by* $\mathbf{NMC}_{\mathcal{D}}(f)$ *and* $\mathbf{NPMC}_{\mathcal{D}}$.

As stated in the Introduction, the first definition of space complexity for P systems introduced in [8] considered a possible real implementation with biochemical materials, thus assuming that every single object and membrane requires some constant physical space. Such a definition (in the improved version from [3], taking into account also the space required by the labels for membranes and the alphabet of symbols) is the following:

Definition 6. *Considering a configuration $\mathcal{C}$ of a P system Π, its* size $|\mathcal{C}|$ *is the number of membranes in the current membrane structure multiplied by* $\log|\Lambda|$, *plus the total number of objects from Γ they contain multiplied by* $\log|\Gamma|$. *If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a computation of Π, then the* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

$$|\boldsymbol{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}.$$

The space required by Π *itself is then obtained by computing the space required by all computations of Π and taking the supremum:*

$$|\Pi| = \sup\{|\boldsymbol{\mathcal{C}}| : \boldsymbol{\mathcal{C}} \text{ is a computation of } \Pi\}.$$

Finally, let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems, and let $s\colon \mathbb{N} \to \mathbb{N}$. We say that $\mathbf{\Pi}$ operates within space bound s *if and only if $|\Pi_x| \leq s(|x|)$ for each $x \in \Sigma^\star$.*

Analogously to what has been done for time complexity classes, we can define space complexity classes. By $\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$ (resp. $\mathbf{MCSPACE}^*_{\mathcal{D}}(f(n))$) we denote the class of languages which can be decided by uniform (resp. semi-uniform) families of confluent P systems of type $\mathcal{D}$ (for example, when we refer to P systems with active membranes, we denote this by setting $\mathcal{D} = \mathcal{AM}$), where each $\Pi_x \in \mathbf{\Pi}$ operates within space bound $f(|x|)$.

In particular, the class of problems solvable in polynomial space by uniform (resp. semi-uniform) confluent systems is denoted by $\mathbf{PMCSPACE}_{\mathcal{D}}$ (resp.

PMCSPACE$^*_{\mathcal{D}}$), and the class of problems solvable in exponential space by uniform (resp. semi-uniform) confluent systems is denoted by **EXPMCSPACE**$_{\mathcal{D}}$ (resp. **EXPMCSPACE**$^*_{\mathcal{D}}$).

The corresponding classes for non-confluent systems are **NPMCSPACE**$_{\mathcal{D}}$ (resp. **NPMCSPACE**$^*_{\mathcal{D}}$) and **NEXPMCSPACE**$_{\mathcal{D}}$ (resp. **NEXPMCSPACE**$_{\mathcal{D}}$).

3 An Alternative Definition of Space Complexity for P Systems

In this section, we first give a different definition of space complexity for P systems with active membranes. This definition considers the information stored in the objects of the systems, and not the single objects themselves. In other words, we store, using binary numbers, the multiplicity of each object in each membrane, thus reducing the amount of needed space with respect to the definition of space given in the previous section. We will refer to this definition of space by *binary space*, and we will add a symbol B where appropriate, to distinguish between the definitions referring to this new measure and the definitions recalled in the previous section.

Definition 7. *Consider a configuration $\mathcal{C}$ of a P system Π. Let us denote by $h_1, h_2, ..., h_z$ the membranes of the current membrane structure (we stress the fact that z can be smaller, equal, or greater than the initial number of membranes d, due to dissolution and duplication of membranes), and by $|O_{i,j}|$ the multiplicity of object i within region j. The binary size $|\mathcal{C}|_B$ of a configuration $\mathcal{C}$ is defined as:*

$$|\mathcal{C}|_B = z \cdot \log|\Lambda| + \left(\sum_{j=1}^{z}\sum_{i=1}^{n}\lceil \log(|O_{i,j}|)\rceil\right) \cdot \log|\Gamma|$$

that is the number of membranes in the current membrane structure multiplied by $\log|\Lambda|$, plus the number of bits required to store the amount of each object in each membrane multiplied by $\log|\Gamma|$.

If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a computation of Π, then the binary space *required by $\boldsymbol{\mathcal{C}}$ is defined as*

$$|\boldsymbol{\mathcal{C}}|_B = \max\{|\mathcal{C}_0|_B, \ldots, |\mathcal{C}_k|_B\}.$$

The binary space *required by Π itself is then obtained by computing the binary space required by all computations of Π and taking the supremum:*

$$|\Pi|_B = \sup\{|\boldsymbol{\mathcal{C}}|_B : \boldsymbol{\mathcal{C}} \text{ is a computation of } \Pi\}.$$

Finally, let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems, and let $s\colon \mathbb{N} \to \mathbb{N}$. We say that $\mathbf{\Pi}$ operates within binary space bound *s if and only if $|\Pi_x|_B \leq s(|x|)$ for each $x \in \Sigma^\star$.*

We can thus define space complexity classes considering this new size measure like we did in the previous section. By $\mathbf{MCBSPACE}_{\mathcal{D}}(f(n))$ we denote the class of languages which can be decided by uniform families of confluent P systems of type $\mathcal{D}$, where each $\Pi_x \in \mathbf{\Pi}$ operates within space bound $f(|x|)$, considering this new definition of binary space. Similarly, we can define the usual complexity classes like we did in the previous section, simply adding a B to underline the use of this new definition of space. For instance, the class of problems solvable in polynomial binary space will be denoted by $\mathbf{PMCBSPACE}_{\mathcal{D}}$.

Once these notions have been defined, we are ready to state some results obtained by considering various complexity classes defined in terms of binary space. Just like it happens with the classes based on the original definition of space given in [8], some results follow immediately from the definitions (here we state results for semi-uniform families, but it is easy to see that they also hold in the uniform case):

Proposition 1 *The following inclusions hold:*

$$\mathbf{PMCBSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{EXPMCBSPACE}^{\star}_{\mathcal{D}}$$
$$\mathbf{NPMCBSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{NEXPMCBSPACE}^{\star}_{\mathcal{D}}.$$

Proposition 2 $\mathbf{MCBSPACE}^{\star}_{\mathcal{D}}(f) \subseteq \mathbf{NMCBSPACE}^{\star}_{\mathcal{D}}(f)$ *for each* $f \colon \mathbb{N} \to \mathbb{N}$, *and in particular*

$$\mathbf{PMCBSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{NPMCBSPACE}^{\star}_{\mathcal{D}}$$
$$\mathbf{EXPMCBSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{NEXPMCBSPACE}^{\star}_{\mathcal{D}}.$$

The results describing closure properties and providing an upper bound for time requirements of P systems operating in bounded binary space are still valid, too:

Proposition 3 *The complexity classes* $\mathbf{PMCBSPACE}^{\star}_{\mathcal{D}}$, $\mathbf{NPMCBSPACE}^{\star}_{\mathcal{D}}$, $\mathbf{EXPMCBSPACE}^{\star}_{\mathcal{D}}$, *and* $\mathbf{NEXPMCBSPACE}^{\star}_{\mathcal{D}}$ *are all closed under polynomial-time reductions.*

Proof. Consider a language $L \in \mathbf{PMCBSPACE}^{\star}_{\mathcal{D}}$ and let M be the Turing machine constructing the family $\mathbf{\Pi}$ that decides L. Let L' be reducible to L via a polynomial-time computable function f.

We can build a Turing machine M' working as follows: on input x of length n, M' computes $f(x)$; then it behaves like M on input $f(x)$, thus constructing $\Pi_{f(x)}$ (we stress the fact that, for the corresponding result concerning the uniform case, the construction of the P system involves two Turing machines, both operating in polynomial time; in this case, we simulate the composition of the two machines). Since $|f(x)|$ is bounded by a polynomial, M' operates in polynomial time and $\Pi_{f(x)}$ in polynomial binary space; it follows that $\mathbf{\Pi'} = \{\Pi_{f(x)} \mid x \in \Sigma^{\star}\}$ is a polynomially semi-uniform family of P systems deciding L' in polynomial binary space. Thus $L' \in \mathbf{PMCBSPACE}^{\star}_{\mathcal{D}}$.

The proof for the three other classes is analogous.

Proposition 4 $\mathbf{MCBSPACE}^{\star}_{\mathcal{D}}(f)$ *is closed under complement for each function* $f\colon \mathbb{N} \to \mathbb{N}$.

Proof. By reversing the roles of objects *yes* and *no*, the complement of a language can be decided.

4 Comparison with standard computational complexity classes

In this section we compare the standard computational complexity classes with the complexity classes defined in the framework of P systems working in binary space.

Most results can be obtained as an immediate consequence of the results given in [8], simply considering that $\mathbf{MCSPACE}_{\mathcal{D}}(f(n)) \subseteq \mathbf{MCBSPACE}_{\mathcal{D}}(f(n))$.

Thus, recalling various results from [8], we have:

Proposition 5 *Let us denote by* $\mathcal{EAM}$ *and* $\mathcal{AM}^0$ *the classes of P systems with active membranes using only elementary membrane division and without polarizations, respectively. The following results hold (we denote a result that holds for both semi-uniform and uniform systems by* $[*]$*):*

$$\begin{aligned}
\mathbf{NP} \cup \mathbf{coNP} &\subseteq \mathbf{EXPMCSPACE}^{\star}_{\mathcal{EAM}} \subseteq \mathbf{EXPMCBSPACE}^{\star}_{\mathcal{EAM}}\\
\mathbf{PSPACE} &\subseteq \mathbf{EXPMCSPACE}^{\star}_{\mathcal{AM}} \subseteq \mathbf{EXPMCBSPACE}^{\star}_{\mathcal{AM}}\\
\mathbf{PSPACE} &\subseteq \mathbf{EXPMCSPACE}_{\mathcal{AM}} \subseteq \mathbf{EXPMCBSPACE}^{\star}_{\mathcal{AM}}\\
\mathbf{PSPACE} &\subseteq \mathbf{EXPMCSPACE}^{[*]}_{\mathcal{AM}^0} \subseteq \mathbf{EXPMCBSPACE}^{[*]}_{\mathcal{AM}^0}
\end{aligned}$$

An interesting research topic concerns the classes for which the inclusion $\mathbf{MCSPACE}_{\mathcal{D}}(f(n)) \subseteq \mathbf{MCBSPACE}_{\mathcal{D}}(f(n))$ is proper and, considering the above inclusions, whether or not the same results can be obtained with stricter binary space classes, by exploiting the improved information storage related to objects with respect to the standard space definition.

Some partial results in this respect are the following:

Theorem 6 *Let us denote by* $\mathcal{NAM}$ *the class of P systems with active membranes that do not use membrane division. The following result holds:* $\mathbf{P} = \mathbf{MCSPACE}^{\star}_{\mathcal{NAM}}(O(1)) = \mathbf{MCBSPACE}^{\star}_{\mathcal{NAM}}(O(1))$

Proof. The inclusion $\mathbf{P} \subseteq \mathbf{MCSPACE}^{\star}_{\mathcal{NAM}}(O(1))$ follows immediately from the definition of semiuniform P systems. Consider a language L in $\mathbf{P}$ and a string x; a deterministic Turing machine can create in polynomial time a P system having a single membrane and one single object *yes* or *no*, directly answering the question whether or not $x \in L$. The inclusion $\mathbf{MCSPACE}^{\star}_{\mathcal{NAM}}(O(1)) \subseteq \mathbf{MCBSPACE}^{\star}_{\mathcal{NAM}}(O(1))$ follows, as stated above, from the definition of binary space.

For the converse, we simply need to recall that a confluent P system without membrane division can be simulated, in polynomial time, by a deterministic Turing machine, like it was shown in [11]. It is easy to see that the proof works both considering the standard space definition as well as the binary space definition for P systems.

Another interesting result concerning the standard definition of space in the framework of P systems was presented in [9], and it focuses on the type of resources used. In particular, a solution for the **PSPACE**-complete problem QUANTIFIED 3SAT was given, for uniform systems using only communication rules (hence no evolution, membrane division and dissolution rules were used), thus proving the inclusion of **PSPACE** in this class. Once again, since the definition of binary space allows a more efficient allocation of space, the result is still valid:

Proposition 7 *Let $\mathcal{AM}(-\mathrm{ev}, +\mathrm{com}, -\mathrm{dis}, -\mathrm{div})$ be the class of P systems with active membranes using only communication rules (no evolution, dissolution, nor division of membranes). Then* $\mathbf{PSPACE} \subseteq \mathbf{PMCBSPACE}_{\mathcal{AM}(-\mathrm{ev},+\mathrm{com},-\mathrm{dis},-\mathrm{div})} \subseteq \mathbf{PMCBSPACE}^{*}_{\mathcal{AM}(-\mathrm{ev},+\mathrm{com},-\mathrm{dis},-\mathrm{div})}$.

Once again, it would be interesting to understand whether or not the result remains valid for a smaller binary space class. In this case, the question can be answered negatively, by considering a result presented in [10]. In the article, it was shown that recognizer P systems with active membranes using polynomial space characterize the complexity class **PSPACE**. The result holds for both confluent and nonconfluent systems, and even in the case that non-elementary division is used. In particular, it was pointed out that such systems can be simulated by polynomial space Turing machines.

By considering the alternative definition for binary space, we can thus obtain the corresponding theorem:

Theorem 8 *Let Π be a nonconfluent P system with active membranes, running in binary space S. Then, it can be simulated by a deterministic Turing machine in space $O(S)$.*

Proof. We simulate Π by a non-deterministic Turing machine N, which can then be reduced to polynomial deterministic space by using Savitch's theorem [5].

The current configuration of Π can be stored explicitly by N: the membrane structure is represented as a rooted tree, where each node is a membrane and contains the information concerning its label, its charge, the multiset of objects in the region, and a list of children nodes (i.e. the membranes immediately inside it). To represent the multiset of objects inside each region, tuples of integers encoded in binary can be used, with one entry for each object type in the alphabet.

Since the simulation algorithm is the same as in [10], it is still valid that the space required to store further information needed to carry on the simulation is limited by S.

It follows that the total requested amount of space for the simulation is of the same order as the one required by Π, that is, $O(S)$.

It follows immediately from this theorem and from Proposition 7:

Theorem 9 *Let $\mathcal{D}$ be a class of P systems with active membranes using at least communication rules. Then* $[\mathbf{N}]\mathbf{PMCBSPACE}_{\mathcal{D}}^{[\star]} = \mathbf{PSPACE}$*, where* $[\mathbf{N}]$ *denotes optional nonconfluence, and* $[\star]$ *optional semi-uniformity.*

In [2] it was shown that exponential space Turing machines can be simulated by polynomially uniform exponential-space P systems with active membranes. In view of this result and of Theorem 8, and of the definition of binary space, we have the following:

Theorem 10 $\mathbf{EXPSPACE} = \mathbf{EXPMCBSPACE}_{\mathcal{AM}} = \mathbf{EXPMCBSPACE}_{\mathcal{AM}}^{\star} = \mathbf{NEXPMCBSPACE}_{\mathcal{AM}}^{\star}$

Proof. The following inclusions hold by definition:
$\mathbf{EXPMCBSPACE}_{\mathcal{AM}} \subseteq \mathbf{EXPMCBSPACE}_{\mathcal{AM}}^{\star} \subseteq \mathbf{NEXPMCBSPACE}_{\mathcal{AM}}^{\star}$, whereas it is easy to see that the inclusion $\mathbf{NEXPMCBSPACE}_{\mathcal{AM}}^{\star} \subseteq \mathbf{EXPSPACE}$ is an immediate corollary of theorem 8.

Finally, the inclusion of $\mathbf{EXPSPACE}$ in $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$ is proved in [2, Theorem 8]. Recalling that $\mathbf{EXPMCSPACE}_{\mathcal{AM}} \subseteq \mathbf{EXPMCBSPACE}_{\mathcal{AM}}$, it follows $\mathbf{EXPSPACE} \subseteq \mathbf{EXPMCBSPACE}_{\mathcal{AM}}$ $\square$

Hence, also in this case, considering binary space instead of the standard one does not result in improved efficiency. Moreover, when we consider an exponential amount of space, we can show that the classes coincide: in fact, considering the theorem just proved and recalling [1, Corollary 1] proving the same results for classes with the original definition of space for P systems, we have

Corollary 11 $\mathbf{EXPSPACE} = \mathbf{EXPMCSPACE}_{\mathcal{AM}} = \mathbf{EXPMCSPACE}_{\mathcal{AM}}^{\star} = \mathbf{NEXPMCSPACE}_{\mathcal{AM}}^{\star} = \mathbf{EXPMCBSPACE}_{\mathcal{AM}} = \mathbf{EXPMCBSPACE}_{\mathcal{AM}}^{\star} = \mathbf{NEXPMCBSPACE}_{\mathcal{AM}}^{\star}$

5 Conclusions

We have proposed an alternative space complexity measure for P systems with active membranes, where the multiplicity of each object in each membrane is stored by using binary numbers. We have defined the corresponding complexity classes and we have compared some of them both with standard space complexity classes and with complexity classes defined in the framework of P systems considering the original definition of space ([8]).

An interesting research topic is to compare such classes for different amounts of allowed space. In particular, it would be interesting to find specific classes defined in terms of binary space which stricly contain classes defined in terms of standard space in the framework of P systems, thus proving that storing in an efficient way the information concerning objects can really be exploited. We showed that, when an exponential amount of space is considered, these classes do not differ. Only partial answers have been obtained for a polynomial amount of space. We expect that the differences can be evident when considering sublinear space complexity classes.

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, G., Pérez-Hurtado, I., Romero-Campero, F.J. (eds.) Tenth Brainstorming Week on Membrane Computing, Volume I. pp. 35–60. No. 1/2012 in RGNC Reports, Fénix Editora (2012), `http://www.gcn.us.es/icdmc2012_proceedings`
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), `https://doi.org/10.1016/j.tcs.2013.11.015`
3. 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), `http://theo.cs.ovgu.de/jalc/search/j19_i.html`
4. 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`
5. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
6. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics **6**(1), 75–90 (2001)
7. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
8. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. International Journal of Computers, Communications & Control **4**(3), 301–310 (2009), `http://univagora.ro/jour/index.php/ijccc/article/view/2779`
9. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. Natural Computing **10**(1), 167–182 (2011), `https://doi.org/10.1007/s11047-010-9189-x`
10. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. International Journal of Foundations of Computer Science **22**(1), 65–73 (2011), `https://doi.org/10.1142/S0129054111007836`
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), `https://doi.org/10.1007/978-1-4471-0313-4_21`

Catalytic P Systems with Weak Priority of Catalytic Over Non-catalytic Rules

Artiom Alhazov[1], Rudolf Freund[2], and Sergiu Ivanov[3]

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[3] IBISC, Univ Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

Summary. Catalytic P systems are among the first variants of membrane systems ever considered in this area. This variant of systems also features some prominent computational complexity questions, and in particularly the problem of using only one catalyst: is one catalyst enough to allow for generating all recursively enumerable sets of multisets? Several additional ingredients have been shown to be sufficient for obtaining even computational completeness with only one catalyst. In this paper we show that one catalyst is sufficient for obtaining even computational completeness if catalytic rules have weak priority over the non-catalytic rules.

1 Introduction

Membrane systems were introduced in [8] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. During two decades now membrane computing has attracted the interest of many researchers, and its development is documented in two textbooks, see [9] and [10]. For actual information see the P systems webpage [12] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

One basic feature of P systems already presented in [8] is the maximally parallel derivation mode, i.e., using non-extendable multisets of rules in every derivation step. The result of a computation can be extracted when the system halts, i.e., when no rule is applicable any more. Catalysts are special symbols which allow only one object to evolve in its context (in contrast to promoters) and in their basic variant never evolve themselves, i.e., a catalytic rule is of the form $ca \to cv$, where c is a catalyst, a is a single object and v is a multiset of objects. In contrast, non-catalytic rules in catalytic P systems are non-cooperative rules of the form $a \to v$.

From the beginning, the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges regarding (catalytic) P systems. In [3] it has already been shown that two catalysts are enough for generating any recursively enumerable set of multisets, without any additional ingredients like a priority relation on the rules as used in the original definition. As already known from the beginning, without catalysts only regular (semi-linear) sets can be generated when using the standard halting mode, i.e., a result is extracted when the system halts with no rule being applicable any more. As shown, for example, in [5], using various additional ingredients, i.e., additional control mechanisms, one catalyst can be sufficient: in P systems with label selection, only rules from one set of a finite number of sets of rules in each computation step are used; in time-varying P systems, the available sets of rules change periodically with time. On the other hand, for catalytic P systems with only one catalyst a lower bound has been established in [6]: P systems with one catalyst can simulate partially blind register machines, i.e., they can generate more than just semi-linear sets.

In this paper we now return to the idea of using a priority relation on the rules, but take only a very weak form of such a priority relation: we only require that overall in the system catalytic rules have weak priority over non-catalytic rules. This means that the catalyst c must not stay idle if the current configuration contains an object a with which it may cooperate in a rule $ca \to cv$; all remaining objects evolve in the maximally parallel way with non-cooperative rules. On the other hand, if the current configuration does not contain an object a with which the catalyst c may cooperate in a rule $ca \to cv$, c may stay idle and *all* objects evolve in the maximally parallel way with non-cooperative rules. Even without using more than this weak priority of catalytic rules over the non-catalytic (non-cooperative) rules, computational completeness can be established for catalytic P systems with only one catalyst, which is the main result of our paper.

2 Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V. The *empty string* is denoted by λ. A *multiset* M with underlying set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $M = (A, f)$ is a multiset then its *support* is defined as $supp(M) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (respectively finite) if its support is the empty set (respectively a finite set). If $M = (A, f)$ is a finite multiset over A and $supp(M) = \{a_1, \dots, a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ over the alphabet $\{a_1, \dots, a_k\}$, and, moreover, all permutations of this string precisely identify the same multiset M. For further notions and results in formal language theory we refer to textbooks like [2] and [11].

2.1 Register Machines

Register machines are well-known universal devices for computing (or generating or accepting) sets of vectors of natural numbers.

Definition 1. *A* register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- *m is the number of registers,*
- *P is the set of instructions bijectively labeled by elements of B,*
- *$l_0 \in B$ is the initial label, and*
- *$l_h \in B$ is the final label.*

The instructions of M can be of the following forms:

- *$p : (ADD(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  Increase the value of register r by one, and non-deterministically jump to instruction q or s.
- *$p : (SUB(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  *If the value of register r is not zero then decrease the value of register r by one (*decrement *case) and jump to instruction q, otherwise jump to instruction s (*zero-test *case).*
- *$l_h : HALT$.*
  Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.*

In the *accepting* case, a computation starts with the input of an l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *computing* case, a computation starts with the input of a l-vector of natural numbers in its first l registers and by executing the first instruction of

P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

A *configuration* of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. M is called *deterministic* if the ADD-instructions all are of the form $p : (ADD(r), q)$.

For useful results on the computational power of register machines, we refer to [7]; for example, to prove our main theorem, we need the following formulation of results for register machines generating or accepting recursively enumerable sets of vectors of natural numbers with k components or computing partial recursive relations on vectors of natural numbers:

Proposition 1. *Deterministic register machines can accept any recursively enumerable set of vectors of natural numbers with l components using precisely $l + 2$ registers. Without loss of generality, we may assume that at the end of an accepting computation all registers are empty.*

Proposition 2. *Register machines can generate any recursively enumerable set of vectors of natural numbers with k components using precisely $k+2$ registers. Without loss of generality, we may assume that at the end of an accepting computation the first two registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

Proposition 3. *Register machines can compute any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l + 2 + k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l + 2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

In all cases it is essential that the output registers never need to be decremented.

2.2 Partially Blind Register Machines

We now consider one-way nondeterministic machines which have registers allowed to hold positive or negative integers and which accept by final state with all registers being zero. Such machines are called *blind* if their actions depend on state and input alone and not on the register configuration. They are called *partially blind* if they block when any register is negative (i.e., only non-negative register contents is allowed) but do not know whether or not any of the registers contains zero.

Definition 2. *A* partially blind register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- *m is the number of registers,*
- *P is the set of instructions bijectively labeled by elements of B,*
- *$l_0 \in B$ is the initial label, and*
- *$l_h \in B$ is the final label.*

The instructions of M can be of the following forms:

- *$p : (ADD(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  Increase the value of register r by one, and non-deterministically jump to instruction q or s.
- *$p : (SUB(r), q)$, with $p \in B \setminus \{l_h\}$, $q \in B$, $1 \leq r \leq m$.*
  If the value of register r is not zero then decrease the value of register r by one and jump to instruction l_2, otherwise abort the computation.
- *$l_h : HALT$.*
  Stop the execution of the register machine.

Again, a *configuration* of a partially blind register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.

A computation works as for a register machine, yet with the restriction that a computation is aborted if one tries to decrement a register which is zero. Moreover, computing, accepting or generating now also requires all registers (except output registers) to be empty at the end of the computation.

Example 1. In [6] it was shown that the vector set

$$S = \{(n, m) \mid 0 \leq n, n \leq m \leq 2^n\}$$

(which is not semi-linear) can be generated by a P system with only one catalyst and 19 rules.

2.3 Catalytic P Systems

As in [6], the following definition cites Definition 4.1 in Chapter 4 of [10].

Definition 3. *An* extended catalytic P system *of degree $m \geq 1$ is a construct*

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$$

where

- *O is the alphabet of objects;*

- *$C \subseteq O$ is the alphabet of catalysts;*
- *μ is a membrane structure of degree m with membranes labeled in a one-to-one manner with the natural numbers $1, \dots, m$;*
- *$w_1, \dots, w_m \in O^*$ are the multisets of objects initially present in the m regions of μ;*
- *R_i, $1 \leq i \leq m$, are finite sets of evolution rules over O associated with the regions $1, 2, \dots, m$ of μ; these evolution rules are of the forms $ca \to cv$ or $a \to v$, where c is a catalyst, a is an object from $O \setminus C$, and v is a string from $((O \setminus C) \times \{here, out, in\})^*$;*
- *$i_0 \in \{0, 1, \dots, m\}$ indicates the output region of Π.*

The membrane structure and the multisets in Π constitute a *configuration* of the P system; the *initial configuration* is given by the initial multisets $w_1, \dots, w_m$. A transition between configurations is governed by the application of the evolution rules, which is done in the maximally parallel way, i.e., only applicable multisets of rules which cannot be extended by further rules are to be applied to the objects in all membrane regions.

The application of a rule $u \to v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to the targets *in* and *out*. We refer to [10] for further details and examples.

The P system continues with applying multisets of rules in the maximally parallel way until there remain no applicable rules in any region of Π. Then the system *halts*. We consider the number of objects from $O \setminus C$ contained in the output region i_0 at the moment when the system halts as the *result* of the underlying computation of Π. The system is called *extended* since the catalytic objects in C are not counted to the result of a computation. Yet as often done in the literature, in the following we will omit the term *extended* and just speak of *catalytic P systems*, especially as we will restrict ourselves to P systems with only one catalyst.

The set of results of all computations possible in Π is called the set of natural numbers *generated by Π* and it is denoted by $N(\Pi)$ if we only count the total number of objects in the output membrane; if we distinguish between the multiplicities of different objects, we obtain a set of vectors of natural numbers denoted by $Ps(\Pi)$.

Remark 1. As in this paper we only consider catalytic P systems with only one catalyst, without loss of generality, we can restrict ourselves to one-membrane catalytic P systems with the single catalyst in the skin membrane, by taking into account the well-known flattening process, e.g., see [4].

Remark 2. Finally, we make the convention that a one-membrane catalytic P system with the single catalyst in the skin membrane and with internal output in the skin membrane, not taking into account the single catalyst c for the results, throughout the rest of the paper will be described without specifying the trivial membrane structure or the output region (assumed to be the skin membrane), i.e., we will just write

$$\Pi = (O, \{c\}, w, R)$$

where O is the set of objects, c is the single catalyst, w is the initial input specifying the initial configuration, and R is the set of rules.

As already mentioned earlier, the following result was shown in [6], establishing a lower bound for the computational power of catalytic P systems with only one catalyst:

Proposition 4. *Catalytic P systems with only one catalyst have at least the computational power of partially blind register machines.*

3 Weak Priority of Catalytic Rules

In this paper we now study catalytic P systems with only one catalyst in which the catalytic rules have weak priority over the non-catalytic rules.

Example 2. To illustrate this weak priority of catalytic rules over the non-catalytic rules, consider the rules $ca \to cb$ and $a \to d$. If the current configuration contains $k > 0$ copies of a, then the catalytic rule $ca \to cb$ *must* be applied to one of the copies, while the rest of objects a may be taken up by the non-catalytic rule $a \to d$. In particular, if $k = 1$, only $ca \to cb$ may be applied.

We would like to highlight the fact that weak priority of catalytic rules is much weaker than the general weak priority, as the priority relation is only constrained by the types of rules.

Remark 3. The reverse weak priority, i.e., non-catalytic rules having priority over catalytic rules, is useless, since it is equivalent to removing all catalytic rules for which there are non-catalytic rules with the same symbol on the left-hand side of the rule. In that way we just end up with an even restricted variant of P systems with only one catalyst.

3.1 Computational Completeness with Weak Priority

In this section, we show that catalytic P systems with one catalyst only and with weak priority of catalytic rules are computationally complete.

Theorem 1. *Catalytic P systems with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules are computationally complete.*

Proof. Given an arbitrary register machine $M = (m, B, l_0, l_h, P)$ we will construct a corresponding catalytic P system with one membrane and one catalyst $\Pi = (O, \{c\}, w, R)$ simulating M. Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M, as stated in Proposition 1, Proposition 2, and Proposition 3, fulfills the condition that on the output registers we never apply any *SUB*-instruction. The following proof is given for the most general case of a register machine computing any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used. In fact, the proof works for any number n of decrementable registers, no matter how many of them are the l input registers and the working registers, respectively.

The main idea behind our construction is that all the symbols except the catalyst c and the output symbols (representing the contents of the output registers) go through a cycle of length $2n$ where n is the number of decrementable registers of the simulated register machine. When the symbols are traversing the r-th section of the n sections of length 2, they "know" that they are to probably simulate a *SUB*-instruction on register r of the register machine M.

The alphabet O of symbols includes register symbols $(a_r, 2i-1), (a_r, 2i)$ for every decrementable register r of the register machine and only the register symbol a_r for each of the k output registers r, $m-k+1 \leq r \leq m$, the state symbols $(p, 2i-1), (p, 2i)$, $1 \leq i \leq n$, for every *ADD*-instruction of the register machine as well as, for $p \in B_{SUB(r)}$ the state symbols $(p, 2i-1), (p, 2i)$ for $1 \leq i \leq r$ as well as $(p, 2j-1)^-$ and $(p, 2j)^0$ for $r+1 \leq j \leq n$ for every *SUB*-instruction $p : (SUB(r), q, s)$ of the register machine, i.e., $p \in B_{SUB(r)}$, where $B_{SUB(r)}$ denotes the set of labels of all *SUB*-instruction $p : (SUB(r), q, s)$ of decrementable registers r. Moreover, we use decrement witness symbols λ_r for every decrementable register r, as well as the catalyst c and the trap symbol $\#$. Observing that $n = m-k$, in total we get the following set of objects:

$$\begin{aligned}
O = &\{a_r \mid n+1 \leq r \leq m\} \\
&\cup \{(a_r, i) \mid 1 \leq r \leq n, 1 \leq i \leq 2n\} \\
&\cup \{\lambda_r \mid 1 \leq r \leq n\} \\
&\cup \{(p, i) \mid p \in B_{ADD}, 1 \leq i \leq 2n\} \\
&\cup \{(p, i) \mid p \in B_{SUB(r)}, 1 \leq i \leq 2r\} \\
&\cup \{(p, i)^-, (p, i)^0 \mid p \in B_{SUB(r)}, 2r+1 \leq i \leq 2n\} \\
&\cup \{c, \#\}.
\end{aligned}$$

B_{ADD} denotes the set of labels of *ADD*-instructions.

The starting configuration of Π is

$$w = c(l_0, 1)\alpha_0,$$

where l_0 is the starting label of the machine and α_0 is the multiset encoding the initial values of the registers.

All register symbols a_r, $1 \leq r \leq n$, representing the contents of decrementable registers, are equipped with the rules evolving them throughout the whole cycle:

$$(a_r, i) \rightarrow (a_r, i+1), 1 \leq r \leq 2n-1; \qquad (a_r, 2n) \rightarrow (a_r, 1). \tag{1}$$

The construction also includes the trap rule $\# \rightarrow \#$: once the trap symbol $\#$ is introduced, it will always keep the system busy and prevent it from halting and thus from producing a result.

For simulating *ADD*-instructions we also need the following rules:

Increment $p : (ADD(r), q, s)$*:*

The (variants of the) symbol p cycles together with all the other symbols, always involving the catalyst:

$$c(p, i) \rightarrow c(p, i+1), \ \ 1 \leq i \leq 2n-1. \tag{2}$$

At the end of the cycle, the register is incremented and the non-deterministic jump to q or s occurs: for r being a decrementable register, we take

$$c(p, 2n) \rightarrow c(q, 1)(a_r, 1), \qquad c(p, 2n) \rightarrow c(s, 1)(a_r, 1), \tag{3}$$

whereas for r being a register never to be decremented, we take

$$c(p, 2n) \rightarrow c(q, 1)a_r, \qquad c(p, 2n) \rightarrow c(s, 1)a_r \tag{4}$$

The output symbols need not undergo the cycle, in fact, they must not do that because otherwise the computation would never stop. When the computation of the register machine halts, only output symbols will be present, as we have assumed that at the end of a computation all decrementable registers will be empty, i.e., no cycling symbols will be present any more in the P system. Finally, we have to mention that if q or s is the final label l_h, then we take λ instead, which means that also the P system will halt, because, as already explained above, the only symbols left in the configuration will be output symbols, for which no rules exist.

The state symbol is not allowed to evolve without the catalyst:

$$(p, i) \rightarrow \#, \ \ 1 \leq i \leq 2n. \tag{5}$$

Hence, in that way it is guaranteed that the catalyst cannot be used in another way, i.e., affecting a symbol (a_r, i) as explained below during the simulation of a *SUB*-instruction on register r.

Decrement and zero-test $p : (SUB(r), q, s)$*:*

The simulation of a *SUB* instruction is carried out in two steps of the cycle, i.e., in steps $2r-1$ and $2r$.

Before reaching simulation phase r, i.e., step $2r-1$, the state symbol goes through the cycle, necessarily involving the catalyst:

$$c(p,i) \to c(p,i+1) \quad > \quad (p,i) \to \#, \qquad 1 \leq i < 2r-1. \tag{6}$$

Although by the definition of the P systems with priority of catalytic rules, the catalytic rule has priority over the non-catalytic rule for (p,i), we indicate the general priority relation by the sign $<$ (or $>$ for the reverse relation) in order to make the situation even clearer.

In the first step of the simulation phase r, i.e., in step $2r-1$, the state symbol releases the catalyst to try to perform the decrement and to produce a witness symbol if register r is not empty:

$$(p,2r-1) \to (p,2r), \qquad c(a_r,2r-1) \to c\lambda_r. \tag{7}$$

Note that due to the counters identifying the position of the register symbols in the cycle, it is guaranteed that the catalytic rule transforming $(a_r,2r-1)$ picks the correct register symbol. Furthermore, due to the priority of the catalytic rules, one of the the register symbols $(a_r,2r-1)$ *must* be transformed by the catalytic rule if present, instead of continuing along its cycle.

In the second step of simulation phase r, i.e., in step $2r$, the detection of the possible decrement happens. The outcome is stored in the state symbol:

$$\begin{aligned} &c\lambda_r \to c && > \quad \lambda_r \to \#, \\ &(p,2r) \to (p,2r+1)^- && < \quad c(p,2r) \to c(p,2r+1)^0. \end{aligned} \tag{8}$$

If in the first step of the simulation phase the catalyst did manage to decrement the register, it produced λ_r. Thus, in the second step, the catalyst must erase λ_r, because otherwise this symbol will trap the computation (and because catalytic rules have priority). This means that the catalyst is not available to produce $(p,2r{+}1)^0$, and the rule $(p,2r) \to (p,2r{+}1)^-$ must be applied due to the maximally parallel mode. If, on the other hand, the decrement did *not* succeed in the previous step, both rules $(p,2r) \to (p,2r+1)^-$ and $c(p,2r) \to c(p,2r+1)^0$ can be applied, but due to the priority of the catalytic rules, the second rule must be preferred, thus producing $(p,2r{+}1)^0$. Therefore, the superscript of the state symbol correctly reflects the outcome of the decrement: it is $-$ if the decrement succeeded, and 0 if it did not.

After the simulation of the decrement, the state symbols evolve to the end of the cycle and produce the corresponding next state symbols:

$$\begin{aligned} &(p,i)^- \to (p,i+1)^-, \quad r+2 \leq i \leq n, && (p,n+1)^- \to (q,1), \\ &(p,i)^0 \to (p,i+1)^0, \quad r+2 \leq i \leq n, && (p,n+1)^0 \to (s,1). \end{aligned} \tag{9}$$

If the register r is the last decrementable one, i.e., $r = n$, then equations 8 and 9 together read as follows:

$$\begin{aligned} &c\lambda_n \to c && > \quad \lambda_n \to \#, \\ &(p,n+1) \to (q,1) && < \quad c(p,n+1) \to c(s,1). \end{aligned} \tag{10}$$

Finally, we again mention that if q or s is the final label l_h, then we take λ instead, which means that not only the register machine but also the P system halts, because, as already explained above, the only symbols left in the configuration will be output symbols, for which no rules exist. $\square$

We would also like to emphasize that the simulation is what may be called toxic/trap-deterministic: the only non-deterministic choice happens between a rule producing a trap symbol # and another one which does not introduce #. This means that the appearance of the trap symbol may immediately abort the computation, which is the concept used for toxic P systems as introduced in [1]. Using the trap symbol # as such a toxic object, the only successful computations are simulating register machines in a quasi-deterministic way with a look-ahead of one, i.e., considering all possible configurations computable from a given one, there is at most one successful continuation of the computation.

For future research it remains a challenging question whether the length of the cycle now being $2n$ can still be reduced.

4 Conclusion

In this paper we revisited a classic problem of computational complexity in membrane computing: can catalytic P systems with only one catalyst already generate all recursively enumerable sets of multisets? This problem has been standing tall for many years, and nobody has yet managed to give it a positive or a negative answer. In this paper, we come closer to showing computational completeness: we give a construction that simulates an arbitrary register machine with a very weak ingredient—the weak priority of catalytic rules over non-catalytic rules.

On the other hand, we still conjecture that P systems with one catalyst and no additional control mechanisms cannot reach computational completeness. Finding an answer to the question of characterizing the computational power of P systems with one catalyst therefore still remains one of the biggest challenges in the theory of P systems, although the result established in our paper has made the gap between the computational power of P systems with one catalyst and computational completeness smaller again.

The result obtained in this paper can also be extended to P systems dealing with strings, following the definitions and notions used in [6], thus showing computational completeness for computing with strings.

Acknowledgements

The ideas, concepts, and results described in this paper have mainly been developed in the inspiring atmosphere of the 18th Brainstorming Week on Membrane Computing during the first week of February 2020 in Sevilla.

Sergiu Ivanov is partially supported by the Paris region via the project DIM RFSI n°2018-03 “Modèles informatiques pour la reprogrammation cellulaire”.

References

1. Artiom Alhazov and Rudolf Freund. P systems with toxic objects. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 99–125. Springer, 2014.
2. Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory.* Springer, 1989.
3. Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2):251–266, 2005.
4. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Claudio Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.
5. Rudolf Freund, Marion Oswald, and Gheorghe Păun. Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundam. Inform.*, 136(1–2):59–84, 2015.
6. Rudolf Freund and Petr Sosík. On the power of catalytic P systems with one catalyst. In Grzegorz Rozenberg, Arto Salomaa, José M. Sempere, and Claudio Zandron, editors, *Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers*, volume 9504 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2015.
7. Marvin L. Minsky. *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
8. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
9. Gheorghe Păun. *Membrane Computing: An Introduction.* Springer, 2002.
10. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
11. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages.* Springer, 1997.
12. The P Systems Website. `http://ppage.psystems.eu/`.

P Systems with Limited Capacity

Artiom Alhazov[1], Rudolf Freund[2], and Sergiu Ivanov[3]

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[3] IBISC, Univ Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

Summary. P systems are a model of compartmentalized multiset rewriting inspired by the structure and functioning of the living cell. In this paper, we focus on a variant in P systems in which membranes have limited capacity, i.e., the number of objects they may hold is statically bounded. This feature corresponds to an important physical property of cellular compartments. We propose several possible semantics of limited capacity and show that one of them allows real-time simulations of partially blind register machines, while the other one allows for obtaining computational completeness.

1 Introduction

Membrane systems were introduced in [9] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. Among the basic features of the original model are the hierarchical arrangement of the membranes and the parallel evolution of the objects contained in the membrane compartments. Usually a result is obtained if the computation halts, i.e., if no rule is applicable any more.

In this paper we consider an additional feature also inspired by biology, namely the limited capacity of cells to include objects – in total or of a specific kind. When the number of cells is not bounded as in P systems with active membranes, this biological feature of limited capacity can be kept for all cells below a given fixed bound. On the other hand, in the standard hierarchical model with a static number of cells, or, even if we allowed membrane dissolution, with a fixed upper bound for the number of cells, we can only limit the number of specific objects and have to allow an unbounded number of other objects when aiming at non-trivial theoretical results.

When the number of cells is not bounded because of using membrane creation and/or membrane division (together with membrane dissolution), the number of objects in one cell/membrane can even be restricted to one, still allowing for obtaining computational completeness, thereby counting the number of membranes/cells instead of the number of objects in an output membrane/cell; for example, see [1, 4, 3].

In this paper we now propose this new feature of limited capacity for (specific) objects to be contained in a cell or a membrane region and several semantics of how to treat the situation when the application of a (multiset of) rule(s) would violate this limiting condition. We will only investigate two special variants in more detail, both of them blocking or aborting computations which try to apply a multiset of rules leading to a violation of the limited capacity conditions. For the first variant we show that it allows for real-time simulations of partially blind register machines (PBRM), while the other variant allows for obtaining computational completeness.

The development of the fascinating area of membrane computing during the last two decades is documented in two textbooks, see [10] and [11]. For actual information see the P systems webpage [13] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

2 Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V. The *empty string* is denoted by λ. For any $a \in V$ and any string w over A, w_a denotes the number of symbols a in w.

A *multiset* M with underlying set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. For a multiset $M = (A, f)$, its *support* is defined as $supp(M) = \{x \in A \mid f(x) > 0\}$. A multiset is called *empty* or *finite* if its support is the empty set or a finite set, respectively. If $M = (A, f)$ is a finite multiset over A and $supp(M) = \{a_1, \dots, a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ over the alphabet $\{a_1, \dots, a_k\}$, and, moreover, all permutations of this string precisely identify the same multiset M. For any $a \in V$ and any multiset M over A, M_a denotes the number of symbols a in w.

For further notions and results in formal language theory we refer to textbooks like [5] and [12].

2.1 Register Machines

Register machines are well-known universal devices for computing (or generating or accepting) sets of vectors of natural numbers.

Definition 1. *A* register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- *m is the number of registers,*
- *P is the set of instructions bijectively labeled by elements of B,*
- *$l_0 \in B$ is the initial label, and*
- *$l_h \in B$ is the final label.*

The instructions of M can be of the following forms:

- *$p : (ADD(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  Increase the value of register r by one, and non-deterministically jump to instruction q or s.
- *$p : (SUB(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  *If the value of register r is not zero then decrease the value of register r by one (*decrement *case) and jump to instruction q, otherwise jump to instruction s (*zero-test *case).*
- *$l_h : HALT$.*
  Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.*

In the *accepting* case, a computation starts with the input of an l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *computing* case, a computation starts with the input of a l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

A *configuration* of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. M is called *deterministic* if the ADD-instructions all are of the form $p : (ADD(r), q)$.

For useful results on the computational power of register machines, we refer to [8]; for example, for proving computational completeness results for specific variants of P systems, usually the following formulations of results for register machines generating or accepting recursively enumerable sets of vectors of natural numbers with k components or computing partial recursive relations on vectors of natural numbers are helpful:

Proposition 1. *Deterministic register machines can accept any recursively enumerable set of vectors of natural numbers with l components using precisely $l+2$ registers. Without loss of generality, we may assume that at the end of an accepting computation all registers are empty.*

Proposition 2. *Register machines can generate any recursively enumerable set of vectors of natural numbers with k components using precisely $k+2$ registers. Without loss of generality, we may assume that at the end of an accepting computation the first two registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

Proposition 3. *Register machines can compute any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

In all cases it is essential that the output registers never need to be decremented.

2.2 Partially Blind Register Machines

We now consider one-way nondeterministic machines which have registers allowed to hold positive or negative integers and which accept by final state with all registers being zero. Such machines are called *blind* if their actions depend on state and input alone and not on the register configuration. They are called *partially blind* if they block when any register is negative (i.e., only non-negative register contents is allowed) but do not know whether or not any of the registers contains zero.

Definition 2. *A* partially blind register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- *m is the number of registers,*
- *P is the set of instructions bijectively labeled by elements of B,*
- *$l_0 \in B$ is the initial label, and*

- $l_h \in B$ *is the final label.*

  The instructions of M *can be of the following forms:*

- $p : (ADD(r), q, s)$, *with* $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.
  Increase the value of register r *by one, and non-deterministically jump to instruction* q *or* s.
- $p : (SUB(r), q)$, *with* $p \in B \setminus \{l_h\}$, $q \in B$, $1 \leq r \leq m$.
  If the value of register r *is not zero then decrease the value of register* r *by one and jump to instruction* l_2, *otherwise abort the computation.*
- $l_h : HALT$.
  Stop the execution of the register machine.

Again, a *configuration* of a partially blind register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.

A computation works as for a register machine, yet with the restriction that a computation is aborted if one tries to decrement a register which is zero. Moreover, computing, accepting or generating now also requires all registers (except output registers) to be empty at the end of the computation.

2.3 P Systems

The standard model of hierarchical P systems can be defined as follows, for example, see [11] for several variants:

Definition 3. *A (hierarchical)* P system *of degree* $m \geq 1$ *is a construct*

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$$

where

- O *is the alphabet of objects;*
- μ *is a membrane structure of degree* m *with membranes labeled in a one-to-one manner with the natural numbers* $1, \ldots, m$;
- $w_1, \ldots, w_m \in O^*$ *are the multisets of objects initially present in the* m *regions of* μ;
- R_i, $1 \leq i \leq m$, *are finite sets of evolution rules over* O *associated with the regions* $1, 2, \ldots, m$ *of* μ; *these evolution rules are of the forms* $u \to v$ *where and* u *is a multiset over* O *and* v *is a string from* $((O \setminus C) \times \{here, out, in\})^*$;
- $i_0 \in \{0, 1, \ldots, m\}$ *indicates the output region of* Π.

The membrane structure and the multisets in Π constitute a *configuration* of the P system; the *initial configuration* is given by the initial multisets $w_1, \ldots, w_m$. A transition between configurations is governed by the application of the evolution rules, which is done in the maximally parallel way, i.e., only applicable multisets of rules which cannot be extended by further rules are to be applied to the objects in all membrane regions.

The application of a rule $u \to v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to the targets *in* and *out*.

The P system continues with applying multisets of rules in the maximally parallel way until there remain no applicable rules in any region of Π. Then the system *halts*. We consider the number of objects from O contained in the output region i_0 at the moment when the system halts as the *result* of the underlying computation of Π. The set of results of all computations possible in Π is called the set of natural numbers *generated by* Π and it is denoted by $N(\Pi)$ if we only count the total number of objects in the output membrane; if we distinguish between the multiplicities of different objects, we obtain a set of vectors of natural numbers denoted by $Ps(\Pi)$. We refer to [11] for further details and examples.

A special variant of P systems uses so-called *catalysts*, which are objects which allow other objects to evolve, but never evolve themselves.

Definition 4. *A* catalytic P system *of degree $m \geq 1$ is a construct*

$$\Pi = (O, C, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0)$$

where $C \subseteq O$ is the alphabet of catalysts; the evolution rules *are of the forms $ca \to cv$ or $a \to v$, where c is a catalyst, a is an object from $O \setminus C$, and v is a string from $((O \setminus C) \times \{here, out, in\})^*$; the other ingredients are defined as for hierarchical P systems in Definition 3. A catalytic P system is called* purely catalytic *if all rules are catalytic ones.*

Since the beginning, the question how many catalysts are needed in catalytic and purely catalytic P systems for obtaining computational completeness has been a challenging theoretical question. The following result was shown in [7], establishing a lower bound for the computational power of catalytic P systems with only one catalyst:

Proposition 4. *Catalytic P systems with only one catalyst have at least the computational power of partially blind register machines.*

Example 1. In [7] it was shown that the vector set

$$S = \{(n, m) \mid 0 \leq n, n \leq m \leq 2^n\}$$

(which is not semi-linear) can be generated by some (even extended version of a) PBRM and therefore by a P system with only one catalyst and 19 rules.

As already shown in [6], register machines with $n \geq 2$ decrementable registers can be simulated by catalytic P systems with n catalysts and by purely catalytic P systems with $n + 1$ catalysts.

3 Limited Capacity

In most of the variants of P systems considered in the literature the number of objects in a membrane region is not limited. In this paper, we propose a variant in which the number of objects a membrane may contain is bounded, with the bound already being given in the definition of the system.

In this paper we consider two variants of limiting the capacity – limiting the total capacity of objects in a cell and only limiting the capacity of specific objects in a cell, respectively.

Definition 5. *A P system with* per-membrane limited capacity *is the following construct:*

$$\Pi = (O, \mu, w_1, \dots, w_n, k_1, \dots, k_n, R_1, \dots R_n, i_0),$$

where $k_i \in \mathbb{N} \cup \{\infty\}$ is the total capacity *of membrane i, $1 \leq i \leq n$, meaning that, for $|v_i|$ denoting the contents of memrane i in the current configuration, the condition $|v_i| \leq k_i$ must always be enforced, unless $k_i = \infty$. The other components of the tuple are as in Subsection 2.3.*

Definition 6. *A P system with* per-symbol limited capacity *is the following construct:*

$$\Pi = (O, \mu, w_1, \dots, w_n, K_1, \dots, K_n, R_1, \dots R_n, i_0),$$

where $K_i : O \to \mathbb{N} \cup \{\infty\}$ are functions defining the per-symbol *capacity of membrane i. The condition $w_a \leq K(a)$ must therefore be enforced at all times, for any $a \in O$, unless $K(a) = \infty$.*

In this paper, we will focus on P systems with per-symbol limited capacity.

Remark 1. We immediately remark that the flattening technique which is folklore in the membrane computing community can be applied in the case of P systems with per-symbol limited capacity. Without loss of generality, we therefore in Section 4 will only consider 1-membrane systems, which can be written in a simplified version as follows with omitting the trivial membrane structure and taking the skin membrane 1 as the output membrane:

$$\Pi = (O, w, K, R) \text{ and } \Pi = (O, C, w, K, R) \text{ for catalytic P systems.}$$

3.1 Semantics of Limited Capacity

What should happen if a membrane is about to exceed its capacity (total or per-symbol)? Multiple kinds of behaviors may be considered, for example the following variants:

1. *Blocking behavior:* Prohibit the application of (multisets of) rules which would produce more objects. Attempting to apply such rules blocks the system, and yields no result.

2. *Destructive behavior:* Completely remove the offending membrane from the system, together with its contents.

3. *Dissolutive behavior*: Dissolve the offending membrane, dumping its contents into its parent membrane; in this case, (one of) the parent membrane(s) must allow for more objects, as otherwise the whole system would be dissolved.

4. *Separation behavior:* Divide the offending membrane separating its contents across the child membranes. Since every child membrane only receives a part of the contents of the parent, the capacity constraints may be satisfied.

The separation behavior may be useful for P systems with active membranes, whereas the first three behaviors may also be applied for hierarchical P systems.

In this paper, we focus on the blocking behavior, see 1. Yet there are still at least two possible semantics for the blocking behavior itself under the maximally parallel derivation mode:

Semantics 1: Take all the applicable multisets of rules in the maximally parallel derivation mode, but discard all those multisets which would violate the constraints.

Semantics 2: Take all the applicable multisets of rules in the asynchronous derivation mode, discard the multisets which would violate the constraints, and then pick the non-extendable, i.e., maximal multisets out of these applicable multisets of rules.

To illustrate the difference between these two semantics, consider the following 1-membrane system with limited capacity:

$$\boxed{\begin{array}{c} a \to c \\ b \to c \\ \\ ab \end{array}}$$

It can formally be written as

$$\Pi_{ab} = (\{a, b\}, ab, K_{ab}, \{a \to c, b \to c\})$$

where $K_{ab}(c) = 1$ and $K_{ab}(a) = K_{ab}(b) = \infty$.

In the case of Semantics 1, no multisets of rules not violating the constraint of limiting the capacity of symbols c in the resulting configuration would be applicable, and the P system will block/abort this computation.

On the other hand, under Semantics 2, Π_{ab} would be allowed to apply *either* $a \to c$ *or* $b \to c$, but not both.

4 Computational Power

In this section we investigate the computational power of P systems with limited per-symbol capacity: when operating with Semantics 1, they at least can simulate partially blind register machines in real time; when operating with Semantics 2, they can simulate any register machines and therefore are computationally complete.

4.1 Semantics 1 Allows for Simulating a PBRM in Real Time

In this subsection, we will show that P systems with limited per-symbol capacity operating under Semantics 1 can simulate partially blind register machines (PBRM) in real time: an instruction of the register machine is simulated in one step of the P system. An additional cleanup procedure at the end of the computation takes 3 more steps. In comparison with the result stated in [7] showing that P systems with one catalyst can simulate partially blind register machines (without any further ingredients), we here obtain a real-time simulation, whereas the result there needs a cycle of $n+3$ for each step of the register machine, with n being the number of decrementable registers.

Theorem 1. *Catalytic P systems with one catalyst and per-symbol limited capacity operating with Semantics 1 can simulate partially blind register machines (PBRM) in real time, plus three additional cleanup steps at the end of the computation.*

Proof. Consider an arbitrary partially blind register machine

$$M = (m, B, l_0, l_h, P)\,.$$

The following proof is given for the most general case of a partially blind register machine computing a partial recursive function on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using n of decrementable registers, no matter how many of them are the first l input registers and the working registers, respectively. Moreover, we may assume that on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used. On the other hand, the computation of the PBRM yields a result if and only if at the end of the computation all registers except the output registers are empty.

We now construct the P system

$$\Pi = (O, \{c\}, w_0, K, R)$$

with per-symbol limited capacity operating under Semantics 1 and simulating the PBRM M.

The set of objects of the construction includes register symbols a_r for representing the contents of register r, the catalyst c, and the state symbols $p \in B$. Moreover, we use decrement witness symbols λ_r for every decrementable register r, $1 \leq r \leq n$, as well as the catalyst c and the trap symbol $\#$ and, finally, the additional symbols $a_0, \lambda_0, l_{h'}$. As we will see later, a_0 can be interpreted as a register symbol for an additional decrementable register 0, which during the whole computation has the value 1, i.e., in every configuration we have exactly one copy of a_0, and it is only eliminated in the final cleanup procedure.

Now let $B_{SUB(r)}$ denote the set of labels of *SUB*-instruction $p : (SUB(r), q)$ of decrementable registers r, $B_{SUB} = \bigcup_{1 \leq r \leq n} B_{SUB(r)}$, and B_{ADD} denote the set of labels of *ADD*-instructions, i.e., $B = B_{ADD} \cup B_{SUB}$.

Observing that $n = m - k$, in total we get the following set of objects:

$$\begin{aligned} O &= \{a_r \mid 0 \leq r \leq m\} \cup B \cup \{l_{h'}\} \cup D \cup \{c, \#\}, \\ D &= \{\lambda_r \mid 0 \leq r \leq n\}. \end{aligned}$$

The capacity of the symbols in D is limited to 1, while all other symbols may appear in an unlimited number of copies:

$$\begin{aligned} K(\lambda_r) &= 1, \quad 0 \leq r \leq n, \\ K(x) &= \infty, \;\; x \in O \setminus D. \end{aligned}$$

Moreover, let $D_\emptyset$ denote the multiset containing exactly one copy of each object in D and D_r the multiset containing exactly one copy of each object in D except λ_r.

Then the starting configuration of the P system is defined as

$$w_0 = c\, l_0\, D_\emptyset\, a_0\, \alpha_0,$$

where α_0 is the multiset encoding the initial values of the registers.

The set of rules now is going to be described in several parts below.

First, we want all symbols in D to disappear after one step:
$\lambda_r \to \lambda \in R$ for all $0 \leq r \leq n$.
We also include the traditional trap rule $\# \to \# \in R$.

Increment $p : (ADD(r), q, s)$:

To simulate the *ADD* instruction $p : (ADD(r), q, s)$ without letting the catalyst block the system or do unwanted decrements, the catalyst is forced to process the state symbol:

$$cp \to cqa_rD_\emptyset \qquad cp \to csa_rD_\emptyset \qquad p \to \#.$$

When the label of an ADD instruction is present in the configuration, the catalyst cannot act on any of the register symbols a_r, $0 \leq r \leq m$, because this would leave the state symbol p to be transformed to $\#$ due to the maximally parallel derivation mode. This evolution will not violate the capacity constraints, but introducing the trap symbol will prevent the system from ever halting. Therefore, the catalyst must be used in one of the two rules simulating the increment. Incidentally, these rules also replenish the supply of the symbols from D.

Decrement $p : (SUB(r), q)$ *(no zero test):*

Consider the configuration $c\,p\,D_\emptyset\,a_0\,\alpha$, where α is a string of register symbols describing the current contents of the registers. The following rules have to be applied in this configuration:

$$p \to qD_r \qquad ca_r \to c\lambda_r.$$

All the symbols from $D_\emptyset$ from the *current* configuration will disappear in the next configuration. The rule $p \to qD_r$ will reintroduce almost all of the symbols, except for the particular λ_r corresponding to the register to be decremented. This allows $ca_r \to c\lambda_r$ to be applied in the *current* step, because in the next configuration there is still room for λ_r. All catalytic rules involving a wrong $\lambda_{r'}$ (and therefore a wrong $a_{r'}$) cannot be applied, because they would introduce a second instance of $\lambda_{r'}$, thus blocking the system.

Therefore, the only possible evolution from the configuration $c\,p\,D_\emptyset\,a_0\,\alpha$ is to the configuration $c\,q\,D_\emptyset\,a_0\,\beta$ where $\beta = \alpha - a_r$. Note that if the expected register symbol a_r is not present in α, then there will be no non-extendable multiset of rules including the correct $ca_r \to c\lambda_p$, because then at least the rule $ca_0 \to c\lambda_0$ described below would become applicable, thus blocking (aborting) the computation without producing any result. This behavior corresponds to a crash in the PBRM when it tries to decrement a register which is already empty.

Final zero test, cleanup, and halting:

The simulation of the decrement instruction on register r only works correctly when there are still some register symbols a_r left. Indeed, as already mentioned above, in order to force the computation in the P system to abort if a decrement on an empty register would be tried, we at least would have the rule $ca_0 \to c\lambda_0$, but as long as the decrement symbol λ_0 is re-introduced by applying a rule $p \to qD_r$ simulating a decrement on register r, the computation in the P system will be forced to crash as two symbols λ_0 are not allowed in a configuration.

On the other hand, if finally, the PBRM has reached a configuration with all decrementable registers r, $1 \leq r \leq n$ being empty, we have to allow for a final zero test: in this case the rule $ca_0 \to c\lambda_0$ is welcome to be applied if we have reached the final (halting) label l_h:

$$l_h \to l_{h'}D_{\lambda_0} \qquad ca_0 \to c\lambda_0$$

The additional label $l_{h'}$ is used to check whether all decrementable registers are empty as required for a computation of the PBRM to be successful:

$$l_{h'} \to D_{\lambda_0} \qquad ca_r \to c\#\lambda_0, 1 \leq r \leq n$$

In this step, the catalyst is free to use one of the rules $ca_r \to c\#\lambda_0$ for any non-empty register r without violating the limiting condition for λ_0, hence, the trap symbol # is introduced if and only if any of the decrementable registers is not empty.

If all decrementable registers have been empty, in the final step, the system will just erase the symbols of D_{λ_0}, which will disappear and the system will halt with only symbols a_r for the output registers $n+1 \leq r \leq m$.

This final cleanup phase takes one step to erase a_0, one more step to test the presence of register symbols a_r, $1 \leq r \leq n$, and one final step to erase the last symbols of D_{λ_0}. Hence, in a successful computation this final phase takes three steps.

In the case of the simulation of a non-successful computation of the PBRM, there may be many more steps applying rules $ca_r \to c\#\lambda_0$, possibly already in the second step, but with the trap rule $\# \to \# \in R$ causing an infinite computation we need not take care about this situation in detail. □

Remark 2 (Trapping by limited capacity). Instead of having the rule $\# \to \#$ to implement the trap symbol as a guarantee for an infinite computation and thus for any computation introducing it to not be successful, we can limit its capacity to 1 and use rules of the form $u \to v\#\#$ instead of $u \to v\#$. Alternatively, we could limit the capacity of # to 0, meaning that even having to pick the rule $u \to v\#$ will already block the evolution. This means that, if all non-extendable multisets of rules contain a rule of the form $u \to v\#$, then we must discard all multisets, thereby blocking the evolution without producing any result. This blocking of computations reflects the concept of using toxic objects as introduced in [2].

4.2 Semantics 2 Allows for Computational Completeness

In this subsection, we show that (purely catalytic) P systems with limited per-symbol capacity are computationally complete when operating with Semantics 2 without any additional ingredients.

Remark 3 (Simulating catalytic rules). We first observe that when operating with Semantics 2 we can limit the parallelism of a non-cooperative rule by producing a marker symbol whose capacity is limited to one. For example, consider the rule $p : a \to u\lambda_p$ together with the rule $\lambda_p \to \lambda$ and the limiting condition $K(\lambda_p) = 1$, i.e., the symbol λ_p may not appear in more than one copy. Then, in any multiset of rules allowed to be applied λ_p may appear in at most one copy. This effectively prohibits applying p more than once in any step.

Moreover, we can ensure that the rules compete for the marker symbol just as catalytic rules would compete for a catalyst. For example, consider two catalytic rules $ca \to cu$ and $cb \to cv$. These two rules cannot be applied at the same time, even if both a and b are present, because the catalyst is only present in a single copy. We can ensure the same mutual exclusion by having the symbol λ_c with the capacity limited to 1 ($K(\lambda_c) = 1$), and the rules $a \to u\lambda_c$ and $b \to v\lambda_c$.

Remark 4 (No catalysts needed). As elaborated in Remark 3, catalytic rules can be replaced by non-cooperative rules, i.e., P systems with per-symbol limited capacity operating with Semantics 2 do not need catalysts for simulating purely catalytic P systems.

All together, these observations imply the following results:

Theorem 2. *P systems with per-symbol limited capacity operating with Semantics 2 without catalysts can simulate purely catalytic P systems.*

Since purely catalytic P systems are computationally complete, for example see [6], we immediately derive the following corollary.

Corollary 1. *P systems with per-symbol limited capacity operating with Semantics 2 are computationally complete, even without using catalysts.*

Remark 5 (Trapping by limited capacity). When following the proofs as given in [6] for simulating register machines by [purely] catalytic P systems, often rules introducing the trap symbol $\#$ as well as the rule $\# \to \#$ are used to guarantee an infinite computation and thus any computation introducing it to not be successful. As already explained in Remark 2, we can avoid these rules by limiting the capacity of the trap symbol to 1 and use rules of the form $u \to v\#\#$ instead of $u \to v\#$, or alternatively, limit the capacity of $\#$ to 0, meaning that even having to pick the rule $u \to v\#$ will already block the computation.

5 Conclusion

In this paper, we have introduced the idea of bounding the number of symbols that may appear in the membranes of a P systems. This is a quite natural restriction to consider, given that actual biological membranes are of limited capacity, too. We defined limited total and per-symbol capacities, and defined two possible semantics for handling the overflow. We then showed that Semantics 1 allows non-cooperative P systems to simulate partially blind register machines in real time, with 3 additional cleanup steps at the end of the computation. We also showed that non-cooperative P systems operating under Semantics 2 of limited capacity directly simulate purely catalytic P systems (in real time), yet without needing catalysts, and therefore are computationally complete.

This paper only scratches the surface of the study of P systems with limited capacity. One immediate open problem is that of computational completeness of (catalytic, purely catalytic) P systems with limited capacity operating with Semantics 1 or else characterizing the computational power of these systems.

Furthermore, Section 3 gives three more different behaviors which P systems may adopt when their membranes overflow. In particular, the separation and the dissolutive behaviors may even better represent the phenomena one would expect to observe in overfull membranes in biological cells.

Acknowledgements

The ideas, concepts, and results described in this paper have mainly been developed in the inspiring atmosphere of the 18^{th} Brainstorming Week on Membrane Computing during the first week of February 2020 in Sevilla.

Sergiu Ivanov is partially supported by the Paris region via the project DIM RFSI n°2018-03 "Modèles informatiques pour la reprogrammation cellulaire".

References

1. Artiom Alhazov. P systems without multiplicities of symbol-objects. *Inf. Process. Lett.*, 100(3):124–129, 2006.
2. Artiom Alhazov and Rudolf Freund. P systems with toxic objects. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 99–125. Springer, 2014.
3. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Length P systems. *Fundam. Inform.*, 134(1-2):17–37, 2014.
4. Artiom Alhazov, Rudolf Freund, and Agustin Riscos-Núñez. Membrane division, restricted membrane creation and object complexity in P systems. *Int. J. Comput. Math.*, 83(7):529–547, 2006.
5. Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory.* Springer, 1989.
6. Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2):251–266, 2005.
7. Rudolf Freund and Petr Sosík. On the power of catalytic P systems with one catalyst. In Grzegorz Rozenberg, Arto Salomaa, José M. Sempere, and Claudio Zandron, editors, *Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers*, volume 9504 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2015.
8. Marvin L. Minsky. *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
9. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
10. Gheorghe Păun. *Membrane Computing: An Introduction.* Springer, 2002.

11. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
12. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages.* Springer, 1997.
13. The P Systems Website. `http://ppage.psystems.eu/`.

Contour Approximation with P Systems

Rodica Ceterchi[1], David Orellana-Martín[2], and Gexiang Zhang[3]

[1] University of Bucharest
Faculty of Mathematics and Computer Science
14 Academiei St, 010014 Bucharest, Romania
[2] Research Group on Natural Computing
Dept. of Computer Science and Artificial Intelligence
Universidad de Sevilla
E.T.S.I Informática, Avda. Reina Mercedes s/n 41012, Sevilla, Spain
[3] Research Center for Artificial Intelligence, Chengdu University of Technology,
Chengdu 610059 China

Summary. We model the problem of contour approximation using Hilbert's space filling curve, with a novel type of parallel array rewriting rules. We further use their pattern to introduce a special type of tissue P system, with novel features, among which is controlling their behavior with input. We propose some further developments.

1 Introduction

Space-filling curves (SFCs) were studied by mathematicians as a curiosity, since Peano discovered the first one in 1980 [7]. A year after this, Hilbert presented a much simpler curve [3]. Many properties and aspects of them were studied, and many other versions appeared in the literature, see for instance the monograph [8]. Lately, interesting applications to problems in Computer Science have been developed [1].

The finite approximations of the Hilbert curve can be described by words over a four letter alphabet $\{u, r, d, l\}$, letters which stand for the four directions in which a writing head can move in the lattice plane and draw a unit line. Formal language instruments have been used to describe families of SFC words.

In a series of papers we have studied the generation of such words with parallel rewriting controled by P systems, and we have proposed to model more complex applications of them. This short paper (rather a sketch) tries to accomplish this last purpose.

In the paper [2] we have proposed parallel array rewriting for the generation of Hilbert words. In Section 3 we modify the rules, introducing an external control, in order to generate contour approximations, after the ideas of [1] presented briefly in Section 2. Section 4 illustrates the generation of tissue P systems to model the rules.

In Section 5 a new variant of tissue P systems where in each transition step it can obtain inputs from external systems. Section 6 is devoted to a P system of the above variant capable of generating a contour approximation based on the Hilbert Curve.

Finally, in Section 7 we propose further developments of the ideas of this paper, and in **??** we indicate a possible development for crisis management.

2 Problem presentation

In his monograph [1], Bader proposes to use SFCs to maintain data about 2D objects. Figure 1 from his monograph illustrates the fact that, among several possible traversals of the quadtree associated to a 2D closed contour, a traversal based on the Hilbert SFC is more suited for applications which process the data in the quadtree nodes, since it has the locality property.

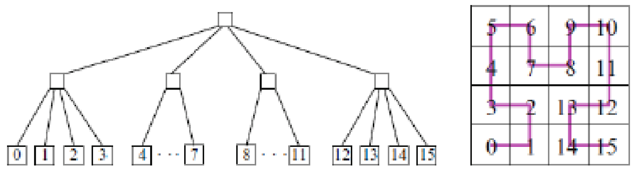


Fig. 1.5 Quadtree representation of a regular 4 × 4 grid, and a sequential order that avoids jumps

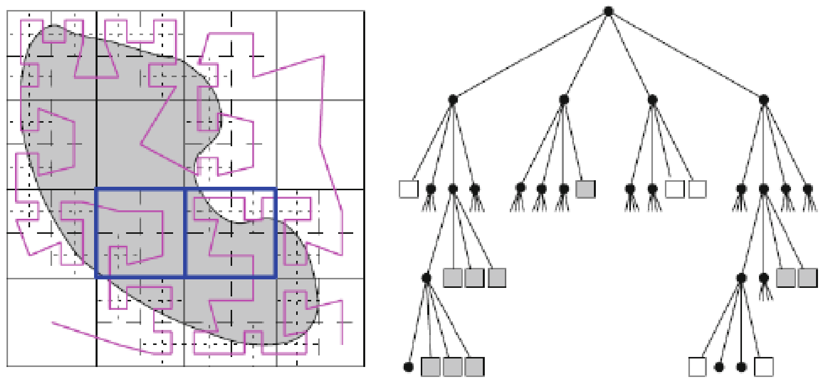


Fig. 1. Figures extracted from monograph [1] (Ch. 1, Pg. 6)

As Figure 2 illustrates, the approximation of the contour is obtained by pasting together pieces of the Hilbert curve, of different orders. Each piece of the Hilbert SFC is obtained in the usual manner, by repeated subdivisions of each sub-square where necessary, that is only for those sub-squares that cross the contour (border) of the 2D picture.

In the following, we propose to model the contour approximation generated by this method, first with arrays, next with P systems.

3 Contour generation with array rewriting

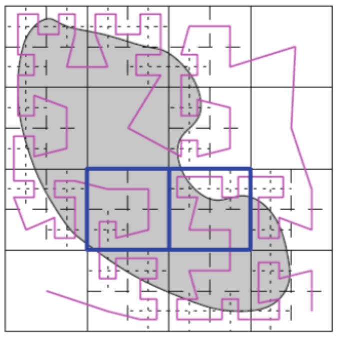


Fig. 2. The approximation of the contour is based on the pasted pieces of the Hilbert curve generated in the process [1]

Note: A combination of term rewriting rules with some constraints (called control) specifying the possible rewrite positions.

Consider the alphabet of non-terminals

$$\bar{N} = \{Ud, Ur, Ru, Rl, Ld, Lr, Du, Dl\},$$

where each element is a 1×1 array.

Denote by Γ the array morphism of the eight rewriting rules bellow:

$$U* \to \begin{matrix} Ur & Ud \\ Ru & L* \end{matrix} \qquad with\ * = d, r \tag{1}$$

$$R* \to \begin{matrix} D* & Rl \\ Ur & Ru \end{matrix} \qquad with\ * = u, l \tag{2}$$

$$L* \to \begin{matrix} Ld & Dl \\ Lr & U* \end{matrix} \qquad with\ * = d, r \tag{3}$$

$$D* \to \begin{matrix} R* & Ld \\ Du & Dl \end{matrix} \qquad with * = u, l \tag{4}$$

Denote by F the array morphism of the eight rewriting rules below:

$$Ud \to d, Ur \to r, Ld \to d, Lr \to r, Ru \to u, Rl \to l, Du \to u, Dl \to l \tag{5}$$

We have shown in [2] that $F(\Gamma^n(Ur))$ = the nth Hilbert word $H_n r$ in array representation.

We will now enlarge the set of non-terminals with one more symbol, #, standing for the blank array, and the set of Γ rules, as follows:

$$U^1* \to \begin{matrix} Ur & Ud \\ Ru & L* \end{matrix} \qquad with * = d, r \tag{6}$$

$$R^1* \to \begin{matrix} D* & Rl \\ Ur & Ru \end{matrix} \qquad with * = u, l \tag{7}$$

$$L^1* \to \begin{matrix} Ld & Dl \\ Lr & U* \end{matrix} \qquad with * = d, r \tag{8}$$

$$D^1* \to \begin{matrix} R* & Ld \\ Du & Dl \end{matrix} \qquad with * = u, l \tag{9}$$

$$U^0* = \#^0 \to \begin{matrix} \#^0 & \#^0 \\ \#^0 & \#^0 \end{matrix} = \#^1 \qquad with * = d, r \tag{10}$$

$$R^0* \to \#^1 \qquad with * = u, l \tag{11}$$

$$L^0* \to \#^1 \qquad with * = d, r \tag{12}$$

$$D^0* \to \#^1 \qquad with * = u, l \tag{13}$$

where $\#^0$ stands for the blank 1×1 array. Since we will have parallel rewriting rules, and we want to keep the growing dimension of the array, we will also have rules

$$\#^0 \to \begin{matrix} \#^0 & \#^0 \\ \#^0 & \#^0 \end{matrix} = \#^1 \tag{14}$$

which rewrite blanks to blanks. Two succesive applications of the rule above will produce $\#^2$, the 4×4 array filled with blanks, and so on.

(The notation $\#^n$ will be useful when we pass to the string representation.)

Each application of array rewriting rules (6)-(9) will correspond to a **division** of a square into four subsquares.

A division step will be followed by a **recognizer** step: each subsquare 'checks' whether it intersects the contour, in which case it gets a 1 superscript, or not, in which case it gets a 0 superscript, and will be rewritten accordingly at the next derivation.

Let us illustrate this with the case of the Figure 2 above:

$$U^1{*} \rightarrow \begin{matrix} Ur & Ud \\ Ru & L{*} \end{matrix} \rightsquigarrow \begin{matrix} U^1r & U^1d \\ R^1u & L^1{*} \end{matrix} \rightarrow \begin{matrix} Ur & Ud & Ur & Ud \\ Ru & Lr & Ru & Ld \\ Du & Rl & Ld & Dl \\ Ur & Ru & Lr & U{*} \end{matrix} \rightsquigarrow \begin{matrix} U^1r & U^1d & U^1r & U^0d \\ R^1u & L^0r & R^1u & L^0d \\ D^1u & R^1l & L^1d & D^1l \\ U^0r & R^1u & L^1r & U^1{*} \end{matrix} \rightarrow$$

$$\rightarrow \begin{matrix} Ur & Ud & Ur & Ud & Ur & Ud & \#^0 & \#^0 \\ Ru & Lr & Ru & Ld & Ru & Lr & \#^0 & \#^0 \\ Du & Rl & \#^0 & \#^0 & Du & Rl & \#^0 & \#^0 \\ Ur & Ru & \#^0 & \#^0 & Ur & Ru & \#^0 & \#^0 \\ Ru & Ld & Dl & Rl & Ld & Dl & Rl & Ld \\ Du & Dl & Ur & Ru & Lr & Ud & Du & Dl \\ \#^0 & \#^0 & Du & Rl & Ld & Dl & Ur & Ud \\ \#^0 & \#^0 & Ur & Ru & Lr & Ur & Ru & L{*} \end{matrix} \rightsquigarrow \begin{matrix} Ur & Ud & Ur & Ud & U^0r & U^0d & \#^0 & \#^0 \\ Ru & L^0r & R^0u & Ld & Ru & L^0r & \#^0 & \#^0 \\ Du & R^0l & \#^0 & \#^0 & Du & R^0l & \#^0 & \#^0 \\ Ur & R^0u & \#^0 & \#^0 & Ur & R^0u & \#^0 & \#^0 \\ Ru & Ld & D^0l & R^0l & Ld & Dl & Rl & L^0d \\ D^0u & Dl & Ur & R^0u & L^0r & U^0d & Du & D^0l \\ \#^0 & \#^0 & Du & Rl & L^0d & D^0l & Ur & U^0d \\ \#^0 & \#^0 & U^0r & Ru & Lr & Ur & Ru & L^0{*} \end{matrix} \rightarrow \cdots$$

After 3rd division-recognizer steps.
We generate the *nw* subsquare of the picture, illustrated below.
We have marked only the 0 superscript of non-terminals, for more clarity.

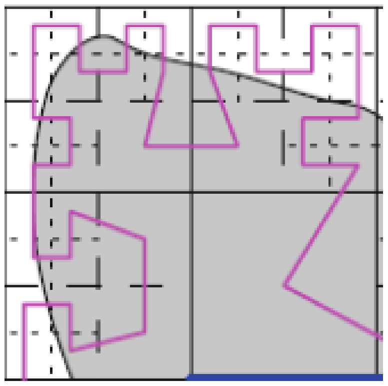


Fig. 3. Detail of the *nw* subsquare of the initial picture.

4 Membrane division rules

We will introduce dynamic P systems, which 'grow' by repeated membrane division rules, which will correspond to the subdivisions of the squares.

4.1 In string format

We have from [2] the linearization procedure which allows to pass from the 2D array representation to the linear one.

$$[U] \to [Ru]_{sw}[Ur]_{nw}[Ud]_{ne}[L*]_{se} \qquad [R] \to [Ur]_{sw}[Ru]_{se}[Rl]_{ne}[D]_{nw}$$

$$[L] \to [Dl]_{ne}[Ld]_{nw}[Lr]_{sw}[U]_{se} \qquad [D] \to [Ld]_{ne}[Dl]_{se}[Du]_{sw}[R]_{nw}$$

4.2 In 2D array format

Each nonterminal array is in a membrane which has the possibility of dividing itself into 4 membranes organized in a tissue manner.

The array rewriting rules become division rules for membranes, resulting in tissue P systems:

$$[U^1*] \to \begin{matrix} [Ur] & [Ud] \\ [Ru] & [L*] \end{matrix} \qquad with\ * = d, r \tag{15}$$

$$[R^1*] \to \begin{matrix} [D*] & [Rl] \\ [Ur] & [Ru] \end{matrix} \qquad with\ * = u, l \tag{16}$$

$$[L^1*] \to \begin{matrix} [Ld] & [Dl] \\ [Lr] & [U*] \end{matrix} \qquad with\ * = d, r \tag{17}$$

$$[D^1*] \to \begin{matrix} [R*] & [Ld] \\ [Du] & [Dl] \end{matrix} \qquad with\ * = u, l \tag{18}$$

An example of 3 succesive subdivisions, which will finally lead to the picture:

$$[U^1*] \to \begin{matrix} [Ur] & [Ud] \\ [Ru] & [L*] \end{matrix} \rightsquigarrow \begin{matrix} [U^1r] & [U^1d] \\ [R^1u] & [L^1*] \end{matrix} \to \begin{matrix} [Ur] & [Ud] & [Ur] & [Ud] \\ [Ru] & [Lr] & [Ru] & [Ld] \\ [Du] & [Rl] & [Ld] & [Dl] \\ [Ur] & [Ru] & [Lr] & [U*] \end{matrix} \rightsquigarrow \begin{matrix} [U^1r] & [U^1d] & [U^1r] & [U^0d] \\ [R^1u] & [L^0r] & [R^1u] & [L^0d] \\ [D^1u] & [R^1l] & [L^1d] & [D^1l] \\ [U^0r] & [R^1u] & [L^1r] & [U^1*] \end{matrix} \to$$

$$\to \begin{matrix} [Ur] & [Ud] & [Ur] & [Ud] & [Ur] & [Ud] & [\#^0] & [\#^0] \\ [Ru] & [Lr] & [Ru] & [Ld] & [Ru] & [Lr] & [\#^0] & [\#^0] \\ [Du] & [Rl] & [\#^0] & [\#^0] & [Du] & [Rl] & [\#^0] & [\#^0] \\ [Ur] & [Ru] & [\#^0] & [\#^0] & [Ur] & [Ru] & [\#^0] & [\#^0] \\ [Ru] & [Ld] & [Dl] & [Rl] & [Ld] & [Dl] & [Rl] & [Ld] \\ [Du] & [Dl] & [Ur] & [Ru] & [Lr] & [Ud] & [Du] & [Dl] \\ [\#^0] & [\#^0] & [Du] & [Rl] & [Ld] & [Dl] & [Ur] & [Ud] \\ [\#^0] & [\#^0] & [Ur] & [Ru] & [Lr] & [Ur] & [Ru] & [L*] \end{matrix} \rightsquigarrow \begin{matrix} [Ur] & [Ud] & [Ur] & [Ud] & [U^0r] & [U^0d] & [\#^0] & [\#^0] \\ [Ru] & [L^0r] & R^0u & Ld & Ru & L^0r & \#^0 & \#^0 \\ [Du] & [R^0l] & [\#^0] & [\#^0] & [Du] & [R^0l] & [\#^0] & [\#^0] \\ [Ur] & [R^0u] & [\#^0] & [\#^0] & [Ur] & [R^0u] & [\#^0] & [\#^0] \\ [Ru] & [Ld] & [D^0l] & [R^0l] & [Ld] & [Dl] & [Rl] & [L^0d] \\ [D^0u] & [Dl] & [Ur] & [R^0u] & [L^0r] & [U^0d] & [Du] & [D^0l] \\ [\#^0] & [\#^0] & [Du] & [Rl] & [L^0d] & [D^0l] & [Ur] & [U^0d] \\ [\#^0] & [\#^0] & [U^0r] & [Ru] & [Lr] & [Ur] & [Ru] & [L^0*] \end{matrix} \to \cdots$$

With labels on membranes:

$$[U] \to [Ru]_{sw}[Ur]_{nw}[Ud]_{ne}[L*]_{se} \qquad [R] \to [Ur]_{sw}[Ru]_{se}[Rl]_{ne}[D]_{nw}$$

$$[L] \to [Dl]_{ne}[Ld]_{nw}[Lr]_{sw}[U]_{se} \qquad [D] \to [Ld]_{ne}[Dl]_{se}[Du]_{sw}[R]_{nw}$$

$$[U^1*] \to \begin{matrix}[Ur]_{nw} & [Ud]_{ne} \\ [Ru]_{sw} & [L*]_{se}\end{matrix} \qquad with\ * = d, r \tag{19}$$

$$[R^1*] \to \begin{matrix}[D*]_{nw} & [Rl]_{ne} \\ [Ur]_{sw} & [Ru]_{se}\end{matrix} \qquad with\ * = u, l \tag{20}$$

$$[L^1*] \to \begin{matrix}[Ld]_{nw} & [Dl]_{ne} \\ [Lr]_{sw} & [U*]_{se}\end{matrix} \qquad with\ * = d, r \tag{21}$$

$$[D^1*] \to \begin{matrix}[R*]_{nw} & [Ld]_{ne} \\ [Du]_{sw} & [Dl]_{se}\end{matrix} \qquad with\ * = u, l \tag{22}$$

4.3 Labels for membranes, and memory

In the above we have used labels $\{sw, nw, ne, se\}$ standing for the obvious notation for corners of a square: southwest, northwest, etc.

Of course, binary labels could be used instead, with interesting properties. For instance:

$$sw = 00,\ nw = 01,\ ne = 11,\ se = 10.$$

This has the property that any 2 adjacent squares have labels differing in only 1 bit (Gray code on 2 bits). Many binary codes can be associated to SFCs.

We will **concatenate (properly!) labels at every derivation step**, such that each membrane: on one hand inherits the label of its 'parent', and gets a label stating what 'son' it is. In this way, membranes have **memory**. Division rules (with labels) will be of the form:

$$[\ ]_{\alpha} \to [\ ]_{\alpha 00}\ [\ ]_{\alpha 01}\ [\ ]_{\alpha 11}\ [\ ]_{\alpha 10}$$

5 Tissue P systems with evolutional communication rules, extended division rules and external inputs

Definition 1. *Let $\Pi = (\Gamma, H, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, \mathcal{I})$ be a tissue P system with evolutional communication rules, extended division and r external inputs rules of degree q, where:*

1. *Γ is a finite alphabet;*
2. *H is the set of labels $\{1, \ldots, q\}$;*
3. *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are multisets over Γ;*
4. *$\mathcal{R}$ is the set of rules of the following forms:*
   a) $[\,u\,]_{h_1}\,[\,v\,]_{h_2} \to [\,v'\,]_{h_1}\,[\,u'\,]_{h_2}, h_1, h_2 \in H, u, v, u', v' \in M_f(\Gamma), |u| + |v| > 0, |u| = 0 \to |u'| = 0, |v| = 0 \to |v'| = 0$ (evolutional communication rules);

b) $[\,a\,]_h \to [\,a_1\,]_{h_1} \dots [\,a_s\,]_{h_s}, h, h_1, \dots, h_s \in H, a, a_1, \dots, a_s \in \Gamma$ *(extended division rules);*

5. $\mathcal{I}$ *be a set of elements* $(\Gamma_i, H_i, \mathcal{R}_i), 1 \le i \le r$ *such that:*
   a) $\Gamma_i \subseteq \Gamma$*;*
   b) $H_i \cap H = \emptyset \wedge H_i \cap H_j = \emptyset, i \neq j$*.*
   c) $\mathcal{R}_i$ *is the set of rules of the following form:*
      i. $[\,u\,]_{h_1}\,[\,v\,]_{h_2} \to [\quad]_{h_1}\,[\,u'\,]_{h_2}, h_1 \in H_i, h_2 \in H, u \in M_f(\Gamma_i), v, u' \in M_f(\Gamma), |u| > 0$ *(evolutional communication rules);*

A tissue P system with communication rules, extended division rules and r inputs

$$\Pi = (\Gamma, H, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, \mathcal{I})$$

of degree q can be viewed as a set of q cells such that $\mathcal{M}_1, \dots, \mathcal{M}_q$ represent the multisets of objects initially placed in the q cells of the system.

A rule of the type $[\,u\,]_{h_1}\,[\,v\,]_{h_2} \to [\,v'\,]_{h_1}\,[\,u'\,]_{h_2}$ is called an *evolutional communication rule*. A rule of the type $[\,a\,]_h \to [\,a_1\,]_{h_1} \dots [\,a_s\,]_{h_s}$ is called an *extended division rule*. The length of evolutional communication rules is defined by $|u| + |v| + |u'| + |v'|$. The length of extended division rules is defined by $s + 1$. These rules were introduced in [9], and more deeply investigated in [4, 5, 6]

An *instantaneous description* or a *configuration* at an instant t of a tissue P system with evolutional communication rules and extended division rules is described by the cells present and the corresponding multisets of objects over Γ associated with all the cells present in the system (not in the inputs). The *initial configuration* is $((1, \mathcal{M}_1), \dots, (q, \mathcal{M}_q))$.

A rule $[\,u\,]_{h_1}\,[\,v\,]_{h_2} \to [\,v'\,]_{h_1}\,[\,u'\,]_{h_2}, h_2 \in H$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if there exist a cell labelled by h_1 containing the multiset u and a cell labelled by h_2 containing the multiset v. When applying such a rule, the objects specified by u and v disappear from their respective cells and multisets v' and u' appear in h_1 and h_2, respectively. If $|u| = 0$ (respectively, $|v| = 0$), then $|u'| = 0$ (resp., $|v'| = 0$) must be satisfied (this would correspond to symport rules). If $h_2 \in H_1 \cup \dots \cup H_r$, the rule is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if there exist a cell labelled by h_1 containing the multiset u and the cell of the external input i such that $h_2 \in H_i$ contains the multiset v. The behaviour of the application of the rule is similar to when $h_2 \in H$.

A rule $[\,a\,]_h \to [\,a_1\,]_{h_1} \dots [\,a_s\,]_{h_s}$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t if there exists a cell labelled by h containing an object a. When applying such a rule, the cell h is divided in s new cells labelled by $h_i (1 \le i \le s)$, where a is changed to a_i in the corresponding cell and the rest of the contents is replicated in each cell.

We can think that the external inputs are independent systems that are computing a function. In each computational step, they will have different contents, that will be stated when the system is defined. In this sense, the contents of each cell of the system has to be defined for every configuration.

The rules from $\mathcal{R}$ of a tissue P system with evolutional communication rules, extended division rules and external inputs are applied in a non-deterministic maximally parallel manner (at each step we apply a multiset of rules which is maximal; that is, no further applicable rule can be added), with the following important remark: if a cell is divided, then the division rule is the only one which is applied to that cell at that step; that is, extended division rules interrupts the communication of that cell with others in that step. The new cells resulting from division will be able to interact with other cells from the next step.

Let us fix a tissue P system with evolutional communication rules, extended division rules and r inputs Π. We say that configuration $\mathcal{C}_t$ yields configuration $\mathcal{C}_{t+1}$ in one *transition step*, denoted by $\mathcal{C}_t \Rightarrow_\Pi \mathcal{C}_{t+1}$ if we can pass from $\mathcal{C}_t$ to $\mathcal{C}_{t+1}$ by applying the rules from $\mathcal{R}$ as follows: A transition step is divided in two micro-steps.

1. First, rules from $\mathcal{R}_i, 1 \leq i \leq r$ are applied in a maximally parallel and non-deterministic way. The "input systems" cannot receive any new contents from the main system. This first step is denoted as $\mathcal{C}_t \rightsquigarrow \mathcal{C}'_t$;
2. Second, rules from $\mathcal{R}$ are applied as stated above. This is denoted as $\mathcal{C}'_t \rightarrow \mathcal{C}_{t+1}$;

We say that a *transition step* $\mathcal{C}_t \Rightarrow_\Pi \mathcal{C}_{t+1}$ is a transition $\mathcal{C}_t \rightsquigarrow \mathcal{C}'_t \rightarrow \mathcal{C}_{t+1}$.

A *computation* of Π is a (finite or infinite) sequence of configurations such that:

1. the first term of the sequence is the initial configuration of the system;
2. each non-initial configuration of the sequence is obtained from the previous configuration by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and
3. if the sequence is finite (called *halting computation*) then the last term of the sequence is a *halting configuration* (a configuration where no rule of the system is applicable to it).

All computations start from an initial configuration and proceed as stated above.

If $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_p)$ of Π ($p \in \mathbb{N}$) is a halting computation, then the *length* of $\mathcal{C}$, denoted by $|\mathcal{C}|$ is p; that is, $|\mathcal{C}|$ is the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. We denote by $\mathcal{C}_t(i), i \in H$, the multiset of objects over Γ contained in all membranes labelled by i (by applying extended division rules different membranes with the same label can be created) at configuration $\mathcal{C}_t$. We denote $\mathcal{C}^*_t$ the multiset $\Sigma_{h \in H} \mathcal{C}_t(h)$

6 Generating contour approximations with P systems

We will use a P system of the type introduced in Section 5. It interacts with a 2D picture with contour as described by Figure 4

Let n be the number of iterations of the Hilbert curve we want to describe, let $L = \{00, 01, 10, 11\}^n$ the set of all words of length at most $2n$ over $\{00, 01, 10, 11\}$ and $\overline{N} = \{Ud, Ur, Ru, Rl, Ld, Lr, Du, Dl\}$. We consider the tissue P system

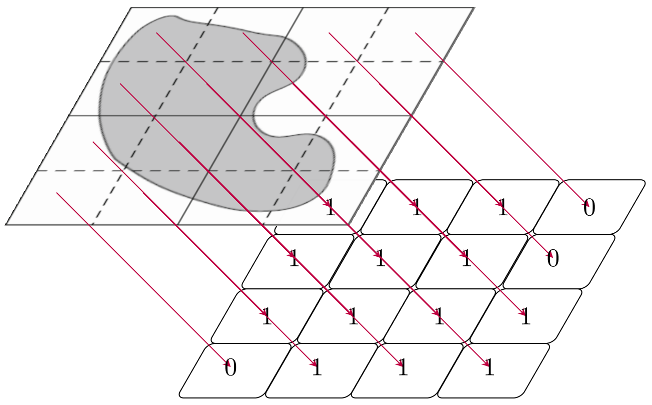


Fig. 4. In each step, the external input system *pic* interacts with Π in such a way that a symbol 0 or 1 is sent to the corresponding cell in the system.

$$\Pi = (\Gamma, H, \mathcal{M}_\lambda, \mathcal{R}, \mathcal{I})$$

with evolutional communication rules, extended division rules and 1 external input defined as follows:

1. Working alphabet: $\Gamma = \overline{N} \cup \{U^\alpha d, U^\alpha r, R^\alpha u, R^\alpha l, L^\alpha d, L^\alpha r, D^\alpha u, D^\alpha l \mid \alpha \in \{0,1\}\} \cup \{0,1\}$, being λ the empty string;
2. $H = \{00, 01, 10, 11\}^*$ (that is, the set of all words over $\{00, 01, 10, 11\}$) is the set of labels;
3. $\mathcal{M}_\lambda = \{U^1 r\}$
4. The set $\mathcal{R}$ consists of the following rules:
   a) Rules to divide the cells with intersections:
   $[U^1 d]_h \to [Ru]_{h00}\ [Ur]_{h01}\ [Ud]_{h11}\ [Ld]_{h10}$
   $[U^1 r]_h \to [Ru]_{h00}\ [Ur]_{h01}\ [Ud]_{h11}\ [Lr]_{h10}$
   $[R^1 u]_h \to [Ur]_{h00}\ [Du]_{h01}\ [Rl]_{h11}\ [Ru]_{h10}$
   $[R^1 l]_h \to [Ur]_{h00}\ [Dl]_{h01}\ [Rl]_{h11}\ [Ru]_{h10}$
   $[L^1 d]_h \to [Lr]_{h00}\ [Ld]_{h01}\ [Dl]_{h11}\ [Ud]_{h10}$
   $[L^1 r]_h \to [Lr]_{h00}\ [Ld]_{h01}\ [Dl]_{h11}\ [Ur]_{h10}$
   $[D^1 u]_h \to [Du]_{h00}\ [Ru]_{h01}\ [Ld]_{h11}\ [Dl]_{h10}$
   $[D^1 l]_h \to [Du]_{h00}\ [Rl]_{h01}\ [Ld]_{h11}\ [Dl]_{h10}$
   b) Rules to divide the cells without intersections:

$[U^0d]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[U^0r]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[R^0u]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[R^0l]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[L^0d]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[L^0r]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[D^0u]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[D^0l]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$
$[\#^0]_h \rightarrow [\#]_{h00}\ [\#]_{h01}\ [\#]_{h11}\ [\#]_{h10}$

5. $I = (\Gamma_{pic}, H_{pic}, \mathcal{R}_{pic})$, where:
   a) $\Gamma_{pic} = \{0, 1\}$
   b) H_{pic} is the set of the elements from H but with superscript *pic*.
   c) The set $\mathcal{R}_{pic}$ consists of the following rules:
      i. Rules to communicate if there is an interesection in a specific area:

$$\left.\begin{array}{l}
[a]_h^{pic}\ [Ud]_h \rightarrow [\ \ ]_h^{pic}\ [U^a d]_h \\
[a]_h^{pic}\ [Ur]_h \rightarrow [\ \ ]_h^{pic}\ [U^a r]_h \\
[a]_h^{pic}\ [Ru]_h \rightarrow [\ \ ]_h^{pic}\ [R^a u]_h \\
[a]_h^{pic}\ [Rl]_h \rightarrow [\ \ ]_h^{pic}\ [R^a l]_h \\
[a]_h^{pic}\ [Ld]_h \rightarrow [\ \ ]_h^{pic}\ [L^a d]_h \\
[a]_h^{pic}\ [Lr]_h \rightarrow [\ \ ]_h^{pic}\ [L^a r]_h \\
[a]_h^{pic}\ [Du]_h \rightarrow [\ \ ]_h^{pic}\ [D^a u]_h \\
[a]_h^{pic}\ [Dl]_h \rightarrow [\ \ ]_h^{pic}\ [D^a l]_h
\end{array}\right\} for\ a \in \{0,1\}, h \in H$$

$$[0]_h^{pic}\ [\#]_h \rightarrow [\ \ ]_h^{pic}\ [\#^0]_h$$

   d) The contents of a cell in this system will be 0 if there is no intersection in the corresponding area of the picture, and 1 otherwise. In each configuration there will exist the cells corresponding to the resolution of the system.

7 Conclusions, Open problems, Suggestions for further developments

The present paper proposes a new variant of parallel array rewriting rules, capable to generate approximations of irregular contours, based on conecting pieces of Hilbert words of different 'resolutions'.

It proposes also a new variant of tissue P systems with evolutional communication rules, extended division rules and external inputs. In this variant, division rules are allowed to change the labels of the new created cells. The capability of receive input from an external source allows these systems to get more precision of a picture in each transition step.

Further developments are possible along several lines.

- to find means of effectively representing in a graphical manner the entire approximation, the problem being the segments which connect pieces of the SFC;

- other variants of P systems and refinements of the proposed one
- use the external input as a catalyst to allow or forbid the system to evolve.
- making use of the array representation in string format;
- taking into account the versatility of P systems, to use this small model as a template for complex applications, which involve the manipulation of spatial data; an example would be looking for applications in robotics (global path planning).

Acknowledgements

The present paper is work in progress.

It was supported in part by the research project TIN2017-89842-P, cofinanced by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

The work of GZ was supported by the National Natural Science Foundation of China (61972324, 61672437, 61702428), by Beijing Advanced Innovation Center for Intelligent Robots and Systems (2019IRS14) and Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06).

On a more personal note, RC would like to thank Luis Valencia-Cabrera for asking a very interesting question.

References

1. M. Bader. Space-filling Curves - An Introduction with applications in Scientific Computing. Texts in Computational Science and Engineering. Springer-Verlag (2013).
2. R. Ceterchi, L. Zhang, L. Pan, K. G. Subramanian, G. Zhang. Generating Hilbert Words in Array Representation with P Systems, presented at ACMC2019, 14-16 November 2019, Xiamen, China (submitted)
3. D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. Math. Annln. 38 459–460 (1891).
4. D. Orellana-Martín, L. Valencia-Cabrera, B. Song, L. Pan, M.J. Pérez-Jiménez. Tuning frontiers of efficiency in tissue P systems with evolutional communication rules. *Complexity*, to be published.
5. D. Orellana-Martín, L. Valencia-Cabrera, B. Song, L. Pan, M.J. Pérez-Jiménez. Narrowing Frontiers with Evolutional Communication rules and Cell Separation. In D. Orellana, Gh. Păun, A. Riscos, L. Valencia (eds.), *Proceedings of the Sixteenth Brainstorming Week on Membrane Computing*, Sevilla, Spain, January 30 - February 2 2018, pp. 123-162.
6. L. Pan, B. Song, L. Valencia-Cabrera, M.J. Pérez-Jiménez. The Computational Complexity of Tissue P Systems with Evolutional Symport/Antiport Rules. *Complexity*, vol. 2018, Article ID 3745210, 21 pages, 2018 (10.1155/2018/3745210).
7. G. Peano. Sur une courbe qui remplit toute une aire plane. Math. Annln. 36 157–160 (1890).

8. H. Sagan. Space-filling curves. Springer-Verlag. New York. 1994.
9. B. Song, C. Zhang, L. Pan. Tissue-like P systems with evolutional symport/antiport rules. *Information Sciences*, **378** (2017), 177-193 (`doi: 10.1016/j.ins.2016.10.046`).

P Colonies and Reaction Systems

Lucie Ciencialová[1], Luděk Cienciala[1], and Erzsébet Csuhaj-Varjú[2]

[1] Institute of Computer Science, Silesian University in Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz
ludek.cienciala@fpf.slu.cz

[2] Department of Algorithms and Their Applications, Faculty of Informatics
Eötvös Loránd University
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary
csuhaj@inf.elte.hu

Summary. P colonies are abstract computing devices modeling communities of very simple reactive agents living and acting in a joint shared environment which is given with a multiset of objects. Reaction systems were proposed as a computing device, components of which represent basic chemical reactions that take place in shared environment given with a set. Although P colonies operate with multisets of objects and reaction systems work with sets, the two models can be related. In this paper, we construct a P colony simulating interactive processes in a reaction system.

P colonies were introduced in [7] as a variant of very simple membrane systems (P systems), inspired by so-called colonies of formal grammars. (For more information on P systems the interested reader is referred to [9], for P colonies to [1], and for grammatical model colony to [8].) A P colony is formed from agents and their shared environment. Each agent is represented by a multiset of objects in a membrane and the environment is given by multiset of objects as well. Agents are equipped with programs composed from rules, the rules are applied to (multisets of) objects. The number of objects inside each agent is set by definition and it is usually a very small number - 1, 2 or 3. The environment of the P colony is used as communication channel for agents. Through the environment, the agents are able to affect the behavior of another agent.

The rules of the agents can be rewriting, communication or checking rules; these three rule types were introduced in [7]. Rewriting rule $a \to b$ allows agent to rewrite (evolve) one object a to object b. Both objects are placed inside the agent. Communication rule $a \leftrightarrow b$ provides the possibility to exchange object c placed inside the agent and object d in the environment. A checking rule is formed of two rules r_1, r_2 which are of type rewriting or communication. Checking feature sets some kind of priority between rules r_1, r_2. The agent tries to apply the first rule and if it cannot be performed, the agent executes the second rule. The agents of P colonies work in a maximally parallel manner, i.e., a maximal number of enabled agent performs one of its applicable program in parallel. An agent is enabled in a computation step if it is able to apply one of its programs. If an agent is not enabled, then its objects remain unchanged. For detailed information on operation of P colonies, see [1].

Reaction systems (R systems, for short) are computational models of another type. The model was introduced in [4] as a computing device, components of which are a simile for basic chemical reactions. Roughly speaking, a reaction system is composed of a finite set of objects that can be considered as chemicals and a finite set of reactions. Each reaction is a triplet of sets: reactants, inhibitors and products. Let T be a set of reactants. A reaction is applied if all reactants are present in T, and there are no inhibitors; then reactants are replaced by the products. All enabled reactions are applied in parallel. The final set of products is the union of all single sets of products of each reaction which is enabled in T. The reader might notice that a reaction can also be considered as an action of an agent.

It is easy to observe that P colonies and R systems have similarities: both of them can be seen as multi-agent systems of very simple reactive agents. However, there are significant differences between them. Namely, P colonies work with finite multisets of objects and R systems operate with finite sets. Furthermore, in case of P colonies, those objects that do not take part in any action at a computation step, remain unchanged, but in case of R systems disappear from the available objects.

It is an intriguing question whether or not P colonies and R systems can be related. In this paper, we focus on construction of P colony that can simulate interactive processes in given reaction system. The paper is structured as follows: The second section is devoted to definitions and notations used in the paper. The third section contains construction of P colony and the paper concludes with possibilities of future work.

1 Definitions

Throughout the paper we assume the reader to be familiar with basics of formal language theory. We introduce notions and notations used in the sequel.

We use $\mathbb{N}\cdot\mathsf{RE}$ to denote the family of recursively enumerable sets of natural numbers and $\mathbb{N}$ to denote the set of natural numbers.

Σ is a notation for the alphabet. Let Σ^* be set of all words over alphabet Σ (including empty word ε). For the length of the word $w \in \Sigma^*$ we use notation $|w|$ and the number of occurrences of symbol $a \in \Sigma$ in w is denoted by $|w|_a$.

A multiset of objects M is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : V \to N$; f assigns to each object in V its multiplicity in M. The set of all multisets over the set of objects V is denoted by V^*. The set V' is called the support of M and denoted by $supp(M)$ if for all $x \in V'$ $f(x) \neq 0$. The cardinality of M, denoted by $card(M)$, is defined by $card(M) = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects

$V = \{a_i, \dots\ a_n\}$ can be represented as a string w over alphabet V with $|w|_{a_i} = f(a_i);\ 1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent M, and ε represents the empty multiset.

1.1 P Colonies

The original concept of a P colony was introduced in [7] and presented in a developed form in [6, 2].

Definition 1. *A P colony of capacity k, $k \geq 1$, is a construct*
$$\Pi = (V, e, f, v_E, B_1, \dots, B_n), \text{ where}$$

- *V is an alphabet, its elements are called objects;*
- *$e \in V$ is the basic (or environmental) object of the colony;*
- *$f \in V$ is the final object of the colony;*
- *v_E is a finite multiset over $A - \{e\}$, called the initial state (or initial content) of the environment;*
- *B_i, $1 \leq i \leq n$, are agents, where each agent $B_i = (o_i, P_i)$ is defined as follows:*
  - *o_i is a multiset over V consisting of k objects, the initial state (or the initial content) of the agent;*
  - *$P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is a finite set of programs, where each program consists of k rules, which are in one of the following forms each:*
    - *$a \to b$, $a, b \in V$, called an evolution rule;*
    - *$c \leftrightarrow d$, $c, d \in V$, called a communication rule;*
    - *r_1/r_2, called a checking rule; r_1, r_2 are both evolution rules or both communication rules.*

We add some brief explanations to the components of the P colony.

The first type of rules associated to the programs of the agents, the *evolution rules*, are of the form $a \to b$. This means that object a inside the agent is rewritten to (evolved to be) object b.

The second type of rules, the *communication rules*, are of the form $c \leftrightarrow d$. If a communication rule is performed, then object c inside the agent and object d in the environment swap their location. Thus, after executing the rule, object d appears inside the agent and object c is located in the environment.

The third type of rules are the checking rules. A checking rule is formed from two rules of one of the two previous types. If a checking rule r_1/r_2 is performed, then the rule r_1 has higher priority to be executed over the rule r_2. This means that the agent checks whether or not rule r_1 is applicable. If the rule can be executed, then the agent must use this rule. If rule r_1 cannot be applied, then the agent uses rule r_2.

We note that these types of rules are the basic ones; in some variants of P colonies other types of rules have been also considered.

The program determines the activity of the agent: the agent can change its state and/or the state of the environment.

The environment is represented by a finite number (zero included) of copies of non-environmental objects and a countably infinite copies of the environmental object e.

In every step, each object inside an agent is affected by the execution of a program. Depending on the rules in the program, the program execution may affect the environment as well. This interaction between the agents and the environment is the key factor of the functioning of the P colony.

An initial configuration of P colony is $(n+1)$-tuple $(o_1, \dots, o_n, v_E)$ of the multisets of objects placed in P colony at the beginning of the computation, where o_i ($1 \leq i \leq n$) is the content of the agent B_i and v_E is the multiset of object in the environment different from e. In general, the configuration of the P colony Π is defined as $(n+1)$-tuple $(w_1, \dots, w_n, w_E)$, where w_i represents all objects inside of i-th agent, $|w_i| = c$, $1 \leq i \leq n$, $w_E \in (V - \{e\})^*$ is composed by objects different from e placed in the environment.

At each step of the (parallel) computation every agent attempts to find one of its programs to use. If the number of applicable programs is higher than one, the agent non-deterministically chooses one of them. At one step of computation, the maximal possible number of agents have to be active, i.e., have to perform a program.

By applying programs, the P colony passes from one configuration to another configuration. A sequence of configurations started from the initial configuration is called a computation. A configuration is halting if the P colony has no applicable program. With halting computation the result is associated and it is the number of copies of final object placed in the environment in halting configuration.

$$N(\Pi) = \{|w_E|_f \mid (o_1, \dots, o_n, v_E) \Rightarrow^* (w_1, \dots, w_n, w_E)\},$$

where $(o_1, \dots, o_n, v_E)$ is the initial configuration, $(w_1, \dots, w_n, w_E)$ is the final configuration, and $\Rightarrow^*$ denotes reflexive and transitive closure of $\Rightarrow$.

The number of agents in a given P colony is called the degree of Π; the maximal number of programs of an agent of Π is called the height of Π and the number of the objects inside an agent is the capacity of Π. The family of all sets of numbers $N(\Pi)$ computed as above by P colonies of capacity at most $c \geq 0$, degree at most $n \geq 0$ and height at most $h \geq 0$, using checking programs, and working in the sequential mode is denoted by $NPCOL_{seq}K(c,n,h)$; whereas the corresponding families of P colonies working in the maximally parallel way are denoted by $NPCOL_{par}K(c,n,h)$. If one of the parameters n or h is not bounded, then we replace it with $*$. If only P colonies using programs without checking rules are considered, then we omit the K. If numerical parameter is unbounded, we denote it by a $*$.

1.2 Reaction Systems

In the following we briefly summarize the basic notions concerning reaction systems (R systems), introduced in [4].

Let S be an alphabet (its elements are called molecules, or only symbols).

Definition 2. *A reaction is a triple $a = (R, I, P)$ such that R, I, P are finite nonempty sets with $R \cap I = \emptyset$.*

R is the reactant set of a,I is the inhibitor set of a, and P is the product set of a; R, I, P are also denoted as R_a, I_a, P_a, respectively. We denote by $rac(S)$ the set of all reactions in S. The set S is usually called background set. In some papers the definition is altered in such a way that set of inhibitors can be empty set.

Definition 3. *A reaction system is an ordered pair $\mathcal{A} = (S, A)$, where S is a background set and A is a nonempty finite subset of $rac(S)$.*

To describe the effect of a set of reactions on a state, we first define the effect of a single reaction.

Definition 4. *Let S be a background set, let $X \subseteq S$, and let $a \in rac(S)$. Then a is enabled by X, denoted by $en_a(X)$, if $R_a \subseteq X$ and $I_a \cap X = \emptyset$. The result of a on X, denoted by $res_a(X)$, is defined by $res_a(X) = P_a$ if $en_a(X)$, and $res_a(X) = \emptyset$. otherwise.*

The effect of a set of reactions on a state is cumulative, defined as follows.

Definition 5. *Let S be a background set, let $X \subseteq S$, and let $A \subseteq rac(S)$. The set of reactions enabled by X is denoted by $en(A, X)$ and it is defined by $en(A, X) = \{a \in A \mid en_a(X)\}$. The result of A on X, denoted by $res(A, X)$, is defined by $res(A, X) = \{res_a(X) \mid a \in A\}$. The set of reactions that can generate X, denoted by $prod(A, X)$, is defined as $prod(A, X) = \{a \in A \mid P_a \subseteq X\}$.*

The dynamic behavior of the reaction systems is captured by the notion of an interactive process.

Definition 6. *Let $\mathcal{A} = (S, A)$ be a reaction system. An interactive process in $\mathcal{A}$ is a pair $\pi = (\gamma, \varphi)$ of finite sequences such that $\gamma = C_0, C_1, \ldots, C_{n-1}, \varphi = D_1, \ldots, D_n$ with $n \geq 1$, where $C_0, \ldots, C_{n-1}, D_1, \ldots, D_n \subseteq S$, $D_1 = res(A, C_0)$, and $D_i = res(A, D_{i-1} \cup C_{i-1})$ for each $2 \leq i \leq n$.*

The sequences $C_0, \ldots, C_{n-1}$ and $D_1, \ldots, D_n$ are the context and result sequences of π, respectively. Context C_0 represents the initial state of π (the state in which the interactive process is initiated), and the contexts $C_1, \ldots, C_{n-1}$ represent the influence of the environment to the computation. It should be noticed that the context sequence $\gamma = C_0, C_1, \ldots, C_{n-1}$ is described by a regular expression over S. The sequence $sts(\pi) = W_0, \ldots, W_n$ denotes the state sequence of π, where $W_0 = C_0$ (the initial state), and $W_i = D_i \cup C_i$ for all $1 \leq i \leq n$. The sequence $act(\pi) = E_0, \ldots, E_{n-1}$ of subsets of A such that $E_i = en(A, W_i)$ for all $0 \leq i \leq n-1$ represents the activity sequence of π. Thus, the evolution can be written as

$$W_0 \xrightarrow{E_0} W_1 \xrightarrow{E_1} \cdots \xrightarrow{E_{n-1}} W_n.$$

If $E_n = en(A, W_n) = \emptyset$ then interactive process terminates.

One step of evolution – the evolution from state W_i to state W_{i+1} can be seen as transition mapping $\delta : 2^S \times 2^S \to 2^S$ defined as

$$\delta(D_i, C_i) = D_{i+1}$$

iff there exists (just one) set $E_i \subseteq A$ such that $E_i = en(A, D_i \cup C_i)$ and $D_{i+1} = res(E_i)$. For the first step of an evolution, there is transition mapping defined as $\delta(\emptyset, C_0) = D_1$.

In some sources the set of inhibitors can be empty.

2 Dynamical Behavior: P Colonies versus R Systems

Let us examine the dynamical changes of the environment of a P colony in the course of the computation. From this point of view we can find some similarities between P colonies and R systems. For example, it is easy to see that the change of the support of the environment (i.e. the support of the finite multiset of non-enviromental symbols forming the environment), resembles to a reaction or several reactions performed by a reaction system. The agents of the P colony can exchange object(s) with the environment in a way similar to actions of reactions in R systems. The exchange rule of the P colony must be enabled - reactants must be included in the environment - and then products will occur in the environment after performing reaction. We can also find some differences in behavior of these two computing devices. The first difference is in the number of active components. In R systems, all enabled reactions are executed while in P colonies, the number of active agents is limited to the number of objects that are placed in the environment. Furthermore, objects not used in any action of the P colony remain unchanged and available for further steps, but in case of reaction systems these objects disappear from the system. One other significant difference between P colony and R system is that P colony operates with finite multisets and R system with finite sets of objects.

In the following we construct a P colony which simulates the behavior of an R system. Notice that a set as a notion is different from a multiset of objects where each object appears only in (at most) one copy, however for our purposes it is enough to define a P colony where objects of certain type appear in the environment only in at most one copy during operation.

Thus, for given R system $\mathcal{A} = (S, A)$ and sequence of inputs $i_0, i_1, \ldots, i_n$ we can construct P colony $\Pi = (V, e, f, v_E, B_1, \ldots, B_n)$ that simulates interactive processes of $\mathcal{A}$.

Instead of the formal statement and its proof, we provide the description of the simulation steps and the main parts of the construction. In addition, we develop an example that helps in the easier understanding.

One step of an interactive process in $\mathcal{A}$ is simulated in five phases in Π:

1. Input Generation
2. Input multiplication
3. Simulation of reactions
4. Consuming phase

For each phase, we construct subset of agents that are responsible of executing corresponding phase. Let k be the maximum of the number of reactants, l the maximum of the number of inhibitors, and m the maximum number of products associated with one reaction of the reactions given by R system $\mathcal{A}$.

1. Generate Input

In this phase the objects (symbols) in the input set are generated. For this phase, we construct agents called i-agents. They generate input symbols in one step. Obviously, the number of i-agents is $|S|$. After generation of current input symbols, i-agents enter a waiting phase, by generating auxiliary objects to wait for the exact number of steps until then they generate another input. The set of programs of i-agent corresponding to symbol a_j is formed from following programs:

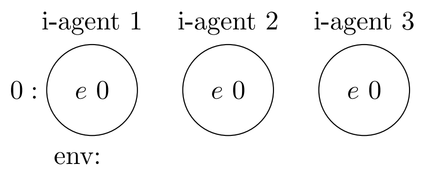
$$\begin{array}{ll}
\left\langle e \leftrightarrow a_j'' \ / \ e \leftrightarrow e; i \to i_o \right\rangle & \\
\left\langle a_j'' \to a_j; i_o \to i' \right\rangle & \\
\langle e \to a_j; i_o \to i_o' \rangle & \text{if } a_j \in C_i;\ i \geq 0 \\
\langle e \to e; i_o \to i_o' \rangle & \text{if } a_j \notin C_i;\ i \geq 0 \\
\langle a_j \leftrightarrow e; i_o' \to i' \rangle & \\
\langle e \leftrightarrow e; i_o' \to i' \rangle &
\end{array}$$

These programs are for generation of current input; the following programs are for synchronization of agents.

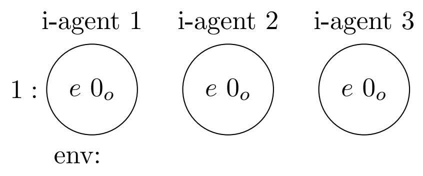
$$\begin{array}{ll}
\langle e \leftrightarrow F; i' \to i'' \rangle & \\
\langle F \to F_1; i'' \to i'' \rangle & \\
\langle F_x \to F_{x+1}; i'' \to i'' \rangle & \text{for } 1 \leq x < 3 + 2k + 2m \\
\langle F_y \to D; i'' \to i_D'' \rangle & y = 3 + 2k + 2m \\
\langle D \leftrightarrow e; i'' \to i_D \rangle & \\
\langle e \to F_y; i_D \to i'' \rangle & y = 5 + 2k + 2m + 1 \\
\langle F_z \to F_{z+1}; i'' \to i'' \rangle & 5 + 2k + 2m + 1 \leq z < 2 + 2k + 2m + 4|M| \\
\langle F_u \to E; i'' \to i_E \rangle & u = 2 + 2k + 2m + 4|A| \\
\langle E \leftrightarrow e; i_E \to i_E \rangle & \\
\langle e \to e; i_E \to (i+1) \rangle &
\end{array}$$

To help the easier understanding, we demonstrate the following example.

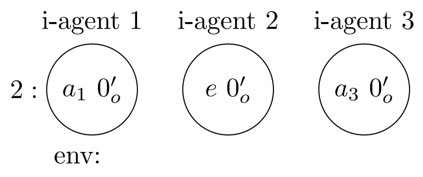
Let $\mathcal{A} = (S, A)$ be reaction system with $S = \{a_1, a_2, a_3\}$ and $A = \{r_1 : (\{a_2\}, \emptyset, \{a_2\});\ r_2 : (\{a_1, a_3\}, \{a_2\}, \{a_1, a_2\});\ r_3 : (\{a_3\}, \{a_1\}, \{a_1, a_2\})\}$. Let $C_0 = \{a_1, a_3\}$ be the input. The i-agents are initialized as follows:



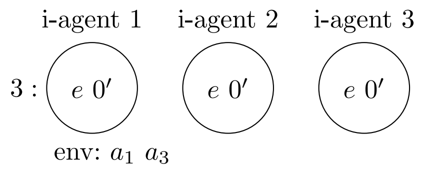
They execute the first program $\langle e \leftrightarrow a''_j \ / \ e \leftrightarrow e; i \rightarrow i_o \rangle$.



In this configuration, the i-agents have applicable programs that can generate objects corresponding to symbols from C_0.



After performing programs $\langle a_j \leftrightarrow e; 0'_o \rightarrow 0' \rangle$ (i-agent 1 and 3) or $\langle e \leftrightarrow e; 0'_o \rightarrow 0' \rangle$ (i-agent 2). All symbols in C_0 are placed in the environment.



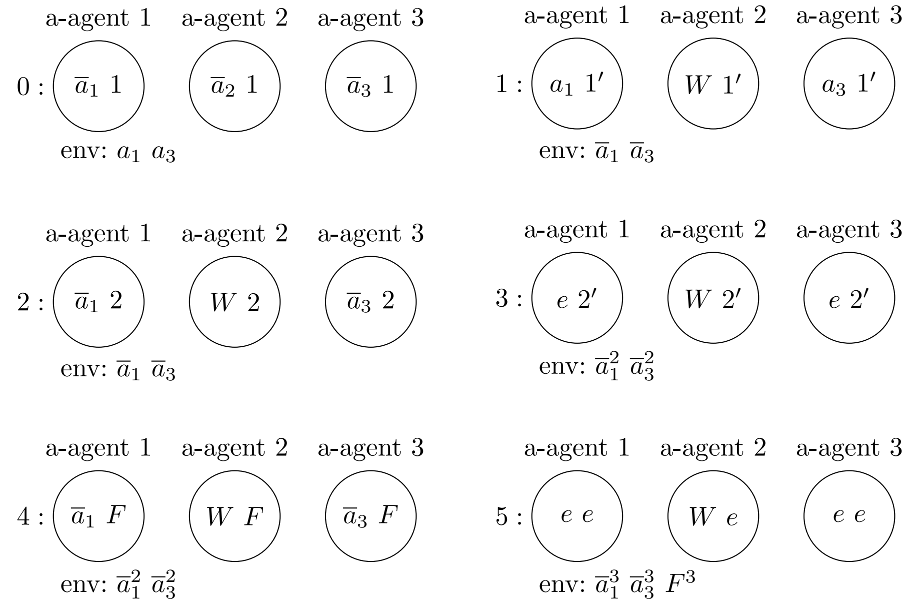
All three agents wait until object F appears in the environment.

2. Multiply Input

This phase is to multiply the input symbols to be "accessible" for every agent that simulates enabled reaction. Notice that in case of reaction systems all enabled reactions should be performed in parallel. We construct a-agents that generate $|A|$ symbols that appear in the environment after the first phase. The programs for this phase are:

$\langle \overline{a}_j \leftrightarrow a_j \ / \ \overline{a}_j \rightarrow W; \ 1 \rightarrow 1' \rangle$		
$\langle a_j \rightarrow \overline{a}_j; \ 1' \rightarrow 2 \rangle$		
$\langle \overline{a}_j \leftrightarrow e; \ i \rightarrow i' \rangle$	$\langle W \rightarrow W; \ i \rightarrow i' \rangle$	$1 \leq i \leq \|A\| - 1$
$\langle e \rightarrow \overline{a}_j; \ i' \rightarrow (i+1) \rangle$	$\langle W \rightarrow W; \ i' \rightarrow (i+1) \rangle$	$1 \leq i < \|A\| - 1$
$\langle e \rightarrow \overline{a}_j; \ i' \rightarrow F \rangle$	$\langle W \rightarrow W; \ i' \rightarrow F \rangle$	$i = \|A\| - 1$
$\langle \overline{a}_j \leftrightarrow e; \ F \leftrightarrow e \rangle$	$\langle W \rightarrow W; \ F \leftrightarrow e \rangle$	
$\langle e \rightarrow \overline{a}_j; \ e \rightarrow 1 \rangle$	$\langle W \rightarrow \overline{a}_j; \ e \rightarrow 1 \rangle$	

The number of a-agents is $|S|$. In the previously given example the second phase of simulation can be depicted as follows:


a-agent 1 a-agent 2 a-agent 3
0 : $\overline{a}_1$ 1 $\overline{a}_2$ 1 $\overline{a}_3$ 1
env: a_1 a_3
a-agent 1 a-agent 2 a-agent 3
1 : a_1 $1'$ W $1'$ a_3 $1'$
env: $\overline{a}_1$ $\overline{a}_3$
a-agent 1 a-agent 2 a-agent 3
2 : $\overline{a}_1$ 2 W 2 $\overline{a}_3$ 2
env: $\overline{a}_1$ $\overline{a}_3$
a-agent 1 a-agent 2 a-agent 3
3 : e $2'$ W $2'$ e $2'$
env: $\overline{a}_1^2$ $\overline{a}_3^2$
a-agent 1 a-agent 2 a-agent 3
4 : $\overline{a}_1$ F W F $\overline{a}_3$ F
env: $\overline{a}_1^2$ $\overline{a}_3^2$
a-agent 1 a-agent 2 a-agent 3
5 : e e W e e e
env: $\overline{a}_1^3$ $\overline{a}_3^3$ F^3

After generation of 3 copies of each kind of objects that appears in C_0 all a-agents stop working until object a_j appears in the environment. When the copies of F appear in the environment, i-agents consume them, i.e. import them from the environment. From this step on, they rewrite their content $y = 4 + 2k + 2m$ times. After y steps they are prepared to produce the next input.

3. Simulation of reactions

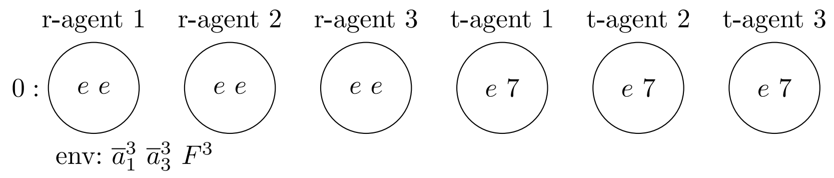
The task of the agents in this phase is to simulate the execution of reactions performed in the same step of the interactive process of the R system. The agents executing this task are called r-agents. Because they need some timing, there is another group of agents called t-agents. In certain steps, these t-agents generate objects that trigger the action of r-agents. The r-agents look for inhibitors, reactants and generate "semi-products" (a'_j) of reactions. After preparation, the search for inhibitors can be done in one step. If there is at least one inhibitor in the environment, then the reaction is not enabled. The r-agents have a program for each inhibitor that allows the agent to consume this inhibitor. If there is no inhibitor in the environment, then the agent has no applicable program. The number of r-agents is the same as the number of t-agents and it equals to $|A|$. Let $r : (R_r, I_r, P_r)$ is a reaction in $\mathcal{A}$. The programs for search for inhibitors are:

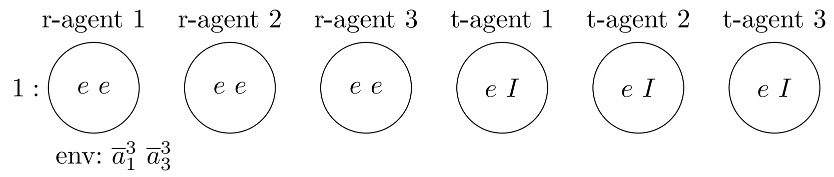
r-agents		t-agents	
a) search for inhibitors		a) activation of r-agents	
$\langle e \to e;\ e \leftrightarrow I \rangle$		$\langle e \to e;\ i \to (i+1) \rangle$	$0 \leq i < 2\|A\|+1$
$\langle e \leftrightarrow \overline{a}_j;\ I \to C \rangle$	$a_j \in I_r$	$\langle e \to e;\ (2\|A\|+1) \to I \rangle$	
$\langle \overline{a}_j \to e;\ C \to e \rangle$		$\langle e \to T';\ I \leftrightarrow e \rangle$	
$\langle C \to C;\ e \leftrightarrow T \rangle$		$\langle T' \to T;\ e \leftrightarrow e \rangle$	
$\langle C \to e;\ T \to e \rangle$		$\langle T \leftrightarrow e;\ e \to 0' \rangle$	
$\langle e \leftrightarrow T;\ I \to R \rangle$		$\langle e \to e;\ i' \to (i+1)' \rangle$	$0 \leq i' \leq 2k+2m$
$\langle T \to e;\ R \to R'_0 \rangle$		$\langle e \to e;\ (2k+2m) \to 0 \rangle$	

Let us return to the previous example. $\mathcal{A}$ has three reactions:

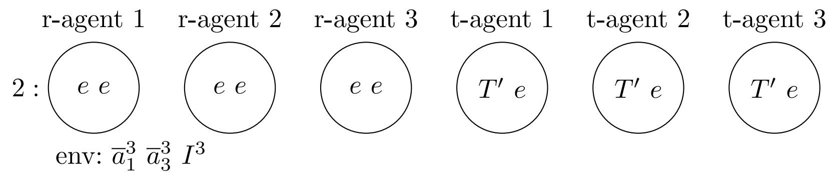
$$
\begin{aligned}
r_1 &= (\{a_2\}, \emptyset, \{a_2\})\\
r_2 &= (\{a_1, a_3\}, \{a_2\}, \{a_1, a_2\})\\
r_3 &= (\{a_3\}, \{a_1\}, \{a_1, a_2\})
\end{aligned}
$$

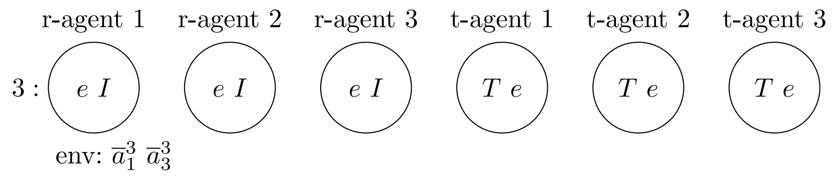
and $C_0 = \{a_1, a_3\}$. Then $k = 2,\ m = 2$ and there are three copies of $\overline{a}_1$ and three copies of $\overline{a}_3$ in the environment of the P colony. The first configuration of this phase is:

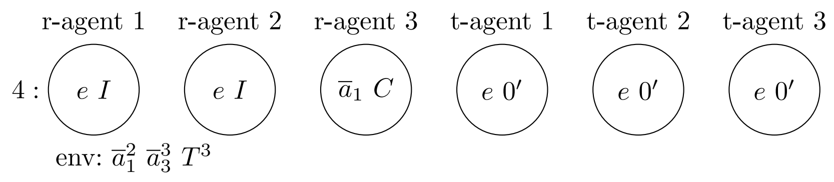




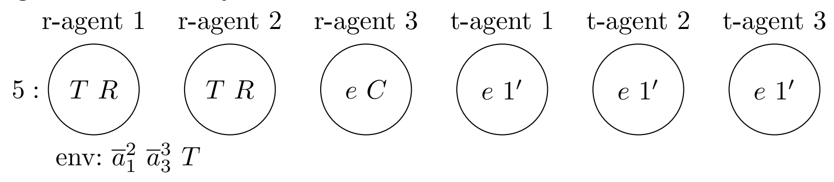
The copies of object F were consumed by i-agents (see the third program of i-agents in the first phase).

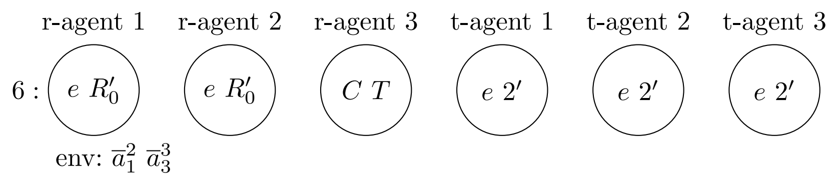






Reaction r_1 has empty inhibitor set, so the corresponding r-agent has no program to apply in this configuration. There is one inhibitor, a_2, in inhibitor set of reaction r_2 and because a_2 is not present in the environment, therefore r-agent 2 has no applicable program in current configuration. Due to the presence of a_1 in C_0, the third reaction is not enabled by C_0 and r-agent 3 has one program to execute. The agent consumes object a_1 from the environment.





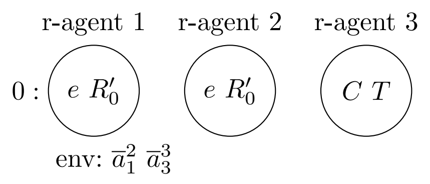
Those agents which have object R inside can continue the simulation of executing reaction by search for reactants. All reactants must be present in the environment to enable reaction. For every reaction we make a sequence of reactants $(a_j)_{j=1}^k$ in random order. Then we can construct programs for search for reactants phase:

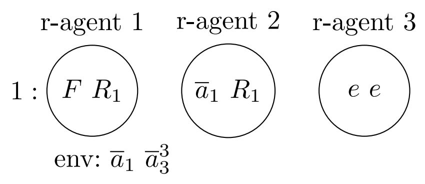
r-agents

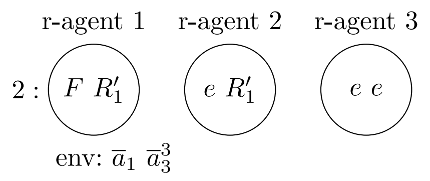
b) search for reactants

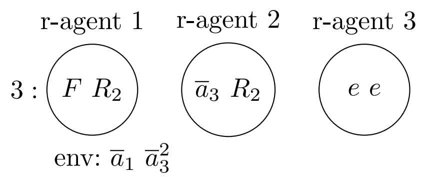
$\langle e \leftrightarrow \overline{a}_j\ /\ e \to F;\ R'_{j-1} \to R_j \rangle$	$a_j \in R_r;\ 1 \le j \le \lvert R_r \rvert$
$\langle \overline{a}_j \to e;\ R_j \to R'_j \rangle$	$a_j \in R_r;\ 1 \le j \le \lvert R_r \rvert$
$\langle F \to F;\ R'_{j-1} \to R_j \rangle$	$1 < j \le k$
$\langle F \to F;\ R_j \to R'_j \rangle$	$1 \le j \le k$
$\langle e \to e;\ R'_{j-1} \to R_j \rangle$	$\lvert R_r \rvert < j \le k$
$\langle e \to e;\ R_j \to R'_j \rangle$	$\lvert R_r \rvert < j \le k$
$\langle F \to e;\ R'_k \to e \rangle$	

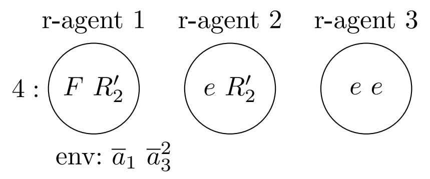
In the example P colony, we develop, the search for reactants is performed as follows:









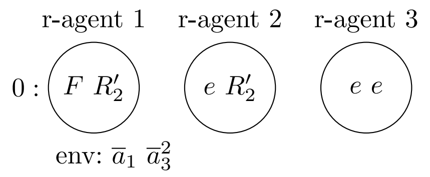


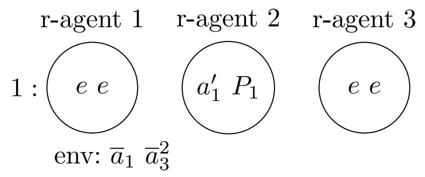
Only r-agents in state (e, R'_k) can continue the simulation. They consumed all the reactants and because they pass the phase a) the corresponding reaction is enabled by W_i. In phase of products generation the r-agents will generate and put into environment "semi-products", i.e. objects corresponding to products. For this purpose, we also need a sequence of products for each reaction.

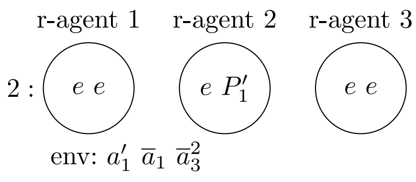
r-agents

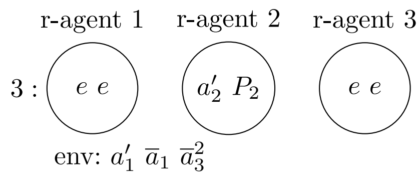
c) generation of products

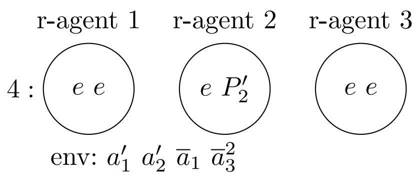
$\langle e \to a'_1;\ R'_k \to P_1 \rangle$
$\langle e \to a'_j;\ P'_{j-1} \to P_j \rangle \quad a_j \in P_r;\ 1 < j \leq |P_r|$
$\langle e \leftrightarrow a'_j;\ P_j \to P'_j \rangle \quad a_j \in P_r;\ 1 \leq j \leq |P_r|$
$\langle e \to e;\ P'_{j-1} \to P_j \rangle \quad |P_r| < j \leq m$
$\langle e \to e;\ P_j \to P'_j \rangle \quad |P_r| < j \leq m$
$\langle e \to e;\ P'_m \to e \rangle$

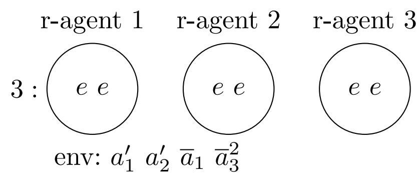








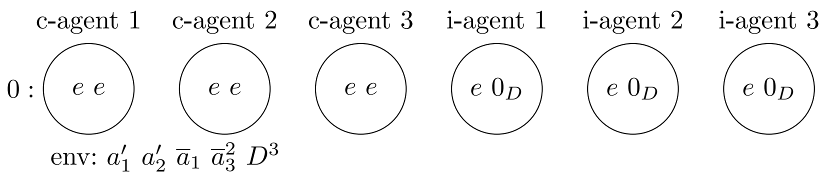


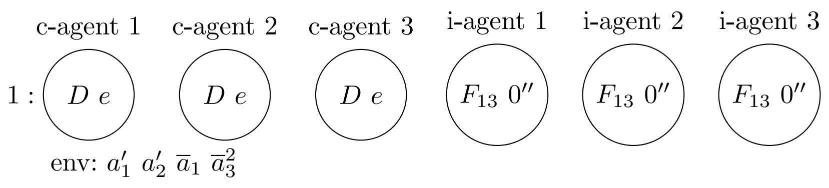


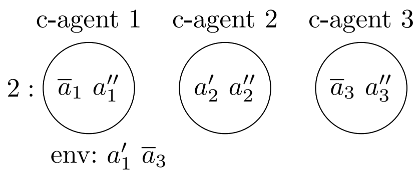
4. Consuming phase

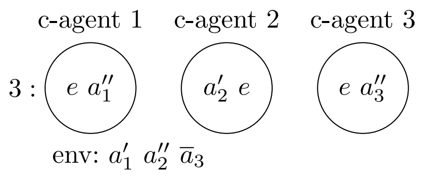
In this phase all unused over-lined objects are consumed by c-agents as well as copies of "semi-products". Only one copy of semi-product stays in the environment. The number of c-agents is $|S|$ and this phase takes at most $4|A|$ steps.

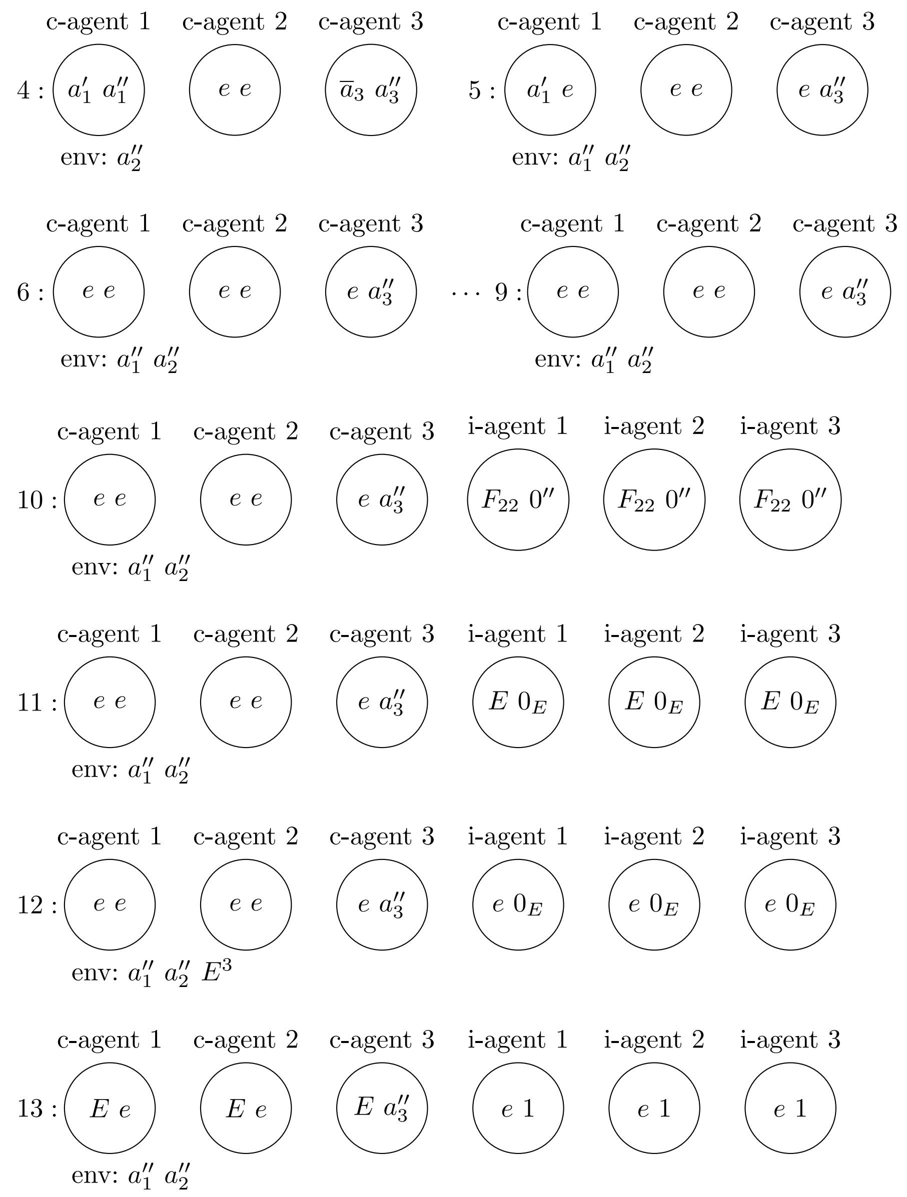
$\langle e \leftrightarrow D;\ e \to e \rangle$ $\quad \left\langle e \leftrightarrow \overline{a}_j\ /\ e \leftrightarrow a_j';\ D \to a_j'' \right\rangle$
$\left\langle \overline{a}_j \to e;\ a_j'' \to a_j'' \right\rangle$ $\quad \left\langle e \leftrightarrow \overline{a}_j\ /\ e \leftrightarrow a_j';\ a_j'' \to a_j'' \right\rangle$
$\left\langle a_j' \to a_j';\ a_j'' \leftrightarrow e \right\rangle$ $\quad \left\langle e \leftrightarrow a_j'\ /\ e \leftrightarrow e;\ a_j' \to e \right\rangle$
$\langle e \leftrightarrow E;\ e \to e \rangle$ $\quad \langle D \to e;\ e \leftrightarrow E \rangle$
$\left\langle a_j'' \to e;\ e \leftrightarrow E \right\rangle$ $\quad \langle E \to e;\ e \to e \rangle$










c-agent 1 c-agent 2 c-agent 3
4 : $a'_1\ a''_1$ | $e\ e$ | $\overline{a}_3\ a''_3$
env: a''_2
c-agent 1 c-agent 2 c-agent 3
5 : $a'_1\ e$ | $e\ e$ | $e\ a''_3$
env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3
6 : $e\ e$ | $e\ e$ | $e\ a''_3$
env: $a''_1\ a''_2$
$\cdots$
c-agent 1 c-agent 2 c-agent 3
9 : $e\ e$ | $e\ e$ | $e\ a''_3$
env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
10 : $e\ e$ | $e\ e$ | $e\ a''_3$ | $F_{22}\ 0''$ | $F_{22}\ 0''$ | $F_{22}\ 0''$
env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
11 : $e\ e$ | $e\ e$ | $e\ a''_3$ | $E\ 0_E$ | $E\ 0_E$ | $E\ 0_E$
env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
12 : $e\ e$ | $e\ e$ | $e\ a''_3$ | $e\ 0_E$ | $e\ 0_E$ | $e\ 0_E$
env: $a''_1\ a''_2\ E^3$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
13 : $E\ e$ | $E\ e$ | $E\ a''_3$ | $e\ 1$ | $e\ 1$ | $e\ 1$
env: $a''_1\ a''_2$

In this configuration, c-agents rewrite all objects inside them to environmental objects. Simulation can continue with the first phase - generation of input. The i-agents can consume objects of a type a''_j and they put into the environment objects a_j and objects of the next input (if they are not included in products of previous step).

3 Conclusions

In this paper we presented the result obtained by examining P colonies with connection to R systems. In future research we plan further investigation of P colonies that resemble reaction systems in terms of shared environment and computation.

Acknowledgments.

The work of L. Ciencialová and L. Cienciala was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602, by SGS/11/2019.

References

1. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., and Sosík, P.: P colonies. Bulletin of the International Membrane Computing Society 1(2):119–156 (2016).
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, Gy.: Computing with cells in environment: P colonies. Journal of Multiple-Valued Logic and Soft Computing 12(3-4 SPEC. ISS.), pp. 201–215 (2006)
3. Csuhaj-Varjú, E., Kelemen, J., Păun, Gh., Dassow, J.(eds.): Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994)
4. Ehrenfeucht, A., Rozenberg, A.: Basic notions of reaction systems. In: Calude, C.S., Calude, E., Dinneen, M.J. (Eds.), Developments in Language Theory, 8th International Conference, DLT 2004. In: Lecture Notes in Computer Science, vol.3340, Springer, 27-–29 (2005)
5. Kelemenová, A.: P Colonies. Chapter 23.1, In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 584–593. Oxford University Press (2010)
6. Kelemen, J., Kelemenová, A.: On P colonies, a biochemically inspired model of computation. In: Proc. of the 6^{th} International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH. pp. 40–56. Hungary (2005)
7. Kelemen, J., Kelemenová, A., Păun, G.: Preview of P Colonies: A Biochemically Inspired Computing Model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX). pp. 82–86. Boston, Mass (2004)
8. Kelemen, J., Kelemenová, A.: A Grammar-Theoretic Treatment of Multiagent Systems. Cybern. Syst. 23(6), 621–633 (1992),
9. Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
10. Rozenberg, G., Salomaa, A.(eds.): Handbook of Formal Languages I-III. Springer Verlag., Berin-Heidelberg-New York (1997)

Extracting Parallelism in Simulation Algorithms for PDP systems

Miguel Á. Martínez-del-Amor, Andrés Doncel-Ramírez, David Orellana-Martín, Ignacio Pérez-Hurtado

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es, andrestp95@gmail.com, dorellana@us.es, perezh@us.es

Summary. Population Dynamics P systems is a modelling framework that have been used successfully for some important real ecosystems. This model is inherently probabilistic, and the scheme of rules is very flexible, allowing even cooperation between membranes. Thus, its simulation has been a challenge in the past years, leading to several simulation algorithms. The latest one, which has been proved to be the most accurate so far, is DCBA. The main drawback of DCBA is its complexity, requiring a very large table to handle all competitions. In this paper, we discuss two strategies to decrease this table, allowing a more lightweight version of DCBA that can be used in parallel implementations.

Keywords: Membrane Computing, Population Dynamics, Parallel simulation

1 Introduction

Some very important real ecosystems have been modelled using the formal framework called Population Dynamics P (PDP) systems [8, 6]. This framework consists of a multienvironment P system model [11] that contains one single cell-like P system within the nodes (environments) of a directed graph. Each of the cell-like P systems have the same skeleton (membrane tree and evolution rules). Thus, PDP systems have to kinds of rules: evolution (skeleton) and communication rules. Moreover, PDP systems is a probabilistic model, in the sense that probabilities are associated with the rules. Skeleton rules may have associated different probabilities regarded the environment where they are located.

The very flexible pattern of skeleton rules, where object cooperation can happen even between membranes (the active membrane and its parent), increases the complexity when designing simulation algorithms. For this reason, several approaches have been made: BBB (Binomial Block Based), DNDP (Direct Non Deterministic distribution with Probabilities) and DCBA (Direct distribution based on Consistent Blocks Algorithm) [13, 3, 5]. These algorithms are designed to tackled specifically the competition for resources that can happen in the models. Specifically, different rules having overlapping but different left-hand sides compete for objects in the multisets.

DCBA is the algorithm that has been demonstrated to show more accurate results, according to the way the formal framework is employed for ecosystem modelling [3]. Moreover, it is highly parallelizable, as shown in the GPU implementation called ABCD-GPU [12, 15]. However, the algorithm has several drawbacks:

1. it consists of four phases to simulate just one computation step;
2. the first phase uses a distribution table that does not scale well when increasing the amount of rules and objects in the alphabet;
3. the second phase is inherently sequential;

In this paper, we discuss two different strategies to cope with the second drawback mentioned above: *adaptative DCBA* and *μ-DCBA* (or DCBA with partitions of rules). The former, already published in [16], is a solution where the designer provides high-level information of the model, such as rule modules, so that the simulator knows that DCBA must be applied locally to each module. The latter is a novel solution, still unpublished, where the rule competition is pre-computed (before starting the simulation), so that DCBA is applied locally to each partition. These two methods were first mentioned in [12], but we extend these ideas here.

The rest of the paper is structured as follows: Section 2 briefly recall the model of PDP systems and Section 3 its definitions for DCBA; Section 4 describes the concept of adaptative simulator and how it is applied to DCBA; Section 5 introduces the idea of μ-DCBA; and finally, Section 6 ends the document with conclusions and future work.

2 Population Dynamics P systems

Next, we recall the formal definition of a PDP system. We also provide some concepts required for DCBA, and the main loop of the algorithm. More information than the one provided here can be found in [7, 9, 3].

Definition 1. *A Population Dynamics P system of degree (q, m) with $q \geq 1$, $m \geq 1$, and taking $T \geq 1$ time units, is a tuple*

$$\Pi = (G, \Gamma, \Sigma, T, R_E, \mu, R, \{f_{r,j} : r \in R, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : 1 \leq i \leq q, 1 \leq j \leq m\})$$

where:

- *$G = (V, S)$ is a directed graph. Let the vertices be $V = \{e_1, \dots, e_m\}$, also called environments;*
- *Γ is the working alphabet and $\Sigma \subsetneq \Gamma$ is an alphabet representing the only objects that can be present in the environments;*
- *T is a natural number that represents the simulation time of the system;*
- *R_E is a finite set containing the so called communication rules, that send objects between environments. They are of the form*

$$(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$$

  where $x, y_1, \dots, y_h \in \Sigma$, $(e_j, e_{j_l}) \in S$ $(1 \leq l \leq h)$ and p is a computable function from $\{1, \dots, T\}$ to $[0, 1]$. By default and for simplicity, we assume $p = 1$, in case it is not specified for a rule. Moreover, for each rule of this form, the following holds: $(e_j, e_{j_h}) \in S$. These functions verify the following:
  - *For each $e_j \in V$ and $x \in \Sigma$, the sum of functions associated with the rules whose left-hand side is $(x)_{e_j}$, is exactly 1.*
- *μ is a membrane structure with q membranes injectively labelled by $1, \dots, q$. The skin membrane is labelled by 1. An electrical charge from the set $EC = \{0, +, -\}$ is also associated with each membrane.*
- *R is a finite set of evolution (skeleton) rules of the form*

$$u[\,v\,]_i^{\alpha} \rightarrow u'[\,v'\,]_i^{\alpha'}$$

  where $u, v, u', v' \in \Gamma^*$, i $(1 \leq i \leq q)$, $u + v \neq \lambda$ and $\alpha, \alpha' \in \{0, +, -\}$. The following restriction must hold:
  - *If $(x)_{e_j}$ is the left-hand side of a rule from R_E, then none of the rules of R has a left-hand side of the form $u[v]_1^{\alpha}$, for any $u, v \in \Gamma^*$ and $\alpha \in \{0, +, -\}$, having $x \in u$.*
- *For each $r \in R$ and for each j $(1 \leq j \leq m)$, the function $f_{r,j} : \{1, \dots, T\} \longrightarrow [0, 1]$ is computable. These functions verify the following:*
  - *For each $u, v \in \Gamma^*$, i $(1 \leq i \leq q)$, $\alpha, \alpha' \in \{0, +, -\}$ and j $(1 \leq j \leq m)$ the sum of functions associated with j and the set of rules whose left-hand side is $u[v]_i^{\alpha}$ and whose right-hand side has polarization α', is the constant function 1.*
- *For each j, $(1 \leq j \leq m)$, $\mathcal{M}_{1j}, \dots, \mathcal{M}_{qj}$ are strings over Γ, describing the multisets of objects initially placed within the regions in environment e_j (also known as initial configuration).*

In other words, a PDP system consists of m environments $e_1, \dots, e_m$ linked between them by the edges from the directed graph G. Each environment e_j contains a P system, $\Pi_j = (\Gamma, \mu, R_{\Pi_j}, \mathcal{M}_{0j}, \dots \mathcal{M}_{q,j})$, of degree q, where every rule $r \in R$ has a computable function $f_{r,j}$ (specific for environment j) associated with it.

A *configuration* of the system at an instant t is a tuple of multisets of objects present in the m environments and at each of the regions of each Π_j, together with the polarizations of the membranes in each P system. At the initial configuration of the system we assume that all environments are empty and all membranes have a neutral polarization. As it is usual in cell-like P systems, we also assume that a global clock exists which synchronizes all environments.

The P system can pass from one configuration to the next one by using the rules from $\bigcup_{j=1}^{m} R_{\Pi_j} \cup R_E$ as follows: at each transition step, the rules to be applied are selected according to the *probabilities* assigned to them, and all applicable rules are simultaneously applied in a maximal way (i.e. no more rules can be further applicable). For rules in R_{Π_j}, the charge of the (active) membrane will be changed. In this sense, the consistency of charges must hold: in order to apply several rules of R_{Π_j} simultaneously to the same membrane, all the rules must have the same electrical charge on their right-hand side.

When a communication rule $(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \dots (y_h)_{e_{j_h}}$ between environments is applied, object x passes from e_j to $e_{j_1}, \dots, e_{j_h}$ possibly modified into objects $y_1, \dots, y_h$ respectively. At any moment t $(1 \leq t \leq T)$ for each object x in environment e_j, if there exist communication rules whose left-hand side is $(x)_{e_j}$, then one of these rules will be applied. If more than one communication rule can be applied to an object, the system selects one randomly, according to their probability which is given by $p(t)$.

3 Simulation algorithms

The simulation algorithms for PDP systems called BBB and DCBA are based on the grouping of rules into blocks. These groups are constructed by looking the left-hand side. Note that rules having the same left-hand side must have associated probabilities summing 1. Specifically, DCBA works using a refined definition of block, called consistent block, as shown in Definition 2. DNDP does not use the concept of blocks, but it selects rules by a random loop instead.

Definition 2. *Rules from R and R_E are classified into consistent blocks by either of the following:*

a. the rule block associated with $(i, \alpha, \alpha', u, v)$ is $B_{i,\alpha,\alpha',u,v} = \{r \in R : LHS(r) = (i, \alpha, u, v) \wedge charge(RHS(r)) = \alpha'\}$;
b. the rule block associated with (e_j, x) is $B_{e_j,x} = \{r \in R_E : LHS(r) = (e_j, x)\}$.

It is important to remark that the selection of rules in BBB and DCBA relies always first on selecting blocks, calculating a multinomial random variate, and therefore obtaining a selection of rules within each block. In this sense, we can say that rules within a block will not compete among objects when using BBB and DCBA, because they are selected altogether. This, again, does not hold in DNDP, where rules are selected individually according to the probabilities. Block competition will be defined later in Definition 3.

DCBA tackles the resource competition issue by performing a proportional distribution of objects among competing blocks. This is done by using the *distribution table*, which is a system-wide time having blocks per columns, and pairs (object,region) per rows. Algorithm 1 shows a summary of the algorithm, which can be depicted in [3]. It can be seen that, as usual, each loop iteration is made by two stages: selection and execution. Selection stage consists of three phases: Phase 1 distributes objects to the blocks in a certain proportional way, Phase 2 ensures *maximality* by checking the maximal number of applications of each block, and Phase 3 translates from block to rule applications by calculating random numbers using a multinomial distribution. Finally, execution stage (or Phase 4) generates the right-hand side of rules.

Algorithm 1 DCBA MAIN LOOP

```
Require: A PDP system $\Pi$ of degree $(q, m)$, $T \geq 1$ (time units), $A \geq 1$ (accuracy
    parameter), and an initial configuration $C_0$.
 1: $(\mathcal{T}) \leftarrow$ INITIALIZATION $(\Pi)$          ▷ (Initializes the distribution table)
 2: for $t \leftarrow 0$ to $T-1$ do                         ▷ (For each transition step)
 3:     SELECTION:        ▷ (Selection of rules, subtracting their left-hand sides)
 4:     $B_{sel} \leftarrow \emptyset$, $R_{sel} \leftarrow \emptyset$
 5:     $(\mathcal{T}^t, C'_t, B_{sel}) \leftarrow$ PHASE 1 $(\Pi, A, C_t, \mathcal{T})$      ▷ (Distribution of objects)
 6:     $(C'_t, B_{sel}) \leftarrow$ PHASE 2 $(\Pi, C'_t, B_{sel}, \mathcal{T}^t)$          ▷ (Ensure Maximality)
 7:     $(R_{sel}) \leftarrow$ PHASE 3 $(\Pi, B_{sel})$              ▷ (Probabilistic distribution)
 8:     EXECUTION:         ▷ (Execution of rules, adding their right-hand sides)
 9:     $(C_{t+1}) \leftarrow$ PHASE 4 $(\Pi, C'_t, R_{sel})$
10: end for
```

It is important to remark that the stages of DCBA can be performed independently (and hence, in parallel) to each environment [14, 15]. However, we need a synchronization point between selection and execution stages, because communication rules might generate objects in different environments. This way, the main loop of DCBA can be rewritten to the form in Algorithm 2.

We can finally identify two main bottlenecks in the simulation algorithms. The first one is tackled by the adaptative

1. All algorithms (DCBA, BBB and DNDP) need to go through every defined rule in the system at every transition step, in order to check if it is applicable. Indeed, there is no way to know in advance which rules can be applicable in each time step.
2. DCBA is specifically designed to cope with block competitions in an accurate way, but it has to assume that all rules can have cross competitions (rule a competes with rule b, and rule b with rule c, then rules a, b and c must agree on the objects to consume).

4 Adaptative DCBA

The idea of adaptative simulators was introduced and analysed in [16]. It is inspired in the way directives work in common programming languages. They are special

Algorithm 2 DCBA MAIN LOOP FOR ENVIRONMENTS

```
Require: A PDP system $\Pi$ of degree $(q,m)$, $T \geq 1$ (time units), $A \geq 1$ (accuracy
    parameter), and an initial configuration $C_0$.
 1: for $j \leftarrow 1$ to $m$ do                                  ▷ (For each environment j)
 2:     $(\mathcal{T}_j) \leftarrow$ INITIALIZATION $(j,\Pi)$            ▷ (Initializes the table for environment j)
 3: end for
 4: for $t \leftarrow 0$ to $T-1$ do                                ▷ (For each transition step)
 5:     for $j \leftarrow 1$ to $m$ do                              ▷ (For each environment j)
 6:         SELECTION:                                     ▷ (Selection of rules for environment j)
 7:         $B^j_{sel} \leftarrow \emptyset$, $R^j_{sel} \leftarrow \emptyset$
 8:         $(\mathcal{T}^t_j, C'_t, B^j_{sel}) \leftarrow$ PHASE 1 $(j,\Pi,A,C_t,\mathcal{T}_j)$      ▷ (Distribution of objects)
 9:         $(C'_t, B^j_{sel}) \leftarrow$ PHASE 2 $(j,\Pi,C'_t,B^j_{sel},\mathcal{T}^t_j)$           ▷ (Ensure Maximality)
10:         $(R^j_{sel}) \leftarrow$ PHASE 3 $(j,\Pi,B^j_{sel})$               ▷ (Probabilistic distribution)
11:     end for
12:     for $j \leftarrow 1$ to $m$ do                              ▷ (For each environment j)
13:         EXECUTION:                                     ▷ (Execution of rules for environment j)
14:         $(C_{t+1}) \leftarrow$ PHASE 4 $(j,\Pi,C'_t,R^j_{sel})$
15:     end for
16: end for
```

syntactic elements that tell extra information to the compiler, allowing to better adapt the code for some purposes if the compiler accepts it (e.g. in OpenMP, one call easily ask to parallelize the iterations of a loop), or just processing the code dismissing that information (the code is still valid without the directives). This way, a P system model designer can also provide very useful information to the simulator, rather than just the syntactic and/or semantic elements of the P system to simulate. For instance, P system models are usually designed bearing in mind a global algorithmic scheme, where the computation of the P system is subdivided into stages of specific purposes (e.g. in SAT solutions for active membranes, there are stages for generating membranes, other stages for check-in solutions, etc.).

Specifically in PDP systems, ecosystem modellers often use algorithmic schemes for their models [7]. This is done by first defining a cycle, which corresponds to a certain time in the simulated ecosystem (e.g. one year). A cycle in the model is a fixed amount of transition steps where a sequence of modules take place. These modules reproduces certain processes such as reproduction of species, feeding, migration, etc. Moreover, these modules consist of certain rules that are carefully designed to model the corresponding process. Therefore, we can say that somehow, the model designer already knows which rules can be executed in each time step. Thus, if they are able to provide that information in form of a directive-like syntax, the simulator can take advantage of this to dismiss rules automatically at each step.

In [16], the ABCD-GPU simulator was turned into adaptative. First, the model designer is able to provide the information of the modules they are defining by using the new P-Lingua 5 software [17]. This new version now includes new syntax elements called *features*. They are written as `@featureName = featureValue`, and can be defined globally (for the whole system) or locally (for individual rules). ABCD-GPU takes this information to organize the rules by modules. Of course, if the simulator does not recognize the information provided by the features, it can proceed and simulate the system without problems.

The simulator also pre-computes which modules are active in each step within the cycle, so that it can easily access the rules that might be applicable at each transition step. Furthermore, a parallel implementation can harness this to reach more parallelism, specifically between parallel modules that can be active at certain steps. This design helped to improve the performance by 2.5x extra when using a P100 GPU [16].

As for environments, DCBA's stages can be performed independently per active module, but it requires a synchronization point. Algorithm 3 shows how it can be re-defined to handle modules and environments. There two variables for steps: t is the global time step of the simulation, and s is the step within a cycle. After reading the information provided by the model designer along with the PDP system (e.g. with P-Lingua 5), we get a map to associate rules to each module, and another to know which modules are active in each step of the cycle. Finally, we also provide the total amount of modules d.

5 DCBA with partitions of rules

As mentioned above, DCBA assumes that all blocks can compete for objects. These competitions can make dependencies between blocks that are encoded in the transition table, which takes care of distributing the resources (objects) to the blocks that can be applied. Later on, rules within blocks will compute its applications using a multinomial random variate. In order to decrease the size of the distribution table, we can pre-calculate which blocks are actually competing for resources one each other. This kind of problems have been already tackled in the literature [1, 19]. Thus, in this paper we propose a similar concept, but adapted for PDP systems, where rules have a more flexible pattern. If we focus in DCBA, we can formally define the condition of block competition as shown in Definition 3.

Definition 3. *Two consistent blocks* $B^1_{i_1,\alpha_1,\alpha'_1,u_1,v_1}$ *and* $B^2_{i_2,\alpha_2,\alpha'_2,u_2,v_2}$ *compete for objects when both the following holds:*

(a) The two blocks are mutually consistent. That is, if $i_1 = i_2 \wedge \alpha_1 = \alpha_2$ *then* $\alpha'_1 = \alpha'_2$*;*
(b) Their left-hand sides overlap. That is, either of the following conditions hold:
- *If* $i_1 = i_2$ *and* $\alpha_1 = \alpha_2$ *then* $u_1 \cap u_2 \neq \emptyset$ *or* $v_1 \cap v_2 \neq \emptyset$*;*
- *If* $i_1 \neq i_2$ *but* i_1 *is the parent membrane of* i_2*, then* $v_2 \cap u_1 \neq \emptyset$*, or* i_2 *is the parent membrane of* i_1*, then* $u_2 \cap v_1 \neq \emptyset$*.*

It is important to remark that blocks from communication rules do not compete with each other, nor with blocks from evolution rules (see the definition in

Algorithm 3 DCBA MAIN LOOP FOR ENVIRONMENTS AND MODULES

Require: A PDP system Π of degree (q, m), $T \geq 1$ (time units), $A \geq 1$ (*accuracy* parameter), an initial configuration C_0, the number of time units per cycle c, the amount of modules d, a structure mapping the rules and blocks per module MR, and another structure mapping which module is active at each step in the cycle MS.

```
1: for $j \leftarrow 1$ to $m$ do                                  ▷ (For each environment j)
2:     for $k \leftarrow 1$ to $d$ do                                  ▷ (For each module k)
3:         $(\mathcal{T}_{j,k}) \leftarrow$ INITIALIZATION $(j,\Pi,k,MR)$        ▷ (Creates the table for environment j and module k)
4:     end for
5: end for
6: $t \leftarrow 0$
7: while $t < T$ do                                              ▷ (For each transition step)
8:     for $t \leftarrow t$ to $t+c-1$ do               ▷ (Looping the transition steps inside a cycle)
9:         $s \leftarrow t \bmod c$                                  ▷ (The step within the cycle)
10:        for $j \leftarrow 1$ to $m$ do                            ▷ (For each environment j)
11:            for $k \leftarrow 1$ to $d$ do                           ▷ (For each module k)
12:                SELECTION:                ▷ (Selection for environment j and module k)
13:                if $MS[k,s]$ then                      ▷ (If module k is active in step s)
14:                    $B_{sel}^{j,k} \leftarrow \emptyset$, $R_{sel}^{j,k} \leftarrow \emptyset$
15:                    $(\mathcal{T}_{j,k}^t, C'_t, B_{sel}^{j,k}) \leftarrow$ PHASE 1 $(j,\Pi,A,C_t,\mathcal{T}_{j,k},k,MR)$
16:                    $(C'_t, B_{sel}^{j,k}) \leftarrow$ PHASE 2 $(j,\Pi,C'_t,B_{sel}^{j,k},\mathcal{T}_{j,k}^t,k,MR)$
17:                    $(R_{sel}^{j,k}) \leftarrow$ PHASE 3 $(j,\Pi,B_{sel}^{j,k},k,MR)$
18:                end if
19:            end for
20:        end for
21:        for $j \leftarrow 1$ to $m$ do                            ▷ (For each environment j)
22:            for $k \leftarrow 1$ to $d$ do                           ▷ (For each module k)
23:                EXECUTION:                ▷ (Execution for environment j and module k)
24:                if $MS[k,s]$ then                      ▷ (If module k is active in step s)
25:                    $(C_{t+1}) \leftarrow$ PHASE 4 $(j,\Pi,C'_t,R_{sel}^{j,k},k,MR)$
26:                end if
27:            end for
28:        end for
29:    end for
30: end while
```

Section 2). Let us represent the competition relationship as an undirected graph $G_c = (V_c, E_c)$, where V_c is the set of all rule blocks and E is the set of edges connecting the blocks that directly compete one with each other. We will therefore say that two blocks will compete, directly or indirectly, if there exists a path between them. Thus, we can calculate partitions of competitions from the set of rule blocks as depicted in Definition 4.

Definition 4. *Given a set of rule blocks $V = \{B_1, \ldots, B_k\}$, a partition of competition is a partition of the set V, $P = \{P_1, \ldots, P_l\}$, where the following holds:*

a block B_i belongs to the set P_i if and only if it competes, directly or indirectly, with the rest of blocks in C_i, and do not compete with any of the rule blocks form the rest of sets in P. The union of the sets in P is V.

Specifically, communication rule blocks form partitions with just one element. It is easy to compute the partitions of competitions from the set of rule blocks by calculating the connected components in the graph G_c. After having this, we can redefine the DCBA algorithm to be executed locally to each partition, if it contains more than one elements (also known as μ-DCBA). Generally, we can re-structure the algorithm as shown in Algorithm 4.

Algorithm 4 DCBA MAIN LOOP FOR ENVIRONMENTS AND PARTITIONS

```
Require: A PDP system $\Pi$ of degree $(q, m)$, $T \geq 1$ (time units), $A \geq 1$ (accuracy parameter), and an initial configuration $C_0$.
 1: $(p, P) \leftarrow$ PARTITIONS $(\Pi)$   ▷ (Compute the $p$ partitions of rules in the map $P$)
 2: for $j \leftarrow 1$ to $m$ do   ▷ (For each environment j)
 3:     for $i \leftarrow 1$ to $p$ do   ▷ (For each partition i)
 4:         $(\mathcal{T}_{j,i}) \leftarrow$ INITIALIZATION $(j,\Pi,i,P)$   ▷ (The table for environment j and partition i)
 5:     end for
 6: end for
 7: for $t \leftarrow 0$ to $T-1$ do   ▷ (For each transition step)
 8:     for $j \leftarrow 1$ to $m$ do   ▷ (For each environment j)
 9:         for $i \leftarrow 1$ to $p$ do   ▷ (For each partition i)
10:             SELECTION:   ▷ (Selection for environment j and partition i)
11:             $B^{j,i}_{sel} \leftarrow \emptyset$, $R^{j,i}_{sel} \leftarrow \emptyset$
12:             $(\mathcal{T}^t_{j,i}, C'_t, B^{j,i}_{sel}) \leftarrow$ PHASE 1 $(j, \Pi, A, C_t, \mathcal{T}_{j,i}, i, P)$
13:             $(C'_t, B^{j,i}_{sel}) \leftarrow$ PHASE 2 $(j, \Pi, C'_t, B^{j,i}_{sel}, \mathcal{T}^t_{j,i}, i, P)$
14:             $(R^{j,i}_{sel}) \leftarrow$ PHASE 3 $(j, \Pi, B^{j,i}_{sel}, i, P)$
15:         end for
16:     end for
17:     for $j \leftarrow 1$ to $m$ do   ▷ (For each environment j)
18:         for $i \leftarrow 1$ to $p$ do   ▷ (For each partition i)
19:             EXECUTION:   ▷ (Execution for environment j and partition i)
20:             $(C_{t+1}) \leftarrow$ PHASE 4 $(j, \Pi, C'_t, R^{j,i}_{sel}, i, P)$
21:         end for
22:     end for
23: end for
```

Preliminary results show an improvement of around 2x of extra speedup when using the partitions to find more parallelism on a K40c GPU with a model of the Bearded Vulture in the Pyrenees [10, 4]. This means that the μ-DCBA implementations usually runs twice faster than the GPU baseline simulator [15].

6 Conclusions and Future Work

PDP systems are a formal framework for ecosystem modelling, whose applications require efficient software simulators. In this concern, GPU-accelerated simulators have been developed so far. However, the designed algorithms for PDP systems, specially DCBA, have several bottlenecks. On the one hand, DCBA assumes that all rules in the system will depend on each other when consuming the left-hand sides. However, this is not always the case, and we can pre-compute partitions of rules actually competing for objects. On the hand, the model designer knows which rules will be applied at each step. An adaptative simulator should be able to use this information to dismiss rules that are known to be non applicable in a certain step.

We have shown how to modify DCBA main loop when implementing these two ideas (adaptative DCBA and DCBA with partitions). These strategies lead to extra speedups (compared with the GPU baseline simulator) of around 2x-2.5x. Let us remark that a simulator implementing these two modes should be used carefully:

- An adaptative simulator of PDP system should be used when deploying a validated model. That is, when the model is already refined and validated by the designer, and the behaviour is already known and proved to work. For example, when using the simulator in virtual experimentation environments.
- A simulator with partitions of PDP systems (e.g. μ-DCBA) can be used not only in virtual experimentation environments, but also in the validation of the model. That is, when the designer is still debugging the model, this strategy can help since it works automatically from the set of rules. However, the performance is not as good as with adaptative simulators (according to our preliminary results), and would require some initial pre-computation for partitions.

Future work includes the development of a μ-DCBA in a stable simulator along with the already existing adaptative simulator. Moreover, we plan to use these improvements to go to the next step and implement parallel parameter calibration methods for the models. We are also working on an automatic inclusion of the GPU simulators inside generic simulation tools such as MeCoSim and P-Lingua, since the only way to use these tools is by a manual protocol [18]. Finally, we are looking into further improvements of the GPU adaptative simulators by including features such as for object counters [2].

Acknowledgements

This work was supported by the research project TIN2017-89842-P (MABICAP), co-financed by *Ministerio de Economía, Industria y Competitividad (MINECO)* of Spain, through the *Agencia Estatal de Investigación (AEI)*, and by *Fondo Europeo de Desarrollo Regional (FEDER)* of the European Union.

References

1. Alhazov, A.: Maximally parallel multiset-rewriting systems: Browsing the configurations. In: Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 1–10. Fénix Editora, Seville, Spain (February 2005), `http://www.gcn.us.es/3BWMC/bravolpdf/bravol1.pdf`
2. Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Design of specific P systems simulators on GPUs. In: Hinze, T., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing. pp. 202–207. Springer International Publishing, Cham (2019)
3. Martínez-del Amor, M.A., Pérez-Hurtado, I., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Romero-Jiménez, Á., Graciani, C., Riscos-Núñez, A., Colomer, M.A., Pérez-Jiménez, M.J.: DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing. Lecture Notes in Computer Science, vol. 7762, pp. 257–276. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
4. Cardona, M., Colomer, M.A., Pérez-Jiménez, M.J., Sanuy, D., Margalida, A.: Modeling ecosystems using P systems: the bearded vulture, a case study. In: Corne, D., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 5391, pp. 137–156. Springer Berlin Heidelberg (2009)
5. Colomer, M., Pérez-Hurtado, I., Pérez-Jiménez, M., Riscos-Núñez, A.: Comparing simulation algorithms for multienvironment probabilistic P systems over a standard virtual ecosystem. Natural Computing 11(3), 369–379 (2012)
6. Colomer, M.A., Margalida, A., Valencia-Cabrera, L., Palau, A.: Application of a computational model for complex fluvial ecosystems: The population dynamics of zebra mussel Dreissena polymorpha as a case study. Ecological Complexity 20, 116 – 126 (2014)
7. Colomer, M., Margalida, A., Pérez-Jiménez, M.: Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools 8(5), e60698 (2013)
8. Colomer, M., Margalida, A., Sanuy, D., Pérez-Jiménez, M.J.: A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. Ecological Modelling 222(1), 33–47 (2011)
9. Colomer-Cugat, M.A., García-Quismondo, M., Macías-Ramos, L.F., Martínez-del Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A., Valencia-Cabrera, L.: Membrane System-Based Models for Specifying Dynamical Population Systems, pp. 97–132. Springer International Publishing, Cham (2014)
10. Doncel-Ramírez, A.: Simulación Acelerada de Sistemas P de Dinámica de Poblaciones con GPU. Master thesis (Universidad de Sevilla), july 2018
11. García-Quismondo, Manuel and Martínez-del-Amor, M.A., Pérez-Jiménez, M.J.: Probabilistic Guarded P Systems, A New Formal Modelling Framework. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing. pp. 194–214. Lecture Notes in computer Science, Springer International Publishing, Cham (2014)
12. Martínez-del-Amor, M., Macías-Ramos, L., Valencia-Cabrera, L., Pérez-Jiménez, M.: Parallel simulation of Population Dynamics P systems: updates and roadmap 15(4), 565–573 (2015)

13. Martínez-del-Amor, M., Pérez-Hurtado, I., Pérez-Jiménez, M., Riscos-Núñez, A., Colomer, M.: A new simulation algorithm for multienvironment probabilistic P systems. In: IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010). vol. 1, pp. 59–68 (September 2010)
14. Martínez-del-Amor, M.A., Karlin, I., Jensen, R.E., Pérez-Jiménez, M.J., Elster, A.C.: Parallel simulation of probabilistic P systems on multicore platforms. In: Proceedings of the Tenth Brainstorming Week on Membrane Computing. vol. II, pp. 17–26. Fénix Editora, Seville, Spain (February 2012), `http://www.gcn.us.es/10BWMC/10BWMCvolII/papers/parallel-dcba.pdf`
15. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Gastalver-Rubio, A., Elster, A.C., Pérez-Jiménez, M.J.: Population Dynamics P Systems on CUDA. In: Gilbert, D., Heiner, M. (eds.) Computational Methods in Systems Biology, pp. 247–266. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2012)
16. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Orellana-Martín, D., Pérez-Jiménez, M.J.: Adaptative parallel simulators for bioinspired computing models. Future Generation Computer Systems 107, 469 – 484 (2020)
17. Pérez-Hurtado, I., Orellana-Martín, D., Zhang, G., Pérez-Jiménez, M.J.: P-lingua in two steps: flexibility and efficiency. Journal of Membrane Computing 1(2), 93–102 (Jun 2019)
18. Valencia-Cabrera, L., Martínez-del Amor, M.Á., Pérez-Hurtado, I.: A Simulation Workflow for Membrane Computing: From MeCoSim to PMCGPU Through P-Lingua, pp. 291–303. Springer International Publishing, Cham (2018)
19. Zhang, G., Shang, Z., Verlan, S., Martínez-del-Amor, M.A., Yuan, C., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: An overview of hardware implementation of membrane computing models. ACM Comput. Surv. 53(4) (Aug 2020)

An optimal solution to the SAT problem with tissue P systems

David Orellana-Martín, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {dorellana, lvalencia, marper}@us.es

Summary. In the framework of membrane computing, several frontiers of efficiency have been found with respect to the resources that different families of P systems take to solve a decision problem. Each of these frontiers provides a new way to tackle the **P** versus **NP** problem. In this sense, optimal frontiers are needed in order to separate close variants of P systems. In a previous work, an efficient solution to SAT was given in the framework of P systems from $\mathcal{TDC}(3)$. In this work, we will provide an optimal solution to the SAT problem in terms of length of the rules.

Key words: Membrane Computing, tissue P systems, symport/antiport rules, SAT problem.

1 Introduction

One of the most prolific fields within the framework of Membrane Computing is computational complexity theory. The search for frontiers of efficiency has produced several results in terms of correspondences between classic computational complexity classes and membrane computing complexity classes. In particular, several results with tissue P systems [5] have been achieved. Recognizer tissue P systems were introduced in [6]. In [3], it was demonstrated that the complexity class of tissue P systems from $\mathcal{TDC}(1)$ are non-efficient systems through the dependency graph technique. In [6], an efficient solution to the SAT problem is given by means of a family of tissue P systems from $\mathcal{TDC}(5)$. In this work, we present an optimal solution for SAT with a family of tissue P systems from $\mathcal{TDC}(2)$. This implies that $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{TDC}(2)}$. However, this last result is not new, since in [9] it was presented a solution to HAM-CYCLE, a well-known **NP**-complete problem, with a family of P systems from $\mathcal{TDC}(2)$. The novelty of this work is to present an efficient SAT solver, given the relevance of the problem. The paper is organized as follows. In Section 2, some prerequisites about languages and sets are given. The next section is devoted to introduce tissue P systems, and the complexity classes associated to them. In Section 4, an efficient solution to `SAT` by means of tissue P systems from $\mathcal{TDC}(2)$ is given, and in the next section an overview of the computations is given. Finally, the work finishes with some conclusions.

2 Preliminaries

An *alphabet* Γ is a non-empty set whose elements are called *symbols*. A *string* u over $\Gamma$is an ordered finite sequence of symbols; that is, a mapping from a natural number $n \in \mathbb{N}$ onto Γ The number $n$is called the *length* of the string u and it is denoted by $|u|$ The empty string (with length 0) is denoted by λ. The set of all strings over an alphabet Γ is denoted by Γ^*. A *language* over Γ is a subset of Γ^*.

A *multiset* over an alphabet Γ is an ordered (Γ, f) where $f$is a mapping from Γ onto the set of natural numbers $\mathbb{N}$. The *support* of a multiset $m = (\Gamma, f)$ is defined as $supp(m) = \{x \in \Gamma \mid f(x) > 0\}$. A multiset is finite (respectively, empty) if its support is a finite (respectively, empty) set. We denote by $\emptyset$ the empty multiset. Let $m_1 = (\Gamma, f_1), m_2 = (\Gamma, f_2)$ be multisets over Γ, then the union of m_1 and m_2, denoted by $m_1 + m_2$, is the multiset (Γ, g) where $g(x) = f_1(x) + f_2(x)$ for each $x \in \Gamma$. The difference of m_1 and m_2, denoted by $m_1 \setminus m_2$, is the multiset (Γ, g) where $g(x) = f_1(x) - f_2(x)$ for each $x \in \Gamma$. We denote by $M(\Gamma)$ the set of all multisets over Γ.

The Cantor pairing function is used to encode two natural numbers into a single one, and is defined as follows: given $x, y \in \mathbb{N}, \langle x, y\rangle = \frac{(x+y+1)(x+y)}{2} + y$

3 Tissue P systems with symport/antiport rules

Definition 1. *A recognizer tissue P system with symport/antiport rules and division rules of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$, where:*

1. *Γ, Σ and $\mathcal{E}$ are finite alphabets such that $\Sigma, \mathcal{E} \subseteq \Gamma$ and $\Sigma \cap \mathcal{E} = \emptyset$.*
2. *$\mathcal{M}_1, \dots, \mathcal{M}_q$ are multisets over $\Gamma \setminus (\Sigma \cup \mathcal{E})$.*
3. *$\mathcal{R}$ is a set of rules of the following types:*
   (a) Communication rules: $(i, u/v, j)$, for $i, j \in \{0, 1, \dots, q\}, i \neq j, u, v \in M(\Gamma), |u| + |v| > 0$.
   (b) Division rules: $[\,a\,]_i \rightarrow [\,b\,]_i[\,c\,]_i$, for $i \in \{1, \dots, q\}, a, b, c \in \Gamma$.
4. *$i_{in} \in \{1, \dots, q\}$ and $i_{out} = env$.*

A recognizer tissue P system with symport/antiport rules and division rules $\Pi = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$ of degree $q \geq 1$ can be viewed as a se of q cells, labelled by $1, \dots, q$ with an environment labelled by 0 such that: (a)

$\mathcal{M}_1, \dots \mathcal{M}_q$are the multisets of objects initially placed in the q cells of the system; (b) $\mathcal{E}$ is the set of objects located initially placed in the environment of the system, all of them appearing in an arbitrary number of copies; and (c) $i_{in} \in \{1, \dots, q\}, i_{out} = env$ represent distinguished cells where objects of the instance are introduced and which will encode the output of the system, respectively.

When applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and, simultaneously, the objects of multiset v are sent from region j to region i. The length of the communication rule $(i, u/v, j)$ is defined as $|u| + |v|$, that is, the total number of objects which appear in the rule.

A communication rule $(i, u/v, j)$ is called a symport rule if $u = \lambda$ or $v = \lambda$. A symport rule $(i, u/\lambda; j)$, with $i \neq 0, j \neq 0$ provides a virtual arc from cell i to cell j. A communication rule $(i, u/v, j)$ is called an antiport rule if $u \neq \lambda$ and $v \neq \lambda$. An antiport rule $(i, u/v, j)$, with $i \neq 0, j \neq 0$, provides two arcs: one from cell i to cell j and another one from cell j to cell i. Thus, every tissue P systems has an underlying directed graph whose nodes are the cells of the system and the arcs are obtained from communication rules. In this context, the environment can be considered as a virtual node of the graph such that their connections are defined by communication rules of the form $(i, u/v, j)$, with $i = 0$ or $j = 0$.

When applying a division rule $[\,a\,]_i \to [\,b\,]_i[\,c\,]_i$, cell i is duplicated into two new cells with the same label. Object a dissapears and an object b is created in the first new cell and an object c is created in the second new cell. The rest of the objects within the cell i are duplicated in the two new cells.

The rules of a system like the one above are used in a non-deterministic maximally parallel manner as it is customary in Membrane Computing. At each step, all communication rules which can be applied will be applied in a maximally parallel way (at each step we apply a multiset of rules which is maximal, no further applicable rule can be added). Only a single division rule can be applied to each cell. If a division rule is applied to a cell, we say that this cell is is "blocked", and no communication rules can be applied to that cell in that computational step.

An instantaneous description or a configuration at any instant of a tissue P system is described by all multisets of objects over Γ associated with all the cells present in the system, and the multiset of objects over $\Gamma \setminus \mathcal{E}$ associated with the environment at that moment. Bearing in mind that the objects from $\mathcal{E}$ have infinite copies in the environment, they are not properly changed along the computation. The initial configuration is $(\mathcal{M}_1, \dots, \mathcal{M}_q; \emptyset)$. A configuration is a halting configuration if no rule of the system is applicable to it.

Let us fix a tissue P system with symport/antiport rules Π. We say that configuration $\mathcal{C}_i$ yields configuration $\mathcal{C}_{i+1}$ in one transition step, denoted $\mathcal{C}_i \Rightarrow_{\Pi} \mathcal{C}_{i+1}$, if we can pass from $\mathcal{C}_i$ to $\mathcal{C}_{i+1}$ by applying the rules from $\mathcal{R}$ following the previous remarks. A computation of Π is a (finite or infinite) sequence of configurations such that: (a) the first term of the sequence is an initial configuration of the system; (b) each non-initial configuration of the sequence is obtained from the previous configuration by applying the rules of the system in a maximally parallel manner with the restrictions previously mentioned; and (c) if the sequence is finite (called halting computation), then the last term of the sequence is a halting configuration. All computations start from an initial configuration and proceed as stated above;only halting computations give a result, which is encoded by the objects present in the environment in the halting configuration. Given that they are *recognizer* P systems, all computations halt and return the same output. We denote a recognizer membrane system Π with the input multiset m in the input region as $\Pi + m$.

3.1 Complexity classes associated to tissue P systems

Let $\mathcal{R}$ be a class of recognizer membrane systems. We say that $\mathbf{PMC}_{\mathcal{R}}$ is the class of problems solvable efficiently in a uniform way by means of a family of recognizer membrane systems from $\mathcal{R}$. The class of recognizer tissue P systems with symport/antiport rules with length at most k and division rules is denoted by $\mathcal{TDC}(k), k \geq 1$. For more information about computational complexity theory in the framework of membrane computing, we refer the reader to [7, 8].

4 A solution to SAT in $\mathcal{TDC}(2)$

In this section, an efficient solution to the SAT problem by means of a family of P systems will cell division and symport/antiport rules of length at most 2 is presented.

For each pair of natural numbers $n, p \in \mathbb{N}$, we will consider the recognizer tissue P system with cell division and symport/antiport rules

$$\Pi(\langle n, p\rangle) = (\Gamma, \mathcal{E}, \Sigma, \mathcal{M}_1, \dots, \mathcal{M}_{np+3}, \mathcal{R}, i_{in}, i_{out})$$

of degree $np + 3$ defined as follows:

(a) $\Gamma = \Sigma \cup \mathcal{E} \cup \{\texttt{yes}, \texttt{no}, \alpha, \beta_0, \gamma_0\} \cup \{c_j \mid 1 \leq k \leq p\} \cup \{\alpha_k \mid 0 \leq k \leq np-1\} \cup \{a_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq p\} \cup \{T_{i,j}, F_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq p\} \cup \{x_{i,j,k}, \overline{x}_{i,j,k} \mid 1 \leq i \leq n, 1 \leq j \leq p, 0 \leq k \leq np\}$

(b) $\mathcal{E} = \{\alpha_k \mid np \leq k \leq 2np+2\} \cup \{\beta_k \mid 1 \leq k \leq 2np+4\} \cup \{\gamma_k \mid 1 \leq k \leq 2np+5\} \cup \{x_{i,j,k}, \overline{x}_{i,j,k} \mid 1 \leq i \leq n, 1 \leq j \leq p, np+1 \leq k \leq 2np\}$

(c) $\Sigma = \{x_{i,j,0}, \overline{x}_{i,j,0} \mid 1 \leq i \leq n, 1 \leq j \leq p\}$

(d) $\mathcal{M}_i = \emptyset$ for $1 \leq i \leq np$, $\mathcal{M}_{np+1} = \{\texttt{yes}, \texttt{no}, \beta_0, \gamma_0\}$, $\mathcal{M}_{np+2} = \{a_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq p\} \cup \{c_j \mid 1 \leq j \leq p\} \cup \{\alpha\}$, $\mathcal{M}_{np+3} = \{\alpha_0\}$.

(e) The set of rules $\mathcal{R}$ is the following:

1. Rules to generate p copies of the 2^n true possible truth assignments. For this, 2^{np} partial truth assignments will be generated.

$$\left.\begin{array}{l} [\,a_{i,j}\,]_{np+2} \rightarrow [\,T_{i,j}\,]_{np+2}[\,F_{i,j}\,]_{np+2} \\ (np+2, T_{i,j}F_{i,j'}, /\lambda, env) \end{array}\right\} \text{for } 1 \leq i \leq n, 1 \leq j, j' \leq p$$

2. Rules to generate 2^{np} copies of $cod(\varphi)$.

$$\left.\begin{array}{l}(np+1, x_{i,j,0}/\lambda, i+n\cdot(j-1))\\(np+1, \overline{x}_{i,j,0}/\lambda, i+n\cdot(j-1))\end{array}\right\}\text{for }\begin{array}{l}1\le i\le n,\\1\le j\le p\end{array}$$

$$\left.\begin{array}{l}[\,x_{i,j,k}\,]_{i+n\cdot(j-1)}\rightarrow[\,x_{i,j,k+1}\,]_{i+n\cdot(j-1)}[\,x_{i,j,k+1}\,]_{i+n\cdot(j-1)}\\ [\,\overline{x}_{i,j,k}\,]_{i+n\cdot(j-1)}\rightarrow[\,\overline{x}_{i,j,k+1}\,]_{i+n\cdot(j-1)}[\,\overline{x}_{i,j,k+1}\,]_{i+n\cdot(j-1)}\end{array}\right\}\text{for }\begin{array}{l}1\le i\le n,\\1\le j\le p,\\0\le k\le np-1\end{array}$$

$$\left.\begin{array}{l}(i+n\cdot(j-1), x_{i,j,k}/x_{i,j,k+1}, env)\\(i+n\cdot(j-1), \overline{x}_{i,j,k}/\overline{x}_{i,j,k+1}, env)\end{array}\right\}\text{for }\begin{array}{l}1\le i\le n,\\1\le j\le p,\\np\le k\le 2np-1\end{array}$$

3. Rules to check which clauses are satisfied by the truth assignment

$$\left.\begin{array}{l}(np+2, T_{i,j}/x_{i,j,2np}, i+n\cdot(j-1))\\(np+2, F_{i,j}/\overline{x}_{i,j,2np}, i+n\cdot(j-1))\end{array}\right\}for\ \begin{array}{l}1\le i\le n,\\1\le j\le p\end{array}$$

$$\left.\begin{array}{l}(np+2, c_j x_{i,j,2np}/\lambda, env)\\(np+2, c_j \overline{x}_{i,j,2np}/\lambda, env)\end{array}\right\}for\ 1\le i\le n, 1\le j\le p$$

$[\,\alpha_k\,]_{np+3}\rightarrow[\,\alpha_{k+1}\,]_{np+3}\,[\,\alpha_{k+1}\,]_{np+3}\,\}\ for\ 0\le k\le np-1$

$(np+3, \alpha_{np+k}/\alpha_{np+k+1}, env)\ for\ 0\le k\le np+1$

$(np+1, \beta_k/\beta_{k+1}, env)\ for\ 0\le k\le 2np+3$

$(np+1, \gamma_k/\gamma_{k+1}, env)\ for\ 0\le k\le 2np+4$

$(np+2, \alpha/\alpha_{2np+2}, np+3)$

$(np+2, \alpha_{2np+2}c_j/\lambda, env)\ for\ 1\le j\le p$

4. Rules to return a negative answer

$(np+1, \beta_{2np+4}, \gamma_{2np+5}/\lambda, np+3)$

$(np+1, \texttt{no}/\beta_{2np+4}, np+3)$

$(np+3, \texttt{no}/\lambda, env)$

5. Rules to return a positive answer

$(np+1, \beta_{2np+4}/\alpha_{2np+2}, np+2)$

$(np+1, \alpha_{2np+2}\texttt{yes}/\lambda, env)$

(f) $i_{in} = np+1$ and $i_{out} = env$

5 An overview of the computations

In this section, we will explain a brief overview of the computations.

Let φ be a propositional logic formula in conjunctive normal form, where $Var(\varphi) = \{x_1, \ldots, x_n\}$. Then, φ is of the form $\varphi = C_1 \wedge \ldots \wedge C_p$, where C_j is a clause such that $C_j = l_{1,j} \vee \ldots l_{p_j,j}, l_{i,j} \in \{x_i, \neg x_i\}$. The pair (cod, s) for this family is $cod(\varphi) = \{x_{i,j} \mid x_i \in C_j\} \cup \{\overline{x}_{i,j} \mid \neg x_i \in C_j\}$, and $s = \langle n, p\rangle$.

For that, let us remember that the input is an instance of the SAT problem. Let φ the input formula with n variables and p clauses. The tissue P system that will give the answer to the instance is $\Pi(\langle n, p\rangle) + cod(\varphi)$.

5.1 Generation stage

The first $2np$ steps will be devoted to generate both the 2^n possible truth assignments. For this, rules from **1** and **2** are used. On the one hand, With rules from **1**, 2^{np} cells with label $np + 2$ with different truth assignments. The second rule is fired in order to remove incompatible truth assignments in the following sense: if two different assignments are given to the same variable, then the corresponding objects are sent to the environment. In this way, the incompatible assignments are removed and the remaining objects will be equivalent to a valid truth assignment.

On the other hand, rules from **2** will create 2^{np} copies of each object from $cod(\varphi)$. Being $l_{i,j,0} \in cod(\varphi)$, at the end of the stage, there will be 2^{np} cells $i + n \cdot (j-1)$ $(1 \leq i \leq n, 1 \leq j \leq p)$ with an object $l_{i,j,2np}$ in the corresponding cell.

Besides these rules, 2^{np} cells with label $np + 3$ with an object α_{2np+2} will be generated with rules from **3**. This stage will take $2np$ steps.

5.2 First checking stage

In this stage, clauses validated by the truth assignments are going to be analyzed. For this, rules from **3** are used. In particular, in the first step of the stage objects $T_{i,j}$ and $F_{i,j}$ are interchanged with objects $x_{i,j,2np}$ and $\overline{x}_{i,j,2np}$, respectively. Then, objects $x_{i,j,2np}$ and $\overline{x}_{i,j,2np}$ will represent that the literal l_i makes true clause C_j. Then, if an object $x_{i,j,2np}$ or $\overline{x}_{i,j,2np}$ exists in a cell labelled by $np + 2$, it will "remove" the corresponding object c_j, sending it to the environment. When the stage is over, cells labelled by $np + 2$ will contain only the objects c_j such that the clause C_j has not been satisfied by the corresponding truth assignment. This stage takes 2 steps.

5.3 Second checking stage

The satisfiability of the input formula φ will be analyzed in this stage. Last two rules from **3** will be used. In the first step, object α from cells labelled by $np + 2$ will be interchanged with object α_{2np+2} from cells labelled by $np + 3$. Object α_{2np+2} in cells labelled by $np + 3$ is a mark to know if there are remaining objects c_j. With the last rule, any remaining object c_j will "remove" object α_{2np+2} from the corresponding cell $np + 2$. Therefore, if a truth assignment does not satisfy the whole formula φ, object α_{2np+2} will not be present in the corresponding cell $np + 2$. This stage takes 2 steps.

5.4 Output stage

The output stage starts at the $2np + 5$ step, and takes 4 steps in the negative case and 2 steps in the affirmative case.

- *Affirmative answer*: In this case, there will exist a cell labelled by $np+2$ that will have an object α_{2np+2}, as it represents a truth assignment that makes true the input formula φ. With the application of the first rule from **5**, it will be interchanged by the object β_{2np+4} from cell $np+1$. At the same time, object γ_{2np+5} will go inside cell labelled by 1. Since object β_{2np+4} has been moved from cell $np+2$, they will not interact with each other. In the next step, as object α_{2np+2} is present in the cell $np+1$, it will send object `yes` to the environment. Then, the computation ends.
- *Negative answer*: In this case, all objects α_{2np+2} will be in the environment, as there is no truth assignment such that it makes true all clauses from φ. Therefore, in the previous stage there was at least one object c_j in each cell labelled by $np+2$ and it will send object α_{2np+2} to the environment. In the first step of this stage, object γ_{2np+5} will go into cell labelled by $np+1$. In the next step, objects β_{2np+4} and γ_{2np+5} will interact and be sent to a cell labelled by $np+3$. Following that, object `no` will be interchanged with the object β_{2np+4}, and then object `no` will be sent to the environment. The computation stops here.

Theorem 1. $\texttt{SAT} \in \mathbf{PMC}_{\mathcal{TDC}(2)}$

Proof. The family of P systems previously constructed verifies the following:

- Every system of the family $\mathbf{\Pi}$ is a recognizer P system from $\mathcal{TDC}(2)$.
- The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines because for each $n,p \in \mathbb{N}$, the rules of $\Pi(\langle n,p\rangle)$ of the family are recursively defined from $n,p \in \mathbb{N}$, and the amount of resources needed to build an element of the family is of a polynomial order in n and p, as shown below:
  - Size of the alphabet: $4n^2p^2 + 10np + p + 17 \in \Theta(n^2p^2)$.
  - Initial number of cells: $np+3 \in \Theta(np)$.
  - Initial number of objects in cells: $np+p+6 \in \Theta(np)$.
  - Number of rules: $2n^2p^2 + np^2 + 11np + p + 7 \in \Theta(n^2p^2)$.
  - Maximal number of objects involved in any rule: $2 \in \Theta(1)$.
- The pair (cod, s) of polynomial-time computable functions defined fulfills the following: for each input formula φ of SAT problem, $s(\varphi)$ is a natural number, $cod(\varphi)$ is an input multiset for the system $\Pi(s(\varphi))$, and for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set.
- The family $\mathbf{\Pi}$ is polynomially bounded: indeed, for each input formula φ of SAT problem, the deterministic P system $\Pi(s(\varphi))+cod(\varphi)$ takes exactly $np+7$ steps, being n the number of variables in φ and p its number of clauses.
- The family $\mathbf{\Pi}$ is sound with regard to (X, cod, s): for each formula φ, if the computation of $\Pi(s(\varphi)) + cod(\varphi)$ is an accepting computation, then φ is satisfiable.
- The family $\mathbf{\Pi}$ is complete with regard to (X, cod, s): for each input formula φ such that it is satisfiable, the computation of $\Pi(s(\varphi))+cod(\varphi)$ is an accepting computation.

Corollary 1. $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{\mathcal{TDC}(2)}$

6 Conclusions

In the framework of membrane computing, as mentioned above, we search for frontiers of efficiency as new ways to attack the **P** versus **NP** problem. For that, it is necessary to have both results of non-efficient classes and efficient classes of membrane systems. The thinner the frontier, the easier would be to try to adapt an efficient solution from the efficient model to the non-efficient model.

The SAT problem is the best-known **NP**-complete problem, since it was the first problem to be demonstrated to be **NP**-complete [1, 4], and therefore is one of the most studied problems to solve the conjecture. SAT solvers are systems implemented to give an answer to an input SAT instance [2]. Implementations on high-performance computing platforms could be useful to simulate this solution since it could provide a good alternative to current industrial SAT solvers. More precisely, this solution only requires of 2 objects at most in communication rules, that could be an advantage in the implementation. As future work we will continue studying the efficiency of different variants of recognizer membrane systems.

Acknowledgements

This work was supported in part by the research project TIN2017-89842-P, co-financed by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

References

1. S. Cook. The complexity of theorem proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 1971, pp. 151–158. (doi:10.1145/800157.805047).
2. C.P. Gomes, H.A. Kautz, A. Sabharwal, B. Selman. Satisfiability Solvers. *Handbook of Knowledge Representation* (2008).
3. R. Gutiérrez-Escudero, M.J. Pérez-Jiménez, M. Rius-Font. Characterizing tractability by tissue-like P systems. *Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges*, Romania, August 24-27, 2009, Revised Selected and Invited Papers. Lecture Notes in Computer Science, **5957** (2010), 289-300 (doi:10.1007/978-3-642-11467-0_21) A preliminary version in Gh. Paun, M.J. Pérez-Jiménez, A. Riscos (eds.) *Proceedings of the Tenth Workshop on Membrane Computing*, Curtea de Arges (Romania), August 24-27, 2009, pp. 269-281.
4. R.M. Karp. Reducibility Among Combinatorial Problems. In R.E. Miller; J.W. Thatcher (eds.) *Complexity of Computer Computations* (1972). New York: Plenum. pp. 85-103. ISBN 0-306-30707-3.
5. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón. A new class of symbolic abstract neural nets: tissue P systems. *Lecture Notes in Computer Science*, **2387** (2002), pp. 290-299

6. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez. Tissue P systems with cell division. *International Journal of Computers, Communications & Control*, **3**, 3 (2008), pp. 295-303
7. M.J. Pérez-Jiménez. An Approach to Computational Complexity in Membrane Computing. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (eds.) *Membrane Computing. WMC 2004. Lecture Notes in Computer Science*, 3365 (2005). Springer, Berlin, Heidelberg, pp. 125-148.
8. M.J. Pérez–Jiménez. A Computational Complexity Theory in Membrane Computing. In Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, A. Salomaa (eds.) *Membrane Computing. WMC 2009. Lecture Notes in Computer Science*, 5957 (2010). Springer, Berlin, Heidelberg, pp. 85-109.
9. A.E. Porreca, N. Murphy, M.J. Pérez-Jiménez. An Optimal Frontier of the Efficiency of Tissue P Systems with Cell Division. In M.Á. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, F.J. Romero-Campero (eds.) *Proceedings of the Tenth Brainstorming Week on Membrane Computing*, Report RGNC 01/2012, 2012, pp. 141-166.

Seven Research Suggestions

Gheorghe Păun

Romanian Academy, Bucharest, Romania
curteadelaarges@gmail.com

Summary. Some rather general research suggestions in membrane computing, as well as a couple of more specific ideas are formulated.

1 Introduction

At more than two decades since membrane computing (MC) was initiated, the achievements in this research area, in terms of publications, PhD theses, collective volumes, monographs, applications and software are considerable – see a detailed overview of membrane computing in the CMC20 talk of Gexiang Zhang, "Membrane Computing: Developmental Analysis" (to be published in *Journal of Membrane Computing*).

By the way, a really important issue in this moment for the membrane computing community is to "help growing" our journal. I would formulate it in a short "triadic form": *write, read, cite*! Write and submit papers to JMC, efficiently participate in reviewing papers for JMC, promote the journal, especially (for this period, before getting an impact factor from ISI) citing papers published in JMC. Of a great help in this last direction is *Bulletin of IMCS* (accessible at `http://membranecomputing.net/IMCSBulletin/`), where the contents of JMC is recalled.

Coming back to the goal of the present notes, as said in the Abstract, some suggestions are formulated, some of them rather general and only a few are more specific. As the field is really mature, no prerequisites are provided and almost no references, except those really necessary.

2 Two Very General Ideas

The first suggestion is somewhat classic and trivial: *back to literature!*

(Q1) There are plenty of open problems and research topics formulated in the MC literature. Some of these problems were solved, some of these research vistas were explored, many others might be now obsolete, of no much interest, but many enough still wait for research efforts.

For instance, I would like to recall the attention about the problems collected in [2], also available in a preliminary form in a Brainstorming volume, [1]. I believe that a nice and useful analysis would be to systematically examine the proposals from this paper and see the status of each of them, thus revealing the topics which need/deserve our attention from now on.

(Q2) The previous idea was to look to the past – now I would like to suggest to look to the future... With the mentioning that *the future started yesterday...* It is about *the Fourth Industrial Revolution.* The key-words describing it are of the kind: connectivity, artificial intelligence, machine learning, cyber-systems, robots.

Which of these syntagma are "reachable" by MC, which were already addressed, which suggestions can we get from this direction? Very general questions, but a good answer can have rather positive consequences. What about a "very distributed" P system/colony, about swarms of membranes? Much work is still needed in the learning (deep learning?) direction. Both these issues can have nice practical applications (the same with evolutionary computing, membrane algorithms and connected areas, making use of the "brute force" brought into the stage by the complex systems of weak components cooperating in a cleaver manner).

3 Three "Hybridization" Suggestions

Suggestions of the forms bellow were formulated many times, in more general or more specific terms. I recall them, with some further details.

(Q3) Systematic comparison of "basic" classes of P systems – cell-like, tissue-like, spiking neural, and numerical, with multiset rewriting rules, active membranes, symport/antiport, spiking rules, programs (production-repartition) rules, respectively, with various specific features – catalysts, polarizations, regular expression guarding the (spiking) rules, unique object (the spike), etc.

Many combinations of these ingredients were considered – but not all of them and not in a systematic manner.

We know, for instance, the power (also the efficiency?) of one-object cell-like P system, or of cell-like SN P systems, but I no not remember papers examining numerical P systems using only one variable (in each region), or using the notion of anti-matter.

A biological detail which is not satisfactorily captured in SN P systems is the sigmoidal function involved in the spiking operation. Maybe by borrowing the way of evolving variables in numerical P systems and using "programs" (again: with two parts, producing and then distributing) in SN P systems one can obtain something of interest.

(Q4) Bridging P and R was requested for several times and there are some attempts in this direction, but for sure much more remains to be done. The two areas are rather connected (cell-structure, evolution rules, biochemical metaphor), but they also differ in essential details (multisets, active membranes, symport/antiport and spiking rules, etc. in MC, zero or arbitrarily large multiplicity, no-surviving principle, different goals than computing, etc., in reaction systems). Borrowing notions investigated in R and investigating them for P (which of them make sense? which of them are decidable?) was suggested many times.

Let me formulate one really "hybridization" idea: in R systems, the evolution is influenced by the environment, which provides arbitrary (multi)sets of objects to the system; what about having these objects produced by a P system – or by several P systems. The idea is illustrated in Figure 1. Of course, the P systems work "in the MC style" (multisets, maximal parallelism, objects that do not evolve persist, etc.) while R is a reaction system. Plenty of questions appear: examine the usual R questions for such a hybrid system; what about computability in this framework? (the first question is how to define the result of a computation); what about the case when the P systems not only send objects to the environment, but they can also bring objects back inside? what about using simple P systems (non-universal), or of various types? in the case of SN P systems, we will have two possibilities: to distinguish between the spikes of various SN P systems or not – in the latter case, the R system is supposed to get only one (type of) object from the environment; how R systems with only one object in their alphabet behave?

Find other types of P-R hybrid systems.

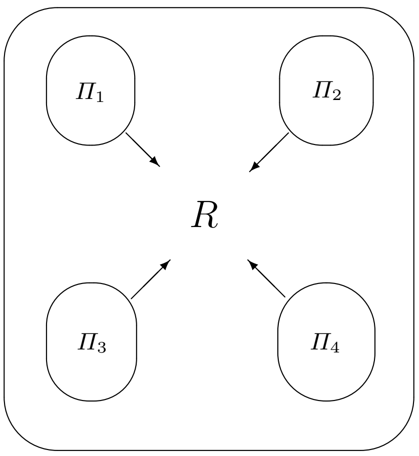


Fig. 1.

(Q5) A similarly promising topic, partially, but not enough explored, is that of bringing to MC further notions from the quantum area.

Two immediate ideas are the following.

1. To consider P systems with *qobjects*, objects having a *name* and a *probability* associated, a number between 0 and 1: $(a, \alpha), a \in A, 0 \leq \alpha \leq 1$. Taking α as a "standard" probability does not seem to be very productive (there are some attempts of this kind). Maybe processing the objects with rules of the form

$$a \to (b, \beta)(c, \gamma),\ \ \beta, \gamma \in [-1, 1],$$

with the effect

$$(a, \alpha) \to (b, \alpha \oplus \beta)(c, \alpha \oplus \gamma),\ \text{where}$$
$$\alpha \oplus \delta = \begin{cases} 0, & \text{if } \ \alpha + \delta < 0, \\ \alpha + \delta, & \text{if } \ 0 \leq \alpha + \delta \leq 1, \\ 1, & \text{if } \ \alpha + \delta > 1, \end{cases}$$

might be more interesting. Maybe also a multiplicative operation can be considered. How to define a successful computation? By halting? And which could be the result of a computation? (Maybe the distance between two prescribed events, without halting, maybe the string of objcts which reach probability 1.) Should the objects of the form $(a, 0)$ be preserved in the system or they should be eliminated?

2. The second idea refers to objects as well, but also to their evolution: entanglement. Define objects which have identical evolution, irrespective where they are placed. There is no obvious definition – e.g., in the case of cooperative rules. Should entanglement be hereditary? (Copies of the same object, having a common ancestor, should be necessarily entangled?)

A good definition is the first step – after that, questions about computing power and efficiency are to be formulated.

Is entanglement a further door towards efficiency? (Entanglement means, in some sense, sending signals at an arbitrary distance in no time, which looks to be a powerful operation.) Maybe entanglement combined with the idea of qobjects? Maybe also imitating efficiency ideas from quantum computing?

4 Two More Precise Proposals

(Q6) There is a fundamental feature of P systems which, in some sense, is departing from the (bio)chemistry: the localization of evolution rules. In theoretical-abstract terms, *the (bio)chemistry is the same everywhere*, the "dictionary" of reactions is unique. What is applicable-active in a given "reactor" (compartment of a cell) is selected according to the local reactants, enzymes, catalysts, promoters and inhibitors, as well as according to the reaction conditions (e.g., temperature). This directly leads to the idea of *homogeneity*, of considering P systems, of any kind, with the same set of rules in each compartment.

The idea was investigated for many classes of P systems, but not for numerical P systems. I am also not aware of efficiency results for homogeneous P systems.

The same for the various ways of using the rules (semantics): maximally parallel, sequential, minimally parallel – whatever definition for these notions is chosen.

What about a sort of an additional "uniform" restriction of the following form: if P is the homogeneous set of rules present in all compartments (instead of using different rules in different compartments, depending on the local reactants and "reaction conditions"), choose a subset $P' \subseteq P$ (maximal?) and use it (in the maximally parallel way, etc.) in all compartments.

Power and efficiency results should be looked for.

(Q7) Still more specific is the last question: consider SN P systems with astrocytes producing calcium, with calcium directly involved in the spiking activity.

Formally, the system will contain two types of cells,

$$\begin{array}{rl} \textit{astrocytes} & \alpha_1, \ldots, \alpha_m, \text{ of the form } (c^{p_{i,0}}, A_i), p_{i,0} \geq 0, \\ & \text{with the rules in } A_i \text{ of the form } E_c/c^s \to c^t, s \geq 1, t \geq 0, \text{ and} \\ \textit{neurons} & \sigma_1, \ldots, \sigma_n, \text{ of the form } (a^{r_{i,0}}, R_i), r_{i,0} \geq 0, \\ & \text{with the rules in } R_i \text{ of the form } E_a/a^s c^{s'} \to a^t, s, s' \geq 1, t \geq 0, \end{array}$$

where E_c, E_a are regular expressions over the one-letter alphabets consisting of c and a, respectively.

The idea is clear: producing t spikes in a neuron means consuming both s spikes and s' calcium units.

The synapses should link either astrocytes or neurons, as well astrocytes to neurons (but not conversely: links (σ_i, α_j) are not permitted).

Of course, versions are possible: with the regular expressions in neurons also depending on the calcium units (hence over the alphabet $\{a, c\}$), with delay, with or without the possibility of replicating calcium, when an astrocyte sends objects c to several neurons. Now, the whole investigation program usual in the SN P systems area should be explored: normal forms, universality, small universal systems, plasticity, homogeneity, etc. Are astrocytes of this form improving the results known for usual SN P systems?

5 Final Remarks

This note had two main goals: to show that still there is much work to do in membrane computing, even at this basic level (not to speak about applications, which is by far the most promising and most important direction of research at this stage), and to recall again and again that a very important task of all of us at this moment is to... *write-read-cite*, supporting our journal JMC!...

Acknowledgements. This work was supported by the research project TIN2017-89842-P (MABICAP), co-financed by *Ministerio de Economía, Industria y Competitividad (MINECO)* of Spain, through the *Agencia Estatal de Investigación (AEI)*, and by *Fondo Europeo de Desarrollo Regional (FEDER)* of the European Union.

References

1. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez (Eds.) Frontiers of Membrane Computing. Open Problems and Research Topics, *Proc. 10th BWMC*, Sevilla Univ., 2012, vol. 1, 171–249.
2. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg (Eds.) Frontiers of Membrane Computing. Open Problems and Research Topics, *Intern. J. Found. Computer Sci.*, 24, 5 (2013), 547–623.

Some open problems for application of spiking neural P systems

Hong Peng

School of Computer and Software Engineering,
Xihua University, Chengdu, China
E-mail: ph.xhu@hotmail.com

Recently, three variants of spiking neural P systems have been proposed: coupled neural P systems (CNP systems) [1], dynamic threshold neural P systems (DTNP systems) [2] and nonlinear spiking neural P systems (NSNP systems) [3]. It was proven that the three variants are Turing universal number generating/accepting devices and function computing devices. The potential motivation of proposing the variants is to provide a modeling tool for real-life applications, for example, image processing tasks. Some open problems related to the variants are listed as follows.

Q1 As stated in the existing SNP systems, some problems that refer to these variants can be investigated, for example, language generator, sequential and asynchronous modes.

Q2 CNP systems and DTNP systems have all or part of spiking mechanism, coupling mechanism and dynamic threshold mechanism. How to apply the two variants to deal with some image processing tasks? for example, image fusion, image segmentation, object segmentation, feature extraction, object detection, and so on. Since image is two-dimensional (or three-dimensional for color image), CNT (or DTNP) systems can be considered as two- dimensional (or three-dimensional) array of neurons. Moreover, due to local spatial characteristics of an image, local topological structure should be further considered.

Q3 NSNP systems has a nonlinear spiking mechanism. Potentially, NSNP systems as a modelling tool could have the ability to handle nonlinear problems. Similarly, how to apply the variant to deal with these image processing tasks?

Q4 Local convolutional structures are easily introduced into these variants with local topological structure, like convolutional neural networks (CNN). How to use them to build deep SNP systems? How to develop the corresponding learning algorithms?

References

1. H. Peng, J. Wang. Coupled neural P systems. *IEEE Transactions on Neural Networks and Learning Systems*, 2019, 30(6), 1672-1682.
2. H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez. Dynamic threshold neural P systems. *Knowledge-Based Systems* 163, 2019, 875–884.
3. H. Peng, Z. Lv, B. Li, X. Luo, J. Wang, X. Song, T. Wang, M.J. Pérez-Jiménez, A. Riscos- Núñez. Nonlinear spiking neural P systems, *International Journal of Neural Systems*, 2020. https://doi.org/10.1142/S0129065720500082

Modelling of Grey Wolf Optimization Algorithm Using 2D P Colonies

Daniel Valenta[1], Lucie Ciencialová[1,2], and Luděk Cienciala[1,2]

[1] Institute of Computer Science, Silesian University in Opava, Czech Republic
[2] Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Czech Republic
{daniel.valenta, lucie.ciencialova, ludek.cienciala }@fpf.slu.cz

Summary. In this paper, we investigate a possibility of Grey wolf optimization algorithm simulation by 2D P colonies. We introduce a new kind of 2D P colony equipped with a blackboard. It is used by agents to store information that is reachable by all the agents from every place in the environment.

Key words: 2D P colonies, blackboard, Grey wolf optimization algorithm.

1 Introduction

2D P colonies are kind of P colonies, very simple membrane systems inspired by colonies of formal grammars. The interested reader is referred to [8] for detailed information on membrane systems (P systems) and to [4] and [3] for more information to grammar systems theory. For more details on P colonies consult the survey [2].

2D P colony consists of a finite number of agents - finite collections of objects in a cell - and their joint shared environment. The environment of 2D P colony is represented by a 2D grid of square cells. In each cell, there is a multiset of objects. The agents have programs consisting of rules. These rules are of three types: they may change the objects of the agents and they can be used for interacting with the joint shared environment of the agents and movement rule. The direction of the movement of the agent is determined by the contents of cells surrounding the cell in which the agent is placed. The program can contain at most one motion rule. To achieve the greatest simplicity in agent behaviour, one other condition was set. If the agent moves, it cannot communicate with the environment. So if the program contains a motion rule, then the other rule is an evolution rule. The number of objects inside each agent is set by definition and it is usually a very small number: 1, 2 or 3.

When the agent is moving around the 2D environment it has no information about the states and placements of the other agents. For remote information exchange, we add the agents the possibility to store and read the information from a blackboard. It is a table with an unchangeable structure given by definition. Agents can change values inside cells but not captions and the number of rows or columns.

Grey wolf optimization algorithm (GWO) is a meta-heuristic optimization technology. Its principle is to imitate the behaviour of grey wolves in nature to hunt cooperatively. Four types of grey wolves such as alpha, beta, delta, and omega are used for simulating the leadership hierarchy. In addition, the three main steps of hunting, searching for prey, encircling prey, and attacking prey, are implemented. The algorithm was introduced by Mirjalili et al. in 2014 in [9].

2 Grey wolf optimization algorithm

This section is to explain the way the GWO works. Grey wolf optimization algorithm is inspired by social dynamics found in packs of grey wolves and by their ability to dynamically create hierarchies in which every member has a clearly defined role. We distinguish the following wolves:

- *Alpha* male and female make up the dominant pair. The pack follows their lead during hunts, while locating a place to sleep, and so on. The most important attributes are their organisational abilities and discipline.
- *Beta* wolves support and respect the Alpha pair during its decisions.
- *Delta* wolves are subservient to Alpha and Beta wolves, follow their orders, and control Omega wolves. There are three types of Delta wolves: *scouts* – they observe the surrounding area and warn the pack, *sentinels* – they protect the pack when endangered and *caretakers* – they provide aid to old and sick wolves.
- *Omega* wolves help to filter the pack's aggression and frustrations by serving as scapegoats.

In GWO, the agents (wolves) primary goal in its environment is to find and hunt down prey, which in our case equals finding the optimal solution to the given problem. The environment is represented by a mathematical fitness function characterising the specific problem. A value found at the current position of the wolf refers to the highest-quality prey located. The wolf with the best value is ranked as Alpha, the second as Beta, third as Delta, and all the other as Omega.

The hunting technique of a wolf pack can be described in 5 steps:

- *Search for prey* (point A in Fig. 2.) – wolves are attempting to find the most valuable prey with respect to the effort required to successfully hunt it.
- *Exploitation of prey* (point B in Fig. 2.) – wolves are attempting to draw attention to themselves and to separate the prey from its herd.
- *Encircling prey* (point C in Fig. 2.) – the attempt to push the prey into a situation it cannot escape from.
- *The prey is surrounded* (point D in Fig. 2.) – it can no longer escape.

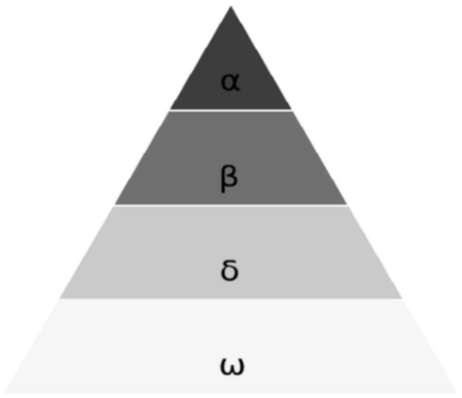


Fig. 1. wolf pack hierarchy

- *Attack* (point E in Fig. 2.) – wolves attack the prey's weak spots (belly, legs, snout) until it succumbs to fatigue. Afterwards, they bring it down and crush its windpipe.

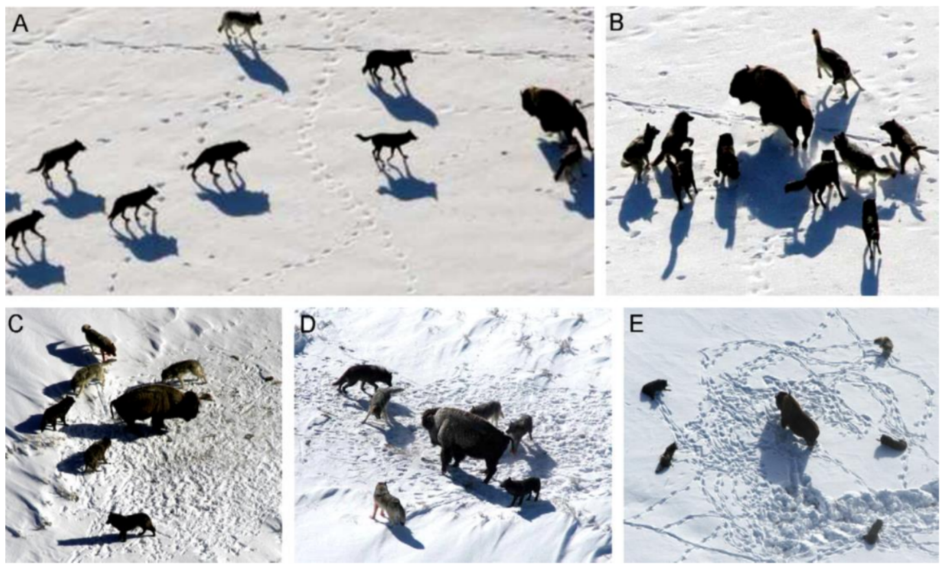


Fig. 2. Hunting technique of grey wolves in [6]

The algorithm is inspired by this process and smoothly transitions between scouting and hunting phases. In the scouting phase, the pack extensively scouts its environment through many random movements so that the algorithm does not get stuck in a local extreme, while in the hunting phase, the influence of random movements is slowly reduced and pack members draw progressively closer to the discovered extreme. To maintain the divergence between those phases, each wolf is assigned vectors **A** and **C**.

$$\mathbf{A} \text{ is a vector with components } rand\,(-1,1) * a,$$

where $rand(-1,1)$ generates a random number between -1 and 1 and where

$$a = 2 - \left(\frac{2i}{i_{max}}\right)$$

while i is the algorithms current iteration, and i_{max} is the maximum number of iterations. It is random value between -2 and 2. You can see its impact in the Fig. 3.

With growing iterations, it is more likely that its value will be between -1 and 1. That makes it more likely for a wolf to be hunting.

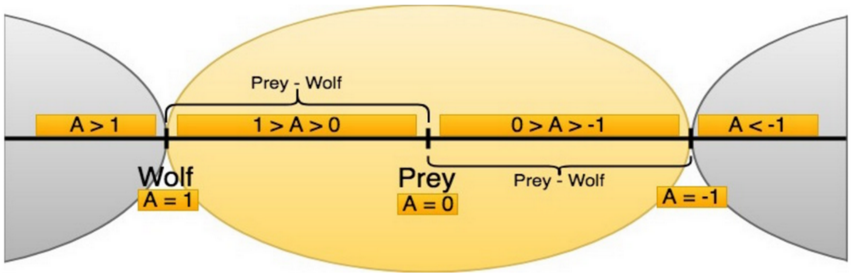


Fig. 3. Vector **A** and its impact in 1D

Another component supporting the scouting phase is vector

$$\mathbf{C} = rand(0,2),$$

where $rand(0,2)$, generates a random number between 0 and 2. Vector **C** is similar to vector **A**, but iterations don't influence it. It helps the wolves behave more naturally. Similarly, in nature, wolves encounter various obstacles which prevent them from approaching prey comfortably.

The vectors **A** and **C** encourage wolves to prefer scouting, or hunting, and so to avoid local optima regardless of the algorithms current iteration.

Wolves' positions within the environment are updated based upon the estimated location of the prey using Alpha, Beta, and Delta wolves as guides.

Let $\mathbf{X_j}(i)$ be a positional vector of wolf j in i-th iteration. Positional vector of wolf j is updated as follows:

$$\mathbf{X_j}\,(i+1) = \frac{\mathbf{X_1} + \mathbf{X_2} + \mathbf{X_3}}{3},$$

where i is the algorithms current iteration and $\mathbf{X_1}$, $\mathbf{X_2}$, $\mathbf{X_3}$ are new potential position vectors of Alpha, Beta, and Delta wolves obtained from following formulas:

$$\begin{aligned}\mathbf{X_1} &= \mathbf{X_\alpha}(i) - \mathbf{A_1} * \mathbf{D_\alpha}\\ \mathbf{X_2} &= \mathbf{X_\beta}(i) - \mathbf{A_2} * \mathbf{D_\beta}\\ \mathbf{X_3} &= \mathbf{X_\delta}(i) - \mathbf{A_3} * \mathbf{D_\delta}\end{aligned}$$

where $\mathbf{X_\alpha}(i)$, $\mathbf{X_\beta}(i)$, $\mathbf{X_\delta}(i)$ are the position vectors of Alpha, Beta, and Delta wolves representing positions within the environment that are closest to the optimum in i-th iteration. Vectors $\mathbf{A_1}, \mathbf{A_2}, \mathbf{A_3}$ are calculated in the same way as vector $\mathbf{D_\alpha}, \mathbf{D_\beta}, \mathbf{D_\delta}$ are vectors defining the distance of the wolf j position from the prey as follows:

$$\begin{aligned}\mathbf{D_\alpha} &= |\mathbf{C_1} * \mathbf{X_\alpha}(i) - \mathbf{X_j}(i)|\\ \mathbf{D_\beta} &= |\mathbf{C_2} * \mathbf{X_\beta}(i) - \mathbf{X_j}(i)|\\ \mathbf{D_\delta} &= |\mathbf{C_3} * \mathbf{X_\delta}(i) - \mathbf{X_j}(i)|\end{aligned}$$

where $|\mathbf{X}|$ is the vector whose components are the absolute values of the components of $\mathbf{X}$. Vectors $\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}$ are computed in the same way as vector $\mathbf{A}$ and influences the weight of the preys estimated position $\mathbf{X_\alpha}, \mathbf{X_\beta}, \mathbf{X_\delta}$ (increasing or decreasing it).

If wolves have the tendency to move closer towards prey, they will begin to encircle it (wolves approach from various directions), as you can see it in Fig. 4.

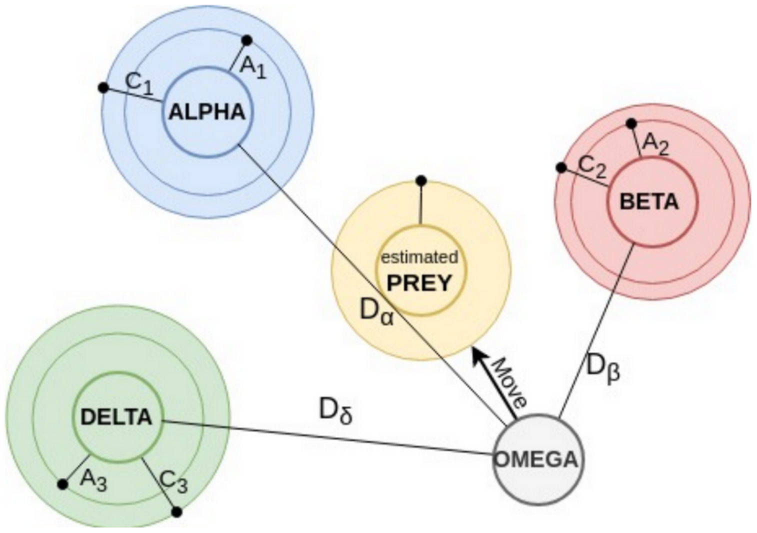


Fig. 4. Positional updates of Omega wolves as it is described by the mathematical formula

2.1 Algorithm pseudocode

In this subsection we describe the algorithm in pseudocode. Algorithms inputs are dimensions of the problems environment, boundaries of the problems environment, fitness function characterising the problem, size of the pack (number of wolves/agents), number of iterations of the algorithm, termination criteria and criteria of the fitness function.

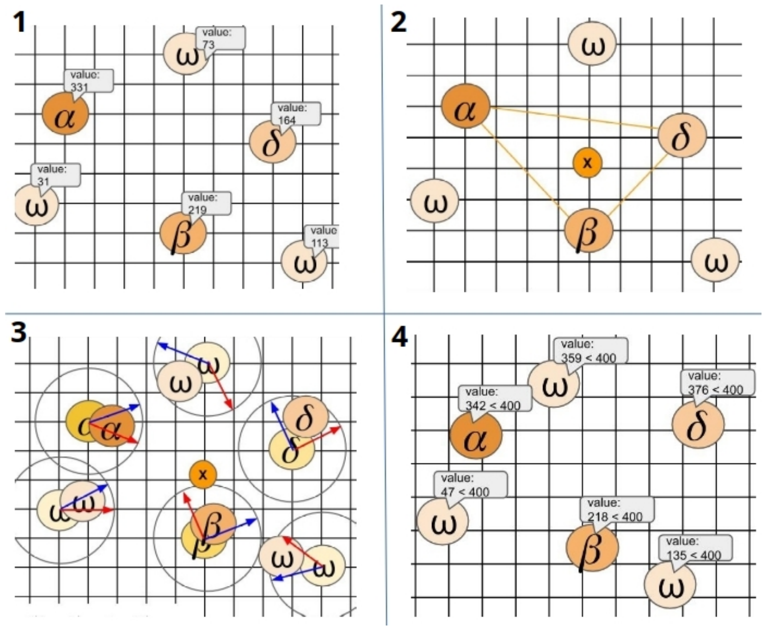


Fig. 5. Algorithm steps

The algorithms pseudocode follows:

- In the first step, agents (wolves) are randomly spread out across the environment.
- In each iteration i:
  - calculate the fitness value of each agent and determine the social hierarchy – Fig. 5. part 1. The agent with the best value (closest to the optimum) is Alpha, second best is Beta, third best is Delta, and all others are Omega.
  - calculate the best solution found thus far by Alpha, Beta and Delta ($\mathbf{X}_{\boldsymbol{\alpha}}(i)$, $\mathbf{X}_{\boldsymbol{\beta}}(i)$, $\mathbf{X}_{\boldsymbol{\delta}}(i)$ and average it – Fig. 5. part 2,
  - update positions of all the wolves $X_j(i+1)$, while random vectors $\mathbf{A}$ and $\mathbf{C}$ are updated for each one – Fig. 5. part 3,
  - check the termination criterion – Fig. 5. part 4. Iterations terminate when fitness function value reaches a preset value.

3 2D P Colonies

In this section, we recall the definition of 2D P colonies and other terms related to them.

Definition 1. *A 2D P colony is a construct*

$$\Pi = (A, e, Env, B_1, \dots, B_k, f), k \geq 1, \text{ where}$$

- *A is an alphabet of the colony, its elements are called objects,*
- *$e \in A$ is the basic environmental object of the 2D P colony,*
- *Env is a pair $(m \times n, w_E)$, where $m \times n, m, n \in N$ is the size of the environment and w_E is the initial contents of environment, it is a matrix of size $m \times n$ of multisets of objects over $A - \{e\}$.*
- *B_i, $1 \leq i \leq k$, are agents, each agent is a construct $B_i = (o_i, P_i, [o, p])$, $0 \leq o \leq m$, $0 \leq p \leq n$, where*
  - *o_i is a multiset over A, it determines the initial state (contents) of the agent, $|o_i| = 2$,*
  - *$P_i = \{p_{i,1}, \dots, p_{i,l_i}\}, l \geq 1, 1 \leq i \leq k$ is a finite set of programs, where each program contains exactly 2 rules, which are in one of the following forms each:*
    - *$a \to b$, called the evolution rule, $a, b \in A$,*
    - *$c \leftrightarrow d$, called the communication rule, $c, d \in A$,*
    - *$[a_{q,r}] \to s, 0 \leq q, r \leq 2, s \in \{\Leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$, called the motion rule,*
- *$f \in A$ is the final object of the colony.*

A configuration of the 2D P colony is given by the state of the environment - matrix of type $m \times n$ with multisets of objects over $A - \{e\}$ as its elements, and by the state of all agents - pairs of objects from alphabet A and the coordinates of the agents. An initial configuration is given by the definition of the 2D P colony.

A computational step consists of three parts. The first part lies in determining the set of applicable programs according to the current configuration of the 2D P colony. In the second part, we have to select from this set one program for each agent, in such a way that there is no collision between the communication rules belonging to different programs. The third part is the execution of the chosen programs.

A change of the configuration is triggered by the execution of programs and it involves changing the state of the environment, contents and placement of the agents.

A computation is non-deterministic and maximally parallel. The computation ends by halting when there is at least one agent that has no applicable program.

The result of the computation is the number of copies of the final object placed in the environment at the end of the computation.

The aim of introducing 2D P colonies is not studying their computational power but monitoring their behaviour during the computation.

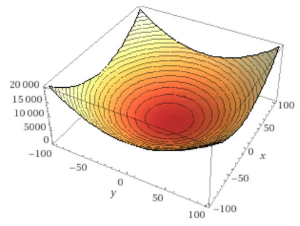
4 Modeling of Grey wolf algorithm using P colonies

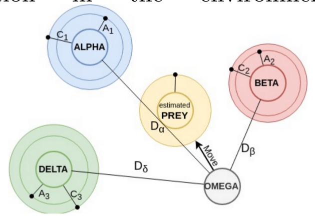
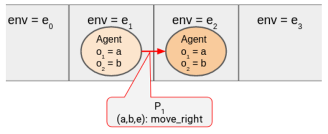
As for the modeling of GWO, some similarities as well as a few differences have been found. Both are inspired by the nature, usable for solving optimisation problems, and both are multiagent system models. For comparison see the differences / problems:

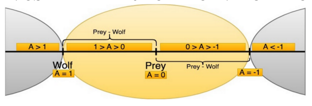
- *Environmental problem*,
- *Communication problem*,
- *Randomness problem.*

These problems are described in the Table 1.

Table 1. Differences between Grey wolf algorithm and P colony tables.

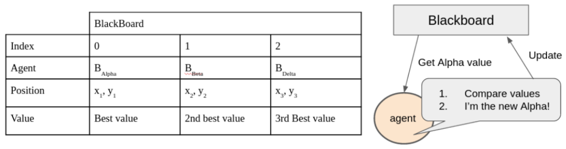
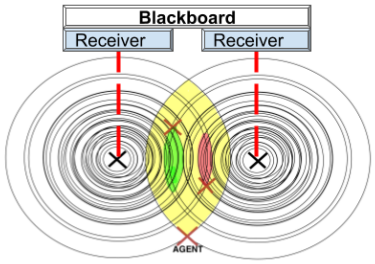
Difference / System	Grey wolf algorithm	P colony
Environmental problem	The environment is represented by a mathematical fitness function. 	The environment is represented by a multiset of symbols. $Env = (6 \times 6, w_E)$, $w_E = \begin{bmatrix} D & D & D & D & D & D \\ D & S & S & D & D & D \\ D & S & S & D & D & D \\ D & D & D & S & S & D \\ D & D & D & S & S & D \\ D & D & D & D & D & D \end{bmatrix}$,

Communication problem	The agents have the knowledge of their global position in the environment.  ALPHA
	C1
	A1
	estimated PREY
	Dα
	BETA
	C2
	A2
	Dβ
	Move
	DELTA
	A3
	C3
	Dδ
	OMEGA
		They are communities of simple reactive agents independently living and acting in a joint shared environment.  env = e0
		env = e1
		Agent
		o1 = a
		o2 = b
		env = e2
		Agent
		o1 = a
		o2 = b
		env = e3
		P1
		(a,b,e): move_right

Randomness problem	Random vectors A and C influence the movement of wolves in the environment.  Prey Wolf A > 1 1 > A > 0 0 > A > -1 A < -1 Wolf A = 1 Prey A = 0 Prey Wolf A = -1	Each rule is deterministic, the only way to implement randomness is to randomise the choosing rule for identical configurations. P_1: 1. (a, a, e): action_1 2. (a, a, e): action_2 ?

2D P colony definition needs to be adjusted to meet the described requirements. Proposed solutions for those problems are described in Table 2.

Table 2. Proposed solution of differences / problems found

Difference / Problem	Proposed solution
Environmental problem	The environment will be a pair of matrix $m \times n$ and fitness function $f(x)$, where $m \times n$, $m, n \epsilon N$ is the size of the environment and $f(x)$ is a mathematical function with the initial contents of environment. Alphabet A will contains real numbers and environmental symbol, $A = \{\mathbb{R}\} \bigcup \{e\}$, The rules of the programs, which guide the agents, will compare the number values of objects using operators smaller " $<$ " or greater " $>$ " then, agents B_i will be defined as $B_i = o_i$, P_i, $[r_x, r_y]$, $o_i = 2$. *Example:* *Env* = (6×6, *f(x)*), *f(x)* = $\begin{pmatrix} 3 & 12 & 17 & 22 & 14 & 4 \\ 8 & 24 & 28 & 31 & 19 & 11 \\ 14 & 27 & 33 & 37 & 31 & 15 \\ 21 & 25 & 41 & 48 & 30 & 18 \\ 15 & 21 & 32 & 33 & 26 & 10 \\ 7 & 11 & 19 & 21 & 9 & -2 \end{pmatrix}$
Communication problem	We extend the P system by adding the Blackboard that saves the agents' best fitness values and is always accessible to read and write by all agents. Agents must know their position in the environment, because the fitness value is not enough for calculating the preys estimated position. 
Randomness problem	We will use the blackboard as means of giving feedback to agents. Agents do not need to know their position in the environment, all agents who can contribute to the search will send solution to the blackboard points called receivers, and estimation of prey position is calculated as average of distances collected by blackboard points from wolves Alpha, Beta and Delta. Omega wolves can ping the blackboard if changing position and get their distance from the prey. If the distance would decrease compared to the original distance, then the wolf will move. We plan to introduce randomness by only using two blackboard receivers and rounding the wolves' distances from the prey. 

	BlackBoard		
Index	0	1	2
Agent	B_{Alpha}	B_{Beta}	B_{Delta}
Position	x_1, y_1	x_2, y_2	x_3, y_3
Value	Best value	2nd best value	3rd Best value

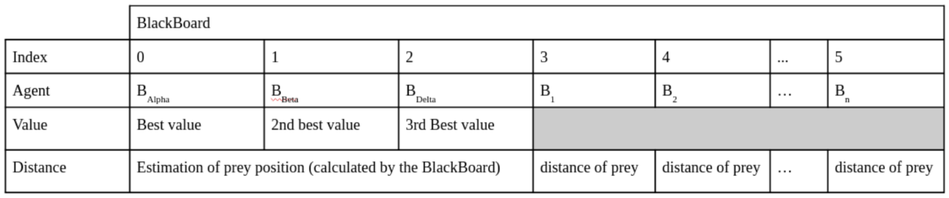
5 Proposed model of 2D P colony with the blackboard

As for the proposed definition of 2D P colony model with the blackboard, it is adapted to the suggestions from the previous chapter so that it allows simulation of the Grey wolf algorithm.

5.1 Definition

P_{gw} = (A, e, env, $B_1, B_2, \ldots, B_n$, X, f), where:

- $A = \{\mathbb{R}\} \bigcup \{e, m, f\}$,
- $e \in A$ is the basic environmental object,
- env is a pair $(m \times n, f(x))$, where $m \times n, m, n \in \mathbb{N}$,
- $B_1, B_2, \ldots, B_n$ are the agents, $B_i = (O_i, P_i, [r_x, r_y])$, where:
  - $O_i = 2$,
  - $P_1 = P_2 = ... = P_n$, P_i rules are defined below,
  - r_x, r_y are the initial coordinates,
- X is the blackboard defined as structure in Fig. 6.,
- f is the final object, $f \in A$.



	BlackBoard						
Index	0	1	2	3	4	...	5
Agent	B_{Alpha}	B_{Beta}	B_{Delta}	B_1	B_2	…	B_n
Value	Best value	2nd best value	3rd Best value	▨	▨	▨	▨
Distance	Estimation of prey position (calculated by the BlackBoard)			distance of prey	distance of prey	…	distance of prey

Fig. 6. Blackboard structure

The initial agents' configuration is: $(O_1[e], O_2[e], env[i])$, where $i \in \mathbb{N}$.
Programs P_i associated with i-th agent are:

1. $(e_1, e_2, x) : e_1, e_2 \leftrightarrow x$; $x \in \mathbb{R}$
2. (x, y, e); $x, y \in \mathbb{R}$:
   a) $y \leftarrow Get(BB[Alpha])$ and $Compare(x, y)$:
      i. $x > y$: I'm new Aplha, $Update(BB[Alpha])$;
      ii. $x < y$: $y \leftarrow Get(BB[Beta])$ and $Compare(x, y)$:
         A. $x > y$: I'm new Beta, $Update(BB[Beta])$;
         B. $x < y$: $y \leftarrow Get(BB[Delta])$ and $Compare(x, y)$:
            - $x > y$: I'm new Delta, $Update(BB[Delta])$;
            - $x < y$: I'm Omega: $y \leftrightarrow e$; $e \rightarrow m$; $m \leftrightarrow y$;
3. (x, y, m); $x, y \in \mathbb{R}$:
   a) $PingBB[i]$ and $y \leftarrow Get(BB[i])$;

```
b) $PingBB[i] + mv1$; $mv1 = (\Leftarrow, \Rightarrow, \Uparrow, \Downarrow)$ and $y \leftarrow Get(BB[i])$;
c) $Compare(x, y)$;
   i. $x > y$: $Do(mv1)$;
   ii. $x < y$:
      A. $PingBB[i] + mv2$; $mv2 = (\Leftarrow, \Rightarrow, \Uparrow, \Downarrow) - mv1$; $mv1 \neq mv2$
      B. $y \leftarrow Get(BB[i])$ and $Compare(x, y)$;
         • $x > y$: $Do(mv2)$;
         • $x < y$: ... (try it with $mv3$ and $mv4$)
           – Can't move: $y \leftrightarrow m$; $m \rightarrow f$; $f \leftrightarrow m$;

4. $(x, y, f)$; $x, y \in \mathbb{R}$: Stop the agent. }
```

P_i rules definition use the following symbols:

- $\leftarrow$ means Get from the blackboard,
- $\rightarrow$ means Rewrite agent's object,
- $\leftrightarrow$ means Change agent's object with environment object,
- $" \Leftarrow " = LEFT$, $" \Rightarrow " = RIGHT$, $" \Uparrow " = UP$, $" \Downarrow " = DOWN$

At this point it is important to focus on the use of the blackboard by the agents. Agents can use blackboard functions:

- $Get(BB[i])$; $i =$ agent's index - agent i gets its distance from the prey (it is calculated by the blackboard),
- $Update(BB[x])$; $x = Alpha, Beta, Delta$ - agent update B_{alpha}, B_{beta}, or B_{delta} field value,
- $Ping[BB[i]]$; i is the agent's index - agent can ping the blackboard from some position, its distance from the prey is recalculated.

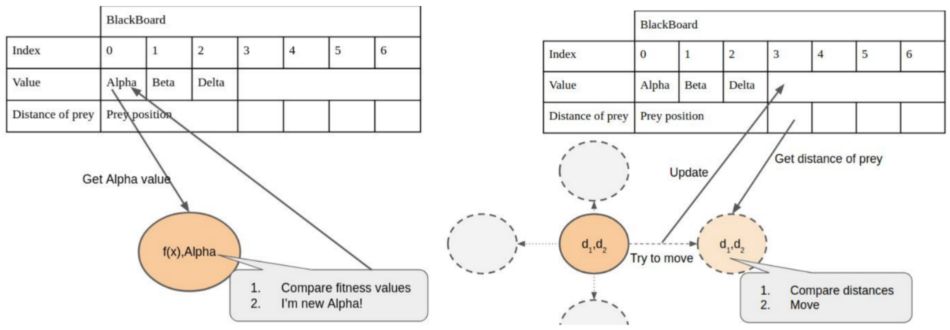


Fig. 7. Blackboard in use

The agent concludes it is Alpha and it rewrites the corresponding blackboard field in Fig. 7. on the left side. On the right side of Fig. 7., the agent concludes it is Omega and it will try to move with blackboard's assistance.

Finally, the following derivation can be created. Fig. 8., a sequence of derivation steps is depicted.
The first step (iteration) (point 1 in Fig. 8.) starting with two agents randomly initialized into the environment. They are in the initial configuration - two objects e inside the agent.
In the second iteration (point 2 in Fig. 8.), agents exchange their objects for objects placed in the environment (computed by fitness function).
In next iterations (point 3 in Fig. 8.), agents get Alpha value from the blackboard and try to compare it to their own value. If Alpha value is empty, the first agent to try to compare its value will become the Alpha and update the blackboard. If more than one agent tries to write value into the blackboard in the same position the winner is non-deterministically chosen.
The same process is used for declaring Beta and Delta agents (point 4 in Fig. 8.). If the agent's fitness value is lower than Delta, this agent becomes Omega. At this point (point 5 in Fig. 8.) this agent rewrites the environmental object to m. Afterwards, it will try to move (point 6 in Fig. 8.). Before moving, however, it will compare the distances. The algorithm iterates until no more movement which would improve the fitness value is possible.

6 Conclusion

In this paper we introduce extended model of 2D P colonies that can simulate solving optimization problem by Grey wolf optimization algorithm. The model will be improved in the near future. It is assumed that the function $Compare()$ can be replaced by special equivalent rules, like $(x > y, e) : action$. It is also assumed that the other functions, such as $Do(mv1)$, can be replaced by the rules in a form closer to the common rules of P systems.

When it comes to testing the model and comparing it with the original version of GWO, the observation of the behaviour of the modified model is the aim of our further work. Potentially, the modified model could be faster or more efficient (in its approach towards the divergence between scouting for prey and hunting) compared to the original algorithm.

Acknowledgments.

The work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602 and by the Silesian University in Opava under the Student Funding Scheme, project SGS/11/2019.

References

1. Cienciala, L., Ciencialová, L., Perdek, M.: 2D P colonies. In: Csuhaj-Varjú E., Gheorghe M., Rozenberg G., Salomaa A., Vaszil Gy. (eds) Membrane Computing. CMC

2012. Lecture Notes in Computer Science, vol 7762. Springer, Berlin, Heidelberg, pp. 161–172 (2012)
2. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., and Sosík, P.: P colonies. J Membr Comput 1, 178—197 (2019)
3. Csuhaj-Varjú, E., Kelemen, J., Păun, Gh., Dassow, J.(eds.): Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994)
4. Kelemen, J., Kelemenová, A.: A Grammar-Theoretic Treatment of Multiagent Systems. Cybern. Syst. 23(6), 621–633 (1992)
5. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX). pp. 82–86. Boston, Massachusetts, USA (September 12-15 2004)
6. Muro, C., Escobedo, R., Spector, L., Coppinger, R.P.: Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations, Behavioural Processes 88(3),
192–197 (2011)
7. Păun, Gh.: Computing with membranes. J. Comput. Syst. Sci. 61(1), 108–143 (2000)
8. Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
9. Mirjalilia, S., Mirjalilib, S.M., Lewisa, A.: Grey Wolf Optimizer. Advances in Engineering Software 69, 46—61(2014)

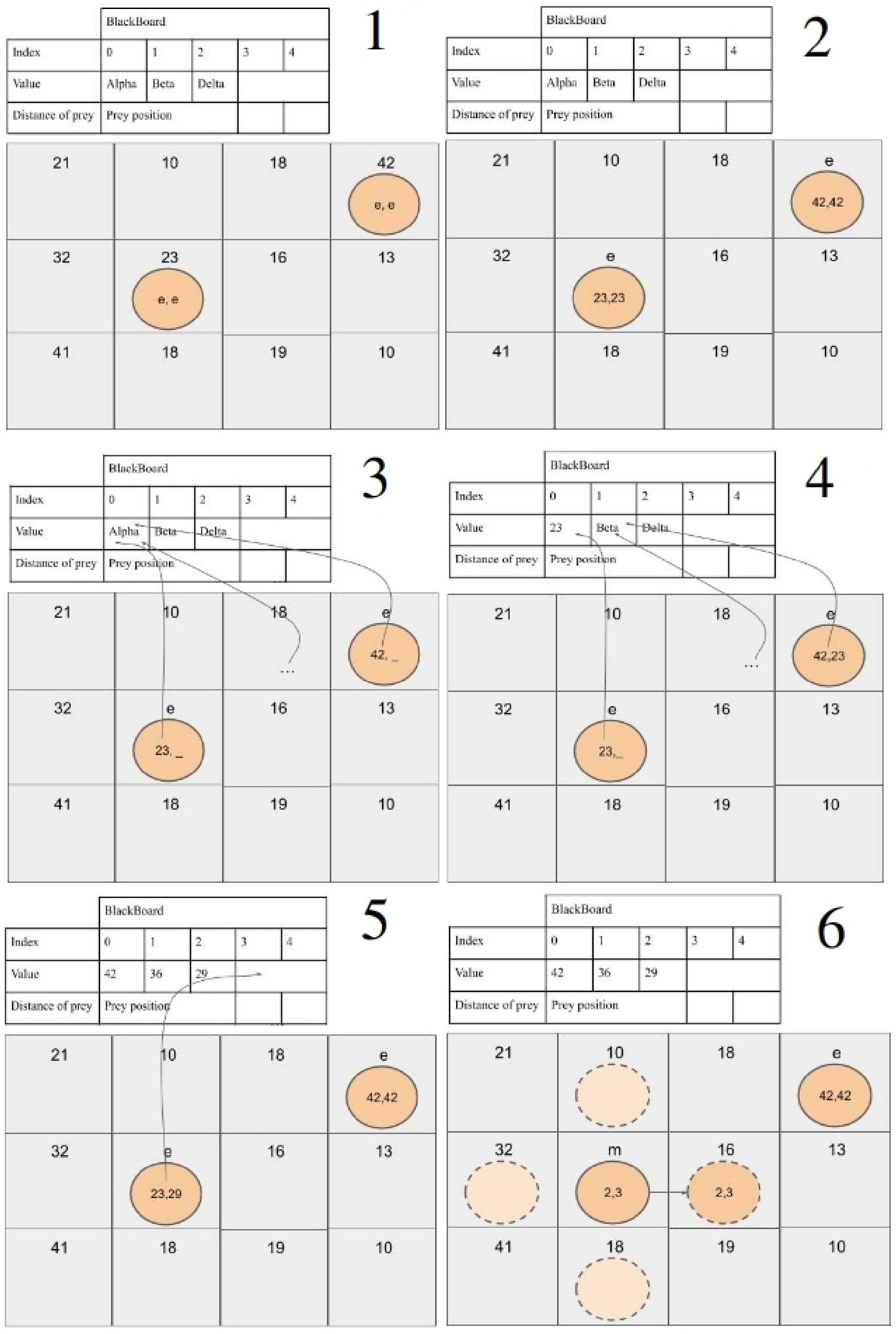


Fig. 8. Example of derivation

Author Index